Annex 5 for D2.3	SSP support for Papyrus
Access[1]:	**PU**
Type[2]:	**Prototype**
Version:	**2.0**
Due Dates[3]:	**M24, M36**

Open Cyber-Physical System Model-Driven Certified Development

Executive summary[4]:

Annex 5 for D2.3 focuses on the tool support developed for the `System Structure and Parameterization` (SSP) standard for defining the input models of the master simulation tool.

[1] Access classification as per definitions in PCA; PU = Public, CO = Confidential. Access classification per deliverable stated in FPP.
[2] Deliverable type according to FPP, note that all non-report deliverables must be accompanied by a deliverable report.
[3] Due month(s) according to FPP.
[4] It is mandatory to provide an executive summary for each deliverable.

Deliverable Contributors:

	Name	Organisation	Primary role in project	Main Author(s)[5]
Deliverable Leader[6]	Ákos Horváth	IncQuery Labs	Task leader of the annex	X
Contributing Author(s)[7]	Rebeka Farkas	IncQuery Labs	Contributor	X
	Krisztián Mócsai	IncQuery Labs	Contributor	X
	Zoltán Ujhelyi	IncQuery Labs	Contributor	X
	Dániel Segesdi	IncQuery Labs	Contributor	X
Internal Reviewer(s)[8]	Sebastien Revol	CEA	Reviewer	

Document History:

Version	Date	Reason for Change	Status[9]
0.1	10/11/2017	First Draft Version	Draft
1.0	17/11/2017	Final version	Final
1.1	17/11/2017	Modification done based on feedback	Final
2.0	28/11/2018	Updated version based on the newly developed features	Final

[5] Indicate Main Author(s) with an "X" in this column.
[6] Deliverable leader according to FPP, role definition in PCA.
[7] Person(s) from contributing partners for the deliverable, expected contributing partners stated in FPP.
[8] Typically person(s) with appropriate expertise to assess deliverable structure and quality.
[9] Status = "Draft", "In Review", "Released".

CONTENTS

ABBREVIATIONS

List of abbreviations/acronyms used in document:

Abbreviation	**Definition**
FMI	Functional Mock-up Interface
FMU	Functional Mock-up Unit
M&S	Modelling and Simulation
N/A	Not Applicable
SotA	State of the Art
TBD	To Be Defined

1 OVERVIEW

1.1 Introduction

The goal of this module is to provide support for a standardized FMU composition format in Papyrus. Currently the majority of the OpenCPS members who are working on the Master Simulation Tool (MST or OMSimulator) want to support the `System Structure and Parameterization` (SSP) soon-to-be standard that is developed by the Modelica Association, which makes it a desired feature of Papyrus to be able to directly work with ssp files.

This document is structured as follows. Section 1 provides some background knowledge about SSP and Papyrus and explains the basics of out solution to the SSP support in Papyrus, Section 2 presents the architecture, including technologies we used in our software and the theoretical basics of the used transformations, Section 3 provides a user guide that explains the usage of our software from the installation and provides an example and Section 4 concludes this document.

M36

The final version of the Papyrus FMU-SSP simulation configuration module uses SysML block definition diagrams for defining the structure of the simulation scenarios instead of the UML class diagram as provided in M24 in order to support the demonstrators using SysML for architecture descriptions (T6.3 and T6.5). Additionally, we extended the module to include support for exchanging FMU's within an SSP simulation configuration and also provide support of handling Buses through a special type of Annotations. These extensions provided the capability to use our module for configuring the OMSimulator through and SSP descriptor.

Due to timing constrains within the project, we have kept the original package names for all our source code, thus many still contains the uml string.

1.2 SSP

The SSP working draft with the documentation, examples and the schema definition (xsd) for the used xml descriptor can be accessed in the OpenCPS SVN

The SSP file shows much similarity with an FMU file, as it is also an archive that packs the components of the system and a system description in xml. This xml file contains the structure of the system described by mainly components, connectors (ports) and connections between these ports. Additionally the descriptor contains information about the parameterization of the inner components, and it may also contain graphical information.

1.3 Papyrus

Papyrus is an open source project to provide an integrated environment for editing UML and SysML models. Additionally to graphical support, Papyrus also provides advanced functions for model-based development, such as simulation and code generation. For more information on Papyrus please visit the Papyrus website.

1.4 Model transformations

The core of the SSP support in Papyrus is a two-way model transformation: a batch transformation that processes the SSD file and creates the corresponding SysML model (block definition diagram), and a live transformation that monitors the SysML model and diagram changes, instantly modifies the in-memory SSD model, and saves it when the Papyrus model is saved.

The workflow starts by parsing the SSD model to its EMF model. Then the SysML components are created and linked to the parent FMUs. While the event-driven transformation is running, every time the corresponding SysML changes, the modified versions are transformed back to SSP. The SysML can be edited through the Papyrus diagram editor.

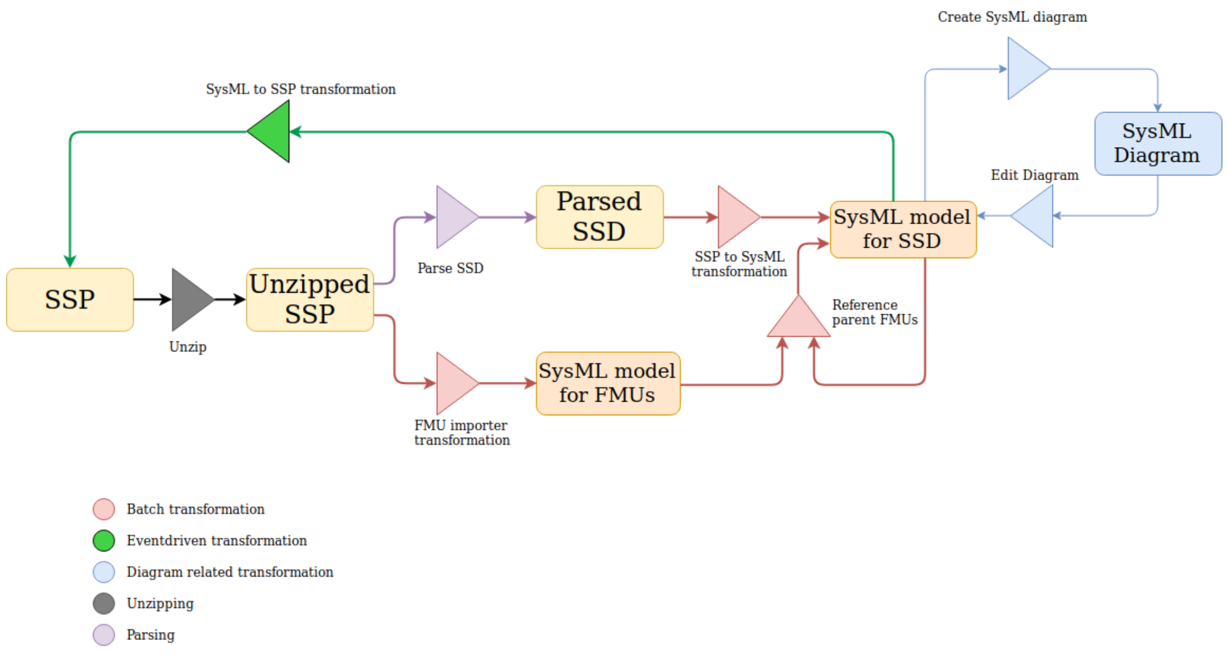


Figure 1. The complete SSP workflow

2 ARCHITECTURE

2.1 Environment

This application is implemented as a plugin for the popular Eclipse framework.

2.1.1 Eclipse

Eclipse is a Java-based IDE with an extensible plug-in system, that allows the user to customize the environment as well as to develop and install their own plug-ins. The developed plugin can be installed to the Eclipse instance or - during earlier development phases - another, temporary Eclipse can be created (the so-called *Runtime Eclipse*) that has the plugin installed, but disappears when closed (however, the workspace is saved so the user doesn't lose files). For more information on Eclipse please visit the Eclipse website.

Many applications are based on Eclipse, including Papyrus. This software is also based on an Eclipse and uses its aforementioned functionalities (see: Section 3).

2.1.2 Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) is also based on Eclipse. It provides a modeling environment, that allows the user to design class diagram-like metamodels and a code generator that generates Java structures (classes, interfaces, etc.) from the model. This improves the simplicity of creating complex data structures. For more information on Eclipse Modeling Framework please visit the EMF website.

This software uses EMF to parse and manage SSP models.

2.1.3 VIATRA

The Eclipse VIATRA framework supports the development of model transformations including event-driven transformations. VIATRA offers a language to define transformations and a reactive transformation engine to execute certain transformations upon changes in the underlying model. Transformations are defined by transformation rules. For more information on VIATRA please visit the VIATRA websie.

This software uses VIATRA for describing and executing both the SSP to SysML and the SysML to SSP transformations.

2.2 Transformations

2.2.1 Parsing the EMF model

First, the unzipped SSD model is parsed to EMF and the corresponding SSP model is created. This is performed by the *SSPParser* that traverses the SSD and creates the model elements.

2.2.2 SSP to SysML transformation

This transformation traverses the elements of an SSD, and hierarchically creates the corresponding elements in SysML. This is a batch transformation designed to be executed once at the beginning of the workflow.

It consist of of four rules:

- The `SsdRule` applies the profile to the model, transforms the root TSystem, creates the resource for traces and adds the first trace (TSystem). It also starts the subsequent rules.
- The `ConnectorRule` transforms the connectors (Ports) of the root TSystem, applies stereotype and adds trace.

- The `SystemComponentRule` transforms the child components of the root TSystem, applies stereotype, adds trace and calls *ConnectorRule* to transform the connectors of these components.
- The `ConnectionRule` transforms all the connections, applies stereotype with the corresponding properties and adds trace.

2.2.3 SysML to SSP transformation

This transformation incrementally converts the changes on the SysML model back to the SSD. This is an event-driven transformation, designed to keep the SSP model updated with the SysML model. It is also based on model transformation rules.

There are two types of changes: removals and modifications (including the creation of new elements).

Removals are taken care of by the `TraceRule`, which fires if anything, that has a trace, is deleted from the SSD's SysML model. It deletes the elements from the SSD and traces from the corresponding models.

The rules fired after modifications depend on the stereotype. For instance, the `ConnectionRule` transforms *Connections* back to SSD and sets certain properties from the stereotype. Similarly, *Ports* and *Components* are transformed by the `PortRule` and the `ComponentRule`. The `TSystemRule` modifies both the root *TSystem* and the *SystemStructureDescriptionType* element in SSD.

When an element is created, the rules add traces to the `Trace` model, except for the `TSystemRule`, where creation is not supported.

2.2.4 Referencing parent FMUs

This is a batch transformation, that goes through every component in SysML, searches for its parent FMU (in SysML), and sets it onto the stereotype property of the component.

2.3 Project structure

- **SSP** (`org.eclipse.papyrus.moka.ssp`)

  This Eclipse project provides the support for parsing SSP models to EMF. It contains the EMF metamodel, and the parser classes.

- **Common** (`org.eclipse.papyrus.moka.ssp.common`)

  This Eclipse projects contains classes that are commonly used by many other projects that handle different domains. The `utils` package contains useful functions for the transformation.

  - Class `EditingDomainHelperUtility` contains a function, in which other functions can run through an editing domain via a Command.

  - Class `TraceHelperUtility` helps creating and adding traces.
- **Profile** (`org.eclipse.papyrus.moka.ssp.profile`)

  This project contains the SysML profile.

  Class `StereotypeStrings` in the root package contains fully qualified names of the stereotypes and stereotype properties (since it is not generated, and functions related to stereotypes need fully qualified names as parameter).

- **Trace** (`org.eclipse.papyrus.moka.ssp.trace`) This project contains the EMF metamodel for Traces. This metamodel helps to connect the logically same model elements between the SSD and the UML models.

  Each SSD element has *one* element in UML and each UML element has *one* SSD element in UML, but the root TSystem (in UML) has *two*, the `SystemStructureDescriptionType` and the root `TSystem` itself.

- **SysML to SSP transformation** (`org.eclipse.papyrus.moka.ssp.uml2ssp`)

  This project is responsible for the SysML to SSP transformation. It contains the common functions, the transformation rules discussed above, a data structure class and everything else that belongs to this transformation. We kept the original package name for this module.

- **SSP to SysML transformation** (`org.eclipse.papyrus.moka.ssp2uml`)

  Similarly to the project above, this project is responible to the SSP to UML transformation. It has the required classes for the transformation as well as the classes responsible for editing the UML model through the diagram. We kept the original package name for this module.

- **UI** (`org.eclipse.papyrus.moka.ssp2uml.ui`)

  This project is responsible for the SysML diagram editor. It contains the stylesheet for the diagram, an `.ini` file that sets the default theme, and classes that help preparing and starting SSP to SysML transformations. We kept the original package name for this module.

3 USER GUIDE

3.1 Installation

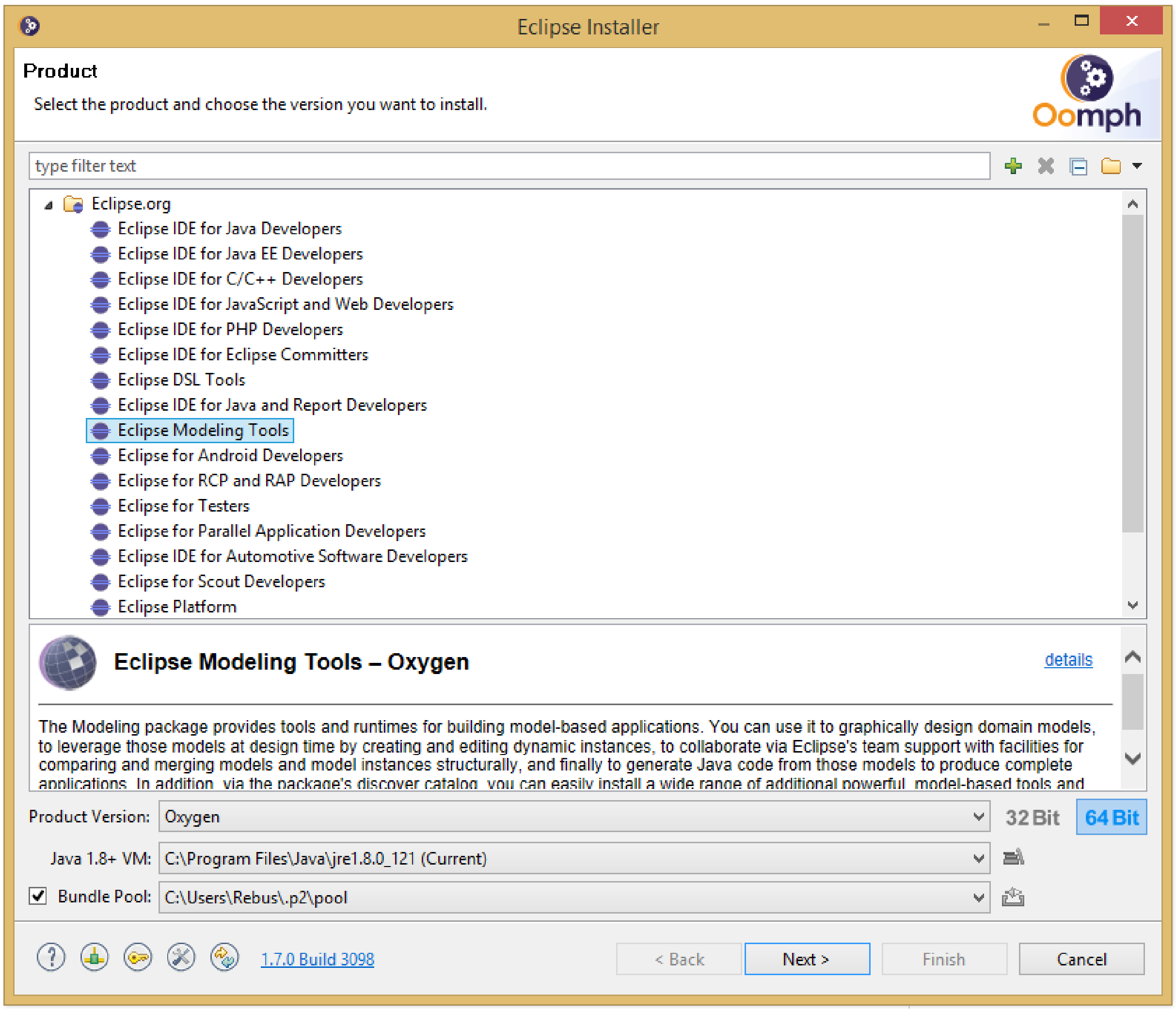
The project environment requires Java to be installed. It can be downloaded from Oracle's website.

There is no update-site yet, where an Eclipse with the required plugins can be downloaded. Instead, this guide uses the Eclipse Installer, which can be downloaded from https://www.eclipse.org/downloads/.

The project also relies on Papyrus-moka, so you will also need to clone this repository.

3.2 Downloading Eclipse

1. Run the Eclipse Installer in *advanced mode*. This can be achieved by clicking on the three horizontal lines in the upper right corner and selecting `ADVANCED MODE`.
2. The first task is to choose the Eclipse distribution to install.
    a. Make sure everything is up-to-date, by clicking on the *Install available updates* button, next to the *Back* and *Next* buttons on the bottom.
    b. From the list of products, choose `Eclipse Modeling Tools`.
    c. In the lower part of the screen select product version `Oxygen`, and make sure the settings are correct.
    d. Finally, hit *Next*



Product

Select the product and choose the version you want to install.

- Eclipse.org
    - Eclipse IDE for Java Developers
    - Eclipse IDE for Java EE Developers
    - Eclipse IDE for C/C++ Developers
    - Eclipse IDE for JavaScript and Web Developers
    - Eclipse IDE for PHP Developers
    - Eclipse IDE for Eclipse Committers
    - Eclipse DSL Tools
    - Eclipse IDE for Java and Report Developers
    - Eclipse Modeling Tools
    - Eclipse for Android Developers
    - Eclipse for RCP and RAP Developers
    - Eclipse for Testers
    - Eclipse for Parallel Application Developers
    - Eclipse IDE for Automotive Software Developers
    - Eclipse for Scout Developers
    - Eclipse Platform

Eclipse Modeling Tools – Oxygen

The Modeling package provides tools and runtimes for building model-based applications. You can use it to graphically design domain models, to leverage those models at design time by creating and editing dynamic instances, to collaborate via Eclipse's team support with facilities for comparing and merging models and model instances structurally, and finally to generate Java code from those models to produce complete applications. In addition, via the package's discover catalog, you can easily install a wide range of additional powerful, model-based tools and

Product Version: Oxygen

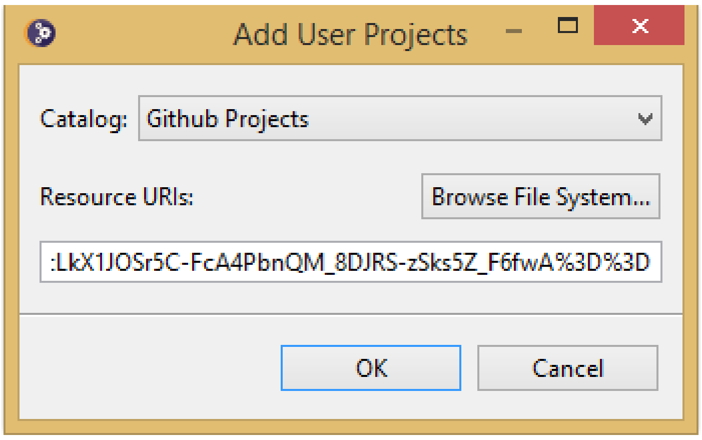
Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_121 (Current)

☑ Bundle Pool: C:\Users\Rebus\.p2\pool

Figure 2. Eclipse Installer 1

3. The next task is to import the eclipse projects you will use.
    a. Click the green + button in the upper right corner.
    b. In the appearing window choose GitHub projects from the drop-down menu.

c. The required *Resource URI* can be retrieved the following way.
   i. Make sure you are logged into GitHub and have an access to the repository.
   ii. Open this link and copy the URL from the browser (it will have your token at the end).
   iii. Copy the link and paste it to the Eclipse Installer.


Add User Projects

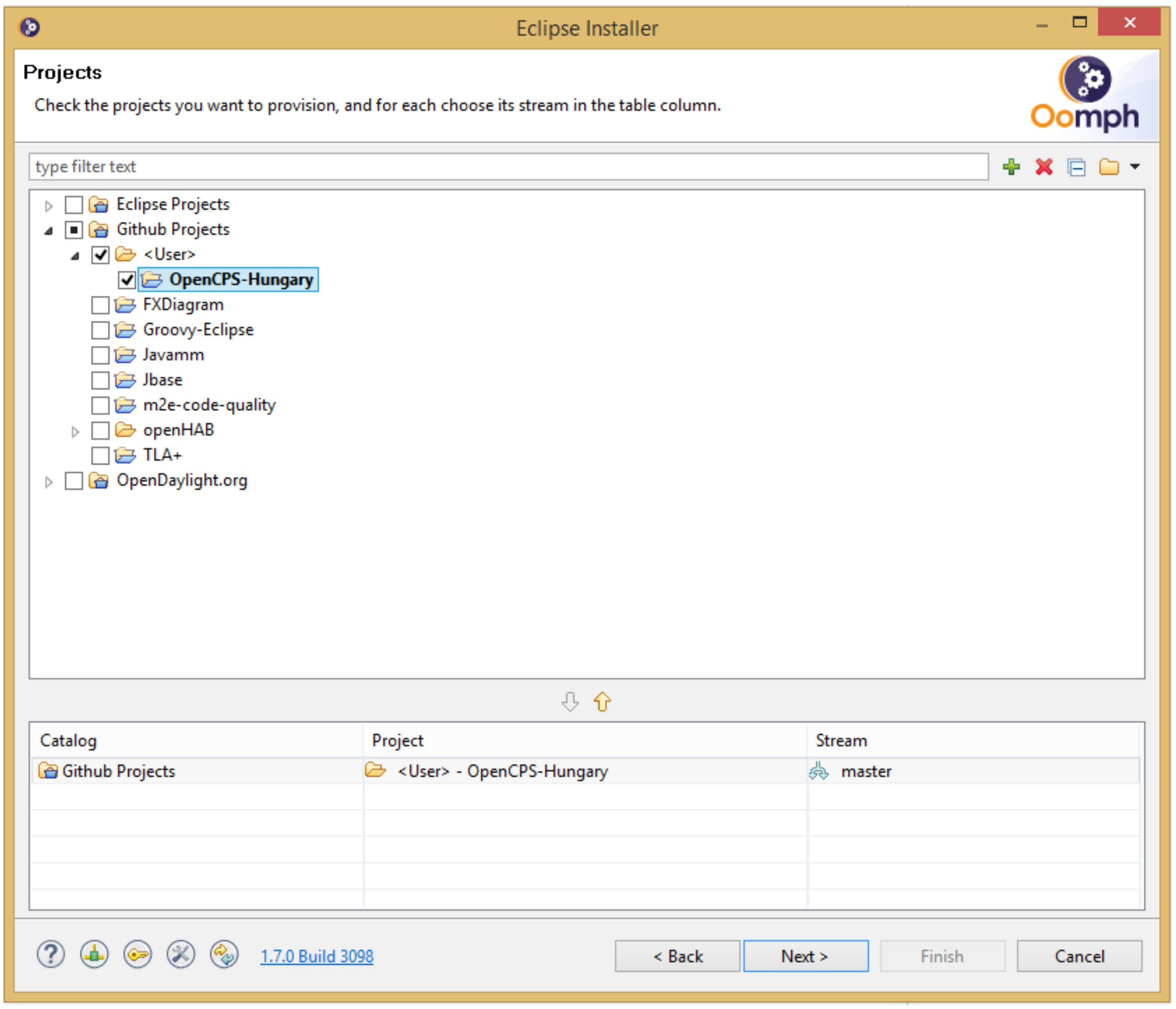
Catalog: Github Projects

Resource URIs:

:LkX1JOSr5C-FcA4PbnQM_8DJRS-zSks5Z_F6fwA%3D%3D

Figure 3. Eclipse Installer 2

d. Hit `OK`.
e. You'll see, that a new folder, named `<User>`, has been created in the GitHub category, which has a project named `opencps-hungary`. Select it.
f. Finally, hit *Next*.


Eclipse Installer

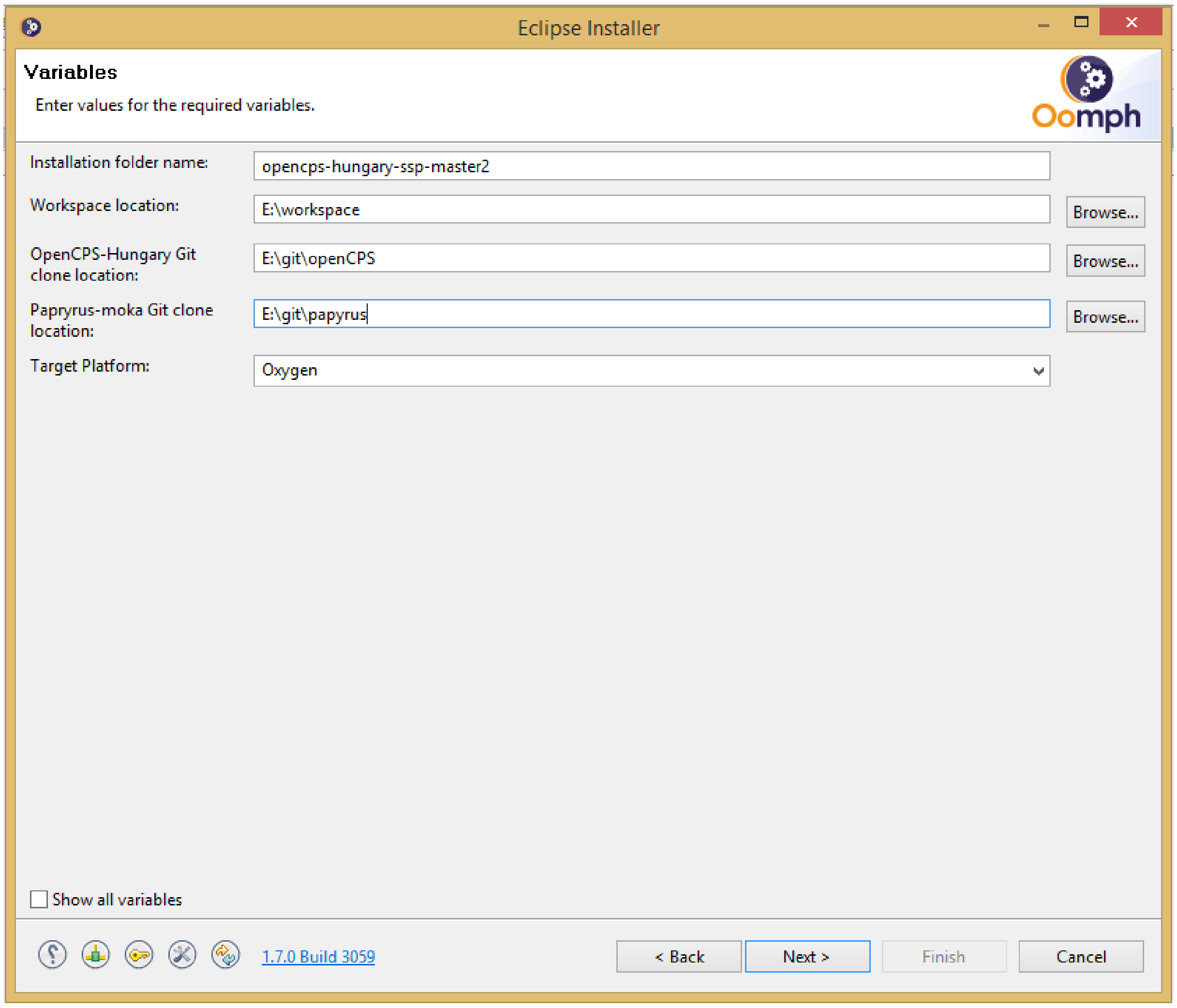
Projects

Check the projects you want to provision, and for each choose its stream in the table column.

Catalog	Project	Stream
Github Projects	<User> - OpenCPS-Hungary	master

Figure 4. Eclipse Installer 3

4. The next task is to select where you want to keep the resources for Eclipse. If you select `Show all variables` you can set the paths of more resources in a more advanced way, otherwise the defaults will be set.
   a. Choose a folder to install Eclipse.
   b. Set a *workspace* folder. Eclipse uses workspaces to organize the Eclipse projects you are working on, many settings for this application will be stored in the selected folder, however the projects themselves will be imported from the Git clone location.
   c. Browse your folders and set the one where you cloned the GitHub repository of this project.
   d. Browse your folders and set the one where you cloned the GitHub repository of *Papyrus*.
   e. Set the target platform to *Oxygen*.
   f. Finally, hit *Next*.



Variables

Enter values for the required variables.

Installation folder name:	opencps-hungary-ssp-master2
Workspace location:	E:\workspace
OpenCPS-Hungary Git clone location:	E:\git\openCPS
Papyrus-moka Git clone location:	E:\git\papyrus
Target Platform:	Oxygen

☐ Show all variables

Figure 5. Eclipse Installer 4

5. In the appearing window, click *Finish* and the setup process begins. It migtht take up a few minutes, but once it's finished, it will open the newly installed Eclipse Oxygen automatically.

3.3 Setting up the Eclipse

After launching, Eclipse will need to take some time and import the projects. You can see the progress in the down right corner. Once it's done, there will be a lot of errors in the workspace (as you can see in the *Model Explorer* once you close the *Welcome* tab), but the following steps will resolve them.

1. First of all, you have to generate the model code from each genmodel (they are located in the model folder in each project), which are in the following projects:
   - **org.eclipse.papyrus.moka.ssp**
   - **org.eclipse.papyrus.moka.ssp.profile**
   - **org.eclipse.papyrus.moka.ssp.trace**

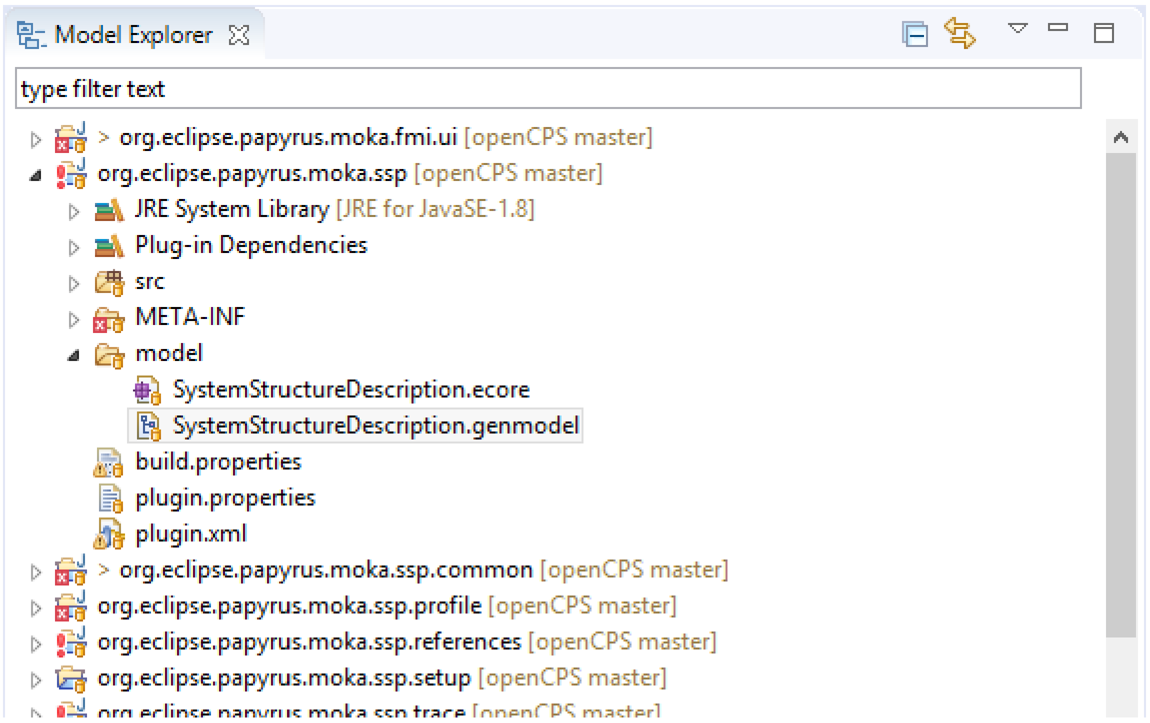

Model Explorer
type filter text
> org.eclipse.papyrus.moka.fmi.ui [openCPS master]
org.eclipse.papyrus.moka.ssp [openCPS master]
JRE System Library [JRE for JavaSE-1.8]
Plug-in Dependencies
src
META-INF
model
SystemStructureDescription.ecore
SystemStructureDescription.genmodel
build.properties
plugin.properties
plugin.xml
> org.eclipse.papyrus.moka.ssp.common [openCPS master]
org.eclipse.papyrus.moka.ssp.profile [openCPS master]
org.eclipse.papyrus.moka.ssp.references [openCPS master]
org.eclipse.papyrus.moka.ssp.setup [openCPS master]

Figure 6. Genmodel in the project

2. To generate model code, open a `.genmodel` file, right click on the root element, and click `Generate Model Code`. Wait until Eclipse builds the workspace and then do the same with `Generate Edit Code` and `Generate Editor Code`.

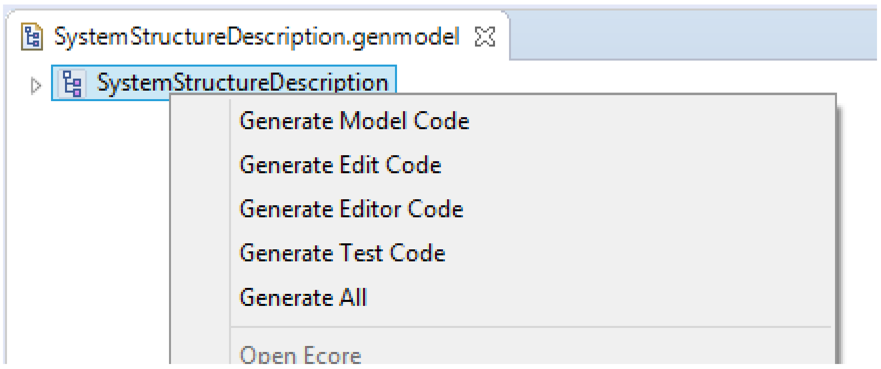


Figure 7. Generating model code

A more simple solution is to simply click `Generate all`, and remove the generated test-related projects.

3. After you've generated code from all genmodel files, select `Project > Clean` and `clean all` projects.
4. The `org.eclipse.papyrus.moka.ssp.uml2ssp` project has still errors, so you need to install the `org.eclipse.papyrus.moka.ssp.profile` project.
   - To do this, right-click on the project, `Export`….

  - From the *Plug-in development* folder choose `Deployable plug-ins and fragments`, *Next*.
  - In the *Destination* tab choose `Install into host Repository:`, and hit *Finish*.
  - It'll pop a security warning about unsigned content. Click *Install anyway*.
  - After it's complete, eclipse will ask for a restart, *Restart now*.
5. A clean might be needed (mostly for uml2ssp project).
6. That's it, you are all set now, the workspace should be errorless.

3.4 Setting up the project

1. First, you have to start a runtime Eclipse with the plugins installed.
2. You have to create a new `Papyrus Project`.
   a. On the first page, you can leave everything as it is, hit *Next*.
   b. Type the `Project Name`.
   c. Click *Finish*, nothing else needs to be set.
3. Open the `.di` file if it doesn't open automatically.
4. Open `Model Explorer` if it was not opened already.
   a. `Window` at top of Window → `Show View` → `Other…` (or use Alt-Shift-q q shortcut), search for `Model Explorer` and click open.
5. To import an SSP and start the transformations, right-click on the root of the model in the Model Explorer: `Import SSP as SysMML model`.
6. Choose an SSP file, and click *Open*.

Now you should see, that there are 3 items in the root package.

- Composite Structure Diagram: the diagram you can edit and work on.
- FMULibrary: the FMUs from the SSP are exported here.
- A (root) System with the `SsdTSystem` stereotype. This is the root element of the SSD file you transformed. When you edit the diagram, basically you edit this, and its children.

Your project folder should look similar to Figure 8.

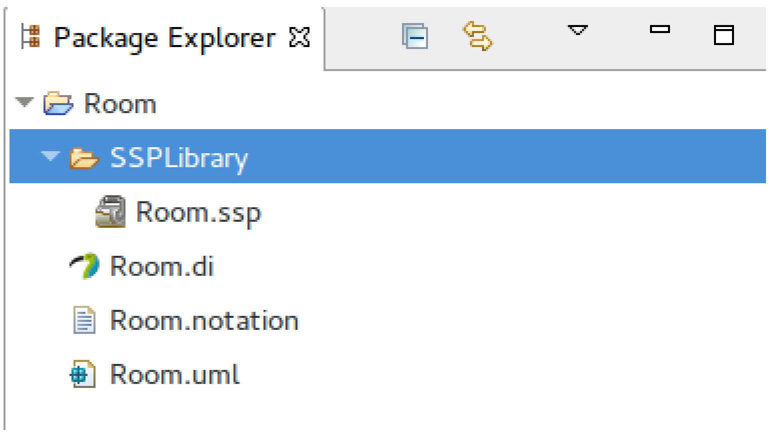
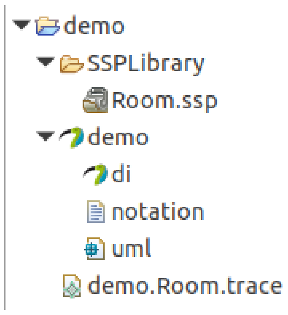


Figure 8. Project structure

In the `SSPLibrary` you can find `.ssp` file(s), witch contain(s) the used FMUs. They are saved here.

There is one other file, but just in memory, which is responsible for connecting between SSP and SysML. If you save the trace file will appear (Figure 9).



```
demo
  SSPLibrary
    Room.ssp
  demo
    di
    notation
    uml
  demo.Room.trace
```

Figure 9. New files

3.5 Using the application

To use the application, you'll need to open the diagram and you are all set to work. The event-driven transformation works in the background, as you work. When you save, it'll save the SSP too in the project folder, so you have it, when you need it.

3.6 Currently supported features

Modeling elements

Unfortunately our SSD to SysML synchronization feature does not support all language elements available in the current version of the SSP standards. The basic elements are supported, such as `System`, `Component`, `Connector` and `Connection`. Additionally, we also provide support for signals (`SignalDictionary, SignalDictionaryReference)`. A unique feature of the module is that it provides support for `buses` an extension developed within OpenCPS to support the TLM algorithm available in the OMSimulator. These buses are added as extensions to the current standard and are handled by the provided `Annotation` mechanism. The currently missing features are the transformations (`LinearTransformation`, `MappingTransformation`), parameterization (`ParameterBindings`), special types (`Enum` and `Unit`) and non-functional elements, e.g. graphical information and annotations are not yet supported.

End-user features

In general, our module provides graphical specification of complex simulation scenarios through SysML block definition diagrams that are automatically transformed into SPP standard compliant descriptions. Additionally, as the standard allows SSPs to contain other SSPs we also provide support for importing already available SSPs into the editor. Finally, as SPPs are encapsulating FMUs that are used as black box components we also provide support for exporting/importing FMU's into an SSP and also to give support for changing already installed FMU's within an SSP to a newer version, with diff/merge capabilities to see what has changed.

As changing an FMU within an already defined SSP is a complex task, its workflow is defined in Figure 10. First the new FMU (v2) is parsed into an EMF representation. Then we execute a diff/merge operation between the two FMU versions (parsed FMU v1 and v2), which based on the results of the selected operations to be executed during the merge operation will result in the v3 version of the FMU. Then the EMF representation of the FMU is transformed into the appropriate SysML format that is updated and can be used in the SSP SysML representation. Importing a new FMU to an SSP works similarly just the diff/merge parts are missing as there is no other version of that particular FMU available.

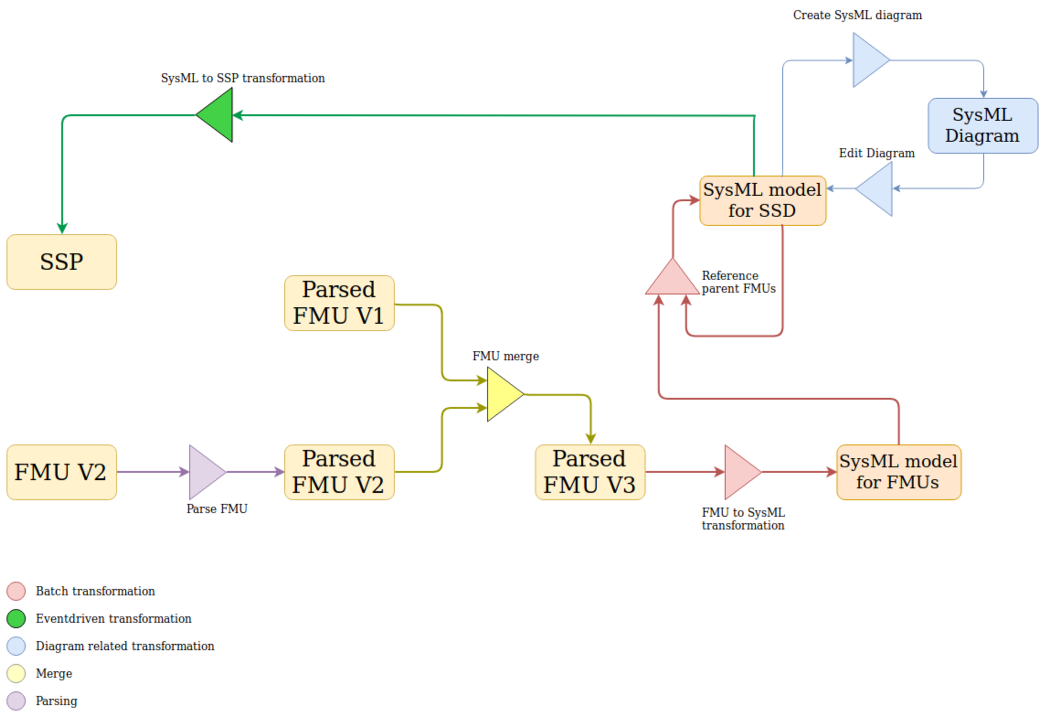


Figure 10: Workflow for replacing FMU witihn an SSP with a newer version.

3.7 Description of the used SSP

This SSP connects two FMUs, which are generated from OpenModelica. You can find the original modelica models of these FMUs, FMU files and the SSP file in the `/moka/ssp` folder of the repository.

Naturally, Controller.fmu is the FMU generated from the Controller.mo modelica model and pidSystem.fmu is the FMU generated from the pidSystem.mo modelica model.

3.7.1 In OpenModelica

Temp_controller.mo modelica model is where the other two have connected two each other, which connects you can see in Figure 10.

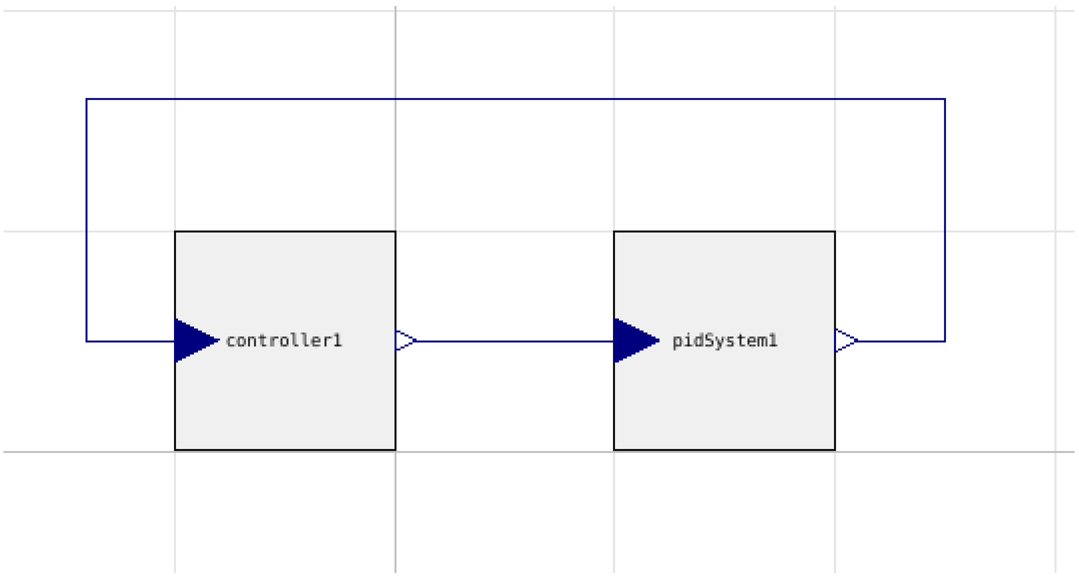


Figure 11. Modelica Temp_controller model

For ease of understanding this model is a simple example for demonstrating how to connect models, however, OpenModelica is able to handle significantly larger and more complex models (e.g., see WP6.3 Saab avionics demonstrator).

The input of `pidSystem1` is the filled triangle at left side. This is the expected temperature. The empty triangle on the right side of `pidSystem1` denotes the output of the model, which is the controller temperature. The controller has a state machine inside with two states, *low* and *high*.

If state machine is in state *high* and the controller temperature is low (lower than 20°), then the state machine will step to state *low* and the output will be 23°. If state machine is in state *low* and the controller temperature is high (higher than 23°), then the state machine will step to state *high* and output will be 20°.

3.7.2 In Eclipse

For the sake of simplicity we use the same model to show what can be done in Papyrus for connecting the two FMUs that are exported from the OpenModelica (controller and pidSystem) with SSP. We use the Papyrus editor to create the connections between the appropriate ports and the graphical representation looks like as depicted in Figure 12, while the abstract syntax representation looks like as depicted in Figure 13.

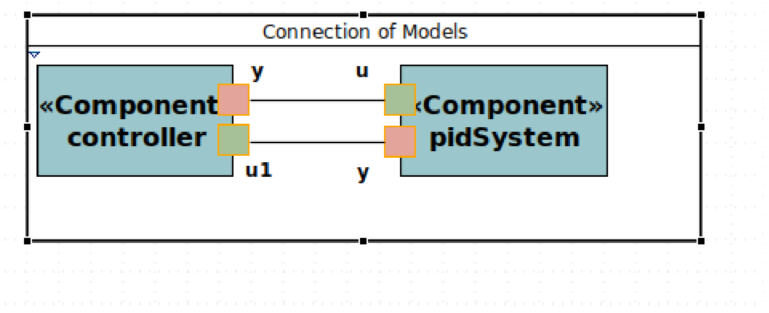
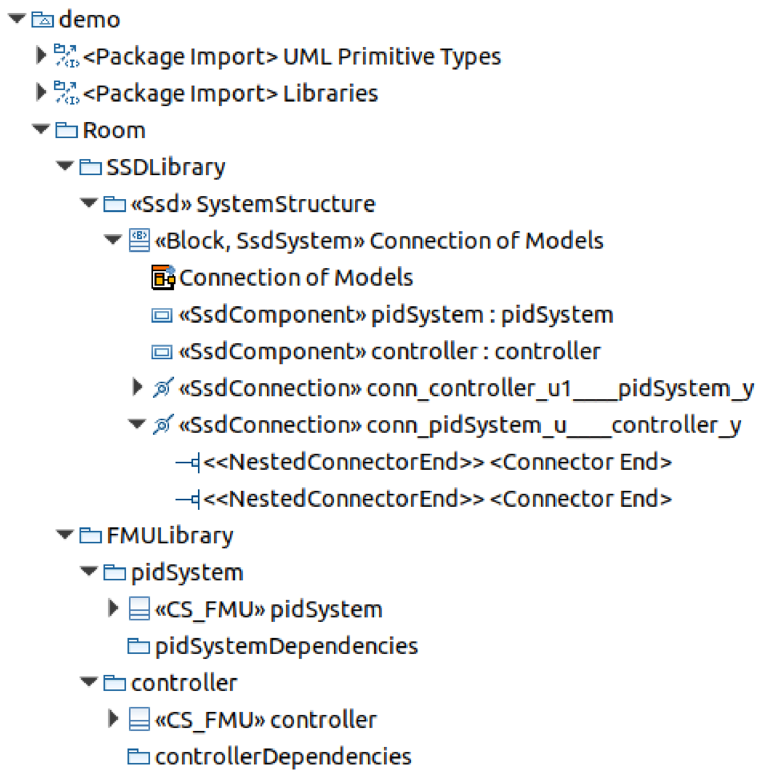


Figure 12. Room model in Eclipse

As it can be seen that abstract syntax representation show all the UML specific elements created for the model with the appropriate profiles (e.g., `<<SSD>>`, `<<CS_FMU>>`, `<<SsdConnection>>`) applied. The FMUs are stored under the FMU library element as each FMU can be instantiated multiple times in a single simulation scenario, while the SSDs are present under the SSD library.



```
▼ demo
  ▶ <Package Import> UML Primitive Types
  ▶ <Package Import> Libraries
  ▼ Room
    ▼ SSDLibrary
      ▼ «Ssd» SystemStructure
        ▼ «Block, SsdSystem» Connection of Models
            Connection of Models
            «SsdComponent» pidSystem : pidSystem
            «SsdComponent» controller : controller
          ▶ «SsdConnection» conn_controller_u1___pidSystem_y
          ▼ «SsdConnection» conn_pidSystem_u___controller_y
              <<NestedConnectorEnd>> <Connector End>
              <<NestedConnectorEnd>> <Connector End>
    ▼ FMULibrary
      ▼ pidSystem
        ▶ «CS_FMU» pidSystem
          pidSystemDependencies
      ▼ controller
        ▶ «CS_FMU» controller
          controllerDependencies
```

Figure 13. Project in Model Exploler

Finally, the corresponding SSP configuration XML (called SSD) look like as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<ssd:SystemStructureDescription          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ssc="http://www.pmsf.net/xsd/SystemStructureCommonDraft"
xmlns:ssd="http://www.pmsf.net/xsd/SystemStructureDescriptionDraft"
xsi:schemaLocation="http://www.pmsf.net/xsd/SystemStructureDescriptionDraft
http://www.pmsf.net/xsd/SSP/Draft20170606/SystemStructureDescription.xsd"          name="Room"
version="Draft20170606">
  <ssd:System description="void description" name="Connection of Models">
    <ssd:Elements>
      <ssd:Component  name="controller"  source="resources/controller.fmu"  type="application/x-
fmu-sharedlibrary">
        <ssd:Connectors>
          <ssd:Connector kind="output" name="y">
            <ssc:Real unit=""/>
          </ssd:Connector>
          <ssd:Connector name="u1">
            <ssc:Real/>
          </ssd:Connector>
        </ssd:Connectors>
      </ssd:Component>
      <ssd:Component name="pidSystem" source="resources/pidSystem.fmu" type="application/x-fmu-
sharedlibrary">
        <ssd:Connectors>
          <ssd:Connector kind="output" name="y">
            <ssc:Real unit=""/>
          </ssd:Connector>
          <ssd:Connector name="u">
            <ssc:Real/>
          </ssd:Connector>
        </ssd:Connectors>
      </ssd:Component>
    </ssd:Elements>
    <ssd:Connections>
      <ssd:Connection     endConnector="y"     endElement="controller"     startConnector="u"
startElement="pidSystem"/>
      <ssd:Connection     endConnector="y"     endElement="pidSystem"     startConnector="u1"
startElement="controller"/>
    </ssd:Connections>
  </ssd:System>
</ssd:SystemStructureDescription>
```

The SSD XML contains all necessary information and links/connections defining the simulation scenario and also provides the relative links to the used FMUs in the `resources` folder within the SSP.

4 SUMMARY

The current document presents a demo software developed within D2.3 to evaluate the possible usage of SSP for defining the input configuration of the master simulation tool. The approach is promising and as OMSimulator also support SSD as its input format the current extension to Papyrus can be used as an authoring tool for defining complex simulation scenarios. The change from UML to SysML was driven by the demonstrators developed in WP6 as both the Saab and the Sherpa demonstrator uses SysML as its architecture description language, thus our final solution provides a complete round-trip engineering approach for both architecture and simulation design as demonstrated in D6.3.