SUBMISSION LWC2012 SIOUX

Author	Paul Zenden, et. al.
Date	16-03-2012
Version	1.0
Status	Qualified
Ref. no.	-
Ref. name	-
Project	LWC2012

TABLE OF CONTENTS

CHANGE HISTORY

Version	Date	Author	Description
1.0	16-3-2012	Paul Zenden	Initial version

1 INTRODUCTION

This document describes the solution as developed by Sioux for the LWC 2012 Piping & Instrumentation case. All mentioned files can be downloaded from our ftp server: ftp.sioux.eu:5521.

Login information can be requested via paul.zenden@sioux.eu. The credentials will be sent to you as quick as possible.

The ftp site has the following content:

LWC2012 Piping & Instrumentation - v2.pdf	Description of the LWC2012 Piping & Instrumentation case.
LWC2012 Submission UML and XPand.pdf	The document you are now reading.
/PPL2010	Contains presentation and example files of a tutorial given on the Practical Product Lines 2010 ().
/Reference Implementation	Contains presentation and supporting files provided as a reference implementation as part of the LWC2012 case.
/Sioux Submission	Contains the files of the solution provided by Sioux, based on UML Profile and Xpand.
/Sioux Submission/eu.sioux.pils.gen.zip	Eclipse project that contains the generator.
/Sioux Submission/ModelFiles.zip	Enterprise Architect Meta-model and model files (also included in eclipse project).
/Sioux Submission/PLC.zip	The TwinCAT project.

If you encounter any problems with replaying our solution please contact us at: paul.zenden@sioux.eu.

2 TOOL-CHAIN

2.1 Overview

This submission for the LWC 2012 challenge provides a solution based on UML, UML Profile and Eclipse/XPand. Next figure gives a global overview of the tool-chain we use.

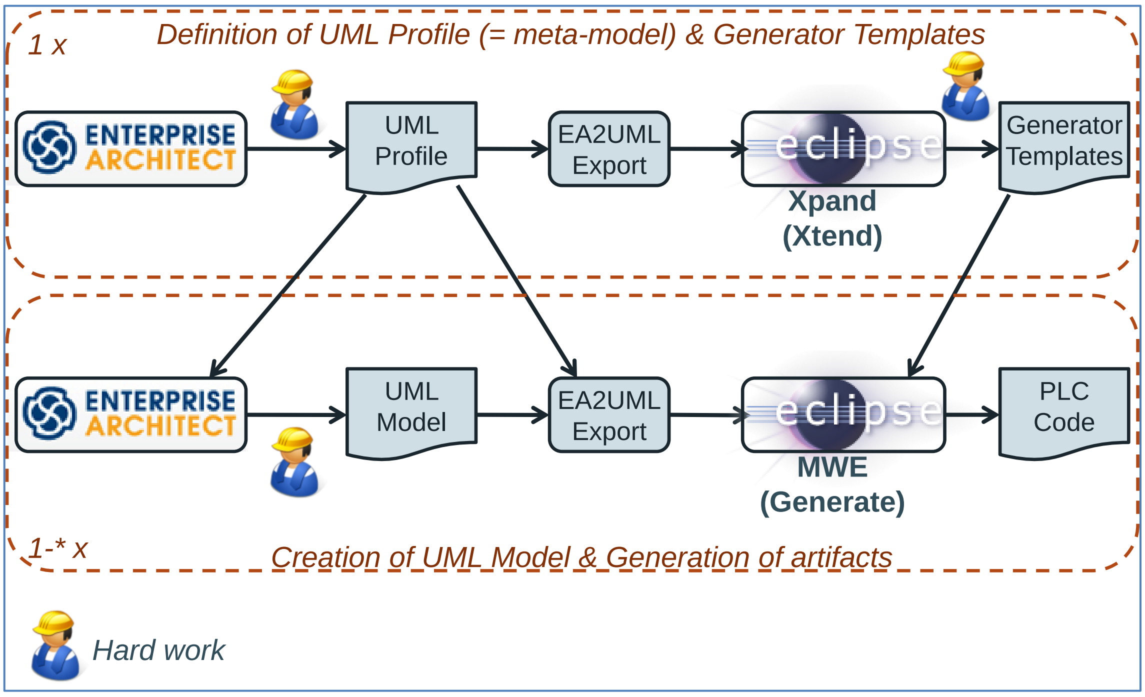


Figure 2-1: Overview Tool-chain

This solution starts with developing the meta-model as a UML Profile. For this we use Enterprise Architect (EA). EA allows besides definition of concepts (as UML Stereotypes) also changing the shape of the concepts. We use this feature to make the concepts look more like the shapes you can find in the Piping & Instrumentation domain.

The UML Profile is imported in Eclipse/XPand. The generator templates are developed in XPand. During development of the generator templates, the concepts defined by the UML Profile are ‘known’ inside Eclipse/XPand editor and allow for dynamic checking while entering the generator code.

Based on the defined UML Profile, a representation of the home heating case is developed (called the model). This is only a structural description of the case. For defining the behaviour we use UML state machines.

Next, the UML Profile (as the generation environment also needs to know the concepts that are being used), the UML structural model and the UML state machines, are imported in the Eclipse/Xpand environment.

The generator is based on XPand templates and MWE workflow engine. The generator generates text files which are in fact just snippets of code which need to be paste in the right places in the Twincat environment. This is a little bit cumbersome, due to the nature of Twincat. Probably in the next release of Twincat (v3.x) this can be handled better. In a real project environment we create merge scripts that paste the pieces code in the right places automatically. We left that out for this submission.

2.2 Tutorial setting up tool-chain

For the Practical Product Lines 2010 (http://www.practicalproductlines.org/ppl2010/sessioninfo.php?session=5), we developed a tutorial that explains the way of working with EA, UML, UML Profiles, XPand in high detail. Although, in the meantime, newer versions of used products have become available, the tutorial is still a good starting point to get a working environment. The information is also made available as part of the LWC2012 submission. Look for the following file:

- PPL2010/Tutorial Enterprise Architect + XPand.pdf

Other sub-direcotories of the PPL2010 folder contain files mentioned in the presentation.

Note: Updated software versions and download locations of the used tooling are given in the subsequent chapters.

2.3 Enterprise Architect

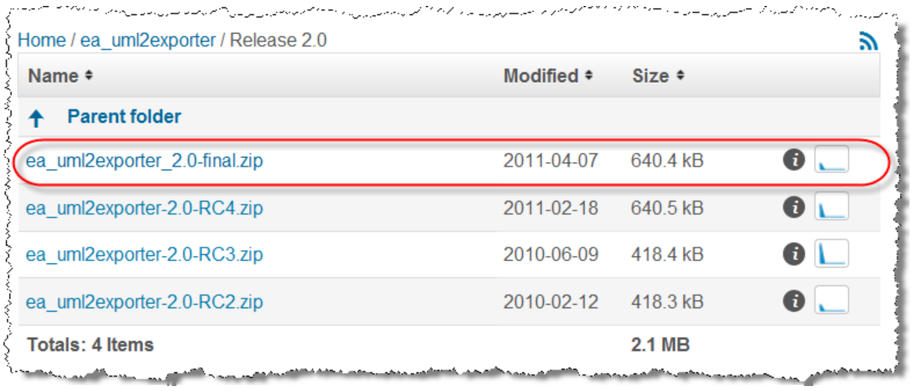
If you don't have Enterprise Architect installed, you can use the trial version. The trial version contains all features, and allows you to use it for 30 days. When model repositories are opened with the trial version, you will get a start-up screen, even if the repository file is opened in the background from within Eclipse. You still need to acknowledge this screen. If it looks like the generation is stalled, please check your windows to see whether this start-up screen is not hiding behind one of your other screens.

The trial can be downloaded from: http://www.sparxsystems.com/products/ea/trial.html .
After installing the EA trial, you find the files needed for the Java API connection in the 'Java API' folder in the directory where the program has been installed (e.g. C:\Program Files (x86)\Sparx Systems\EA Trial\Java API).

2.4 EA to Eclipse exporter

For exporting EA models into Eclipse, we use the open source library: ea_uml2exporter (http://components4oaw.sourceforge.net/) developed by Ulrich Brawand (see also http://uml2ea.blogspot.com/).

We use 2.0-Final.zip (location: http://sourceforge.net/projects/components4oaw/files/ea_uml2exporter/Release%202.0/)
See figure 2-2.



Home / ea_uml2exporter / Release 2.0

Name ⬍	Modified ⬍	Size ⬍
Parent folder		
ea_uml2exporter_2.0-final.zip	2011-04-07	640.4 kB
ea_uml2exporter-2.0-RC4.zip	2011-02-18	640.5 kB
ea_uml2exporter-2.0-RC3.zip	2010-06-09	418.4 kB
ea_uml2exporter-2.0-RC2.zip	2010-02-12	418.3 kB
Totals: 4 Items		2.1 MB

Figure 2-2: Download ea_uml2exporter

You find some more valuable documentation about using the exporter in combination with EA on: http://sourceforge.net/projects/components4oaw/files/ea_uml2exporter%20documentation/ and http://sourceforge.net/projects/components4oaw/files/ea_uml2exporter%20tutorials/.

2.5 Eclipse/XPand

Requires (of course) that Java is installed. Developing and testing has been carried out with java jdk6 update 22 32-bits for windows (http://www.oracle.com/technetwork/java/javase/downloads/index.html). Probably any jdk6 release will work. We didn't test jdk7 release.

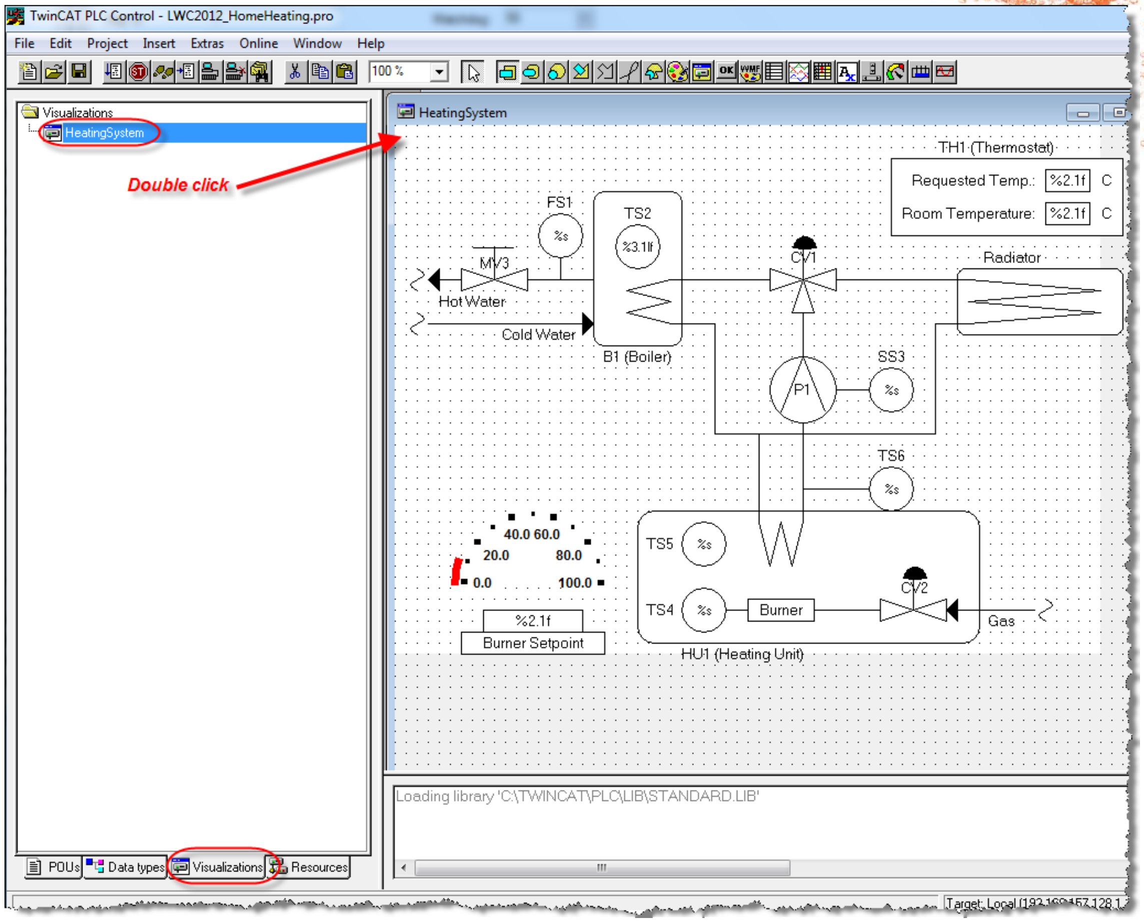
The quickest way to get started with Eclipse/Xpand is to use the eclipse archive as kindly provided by Itemis: http://download.itemis.com/distros/eclipse-SDK-3.6.2-xtext-1.0.2-win32.zip

2.6 UML & UML Profiles

The Unified Modeling Language^{TM} (UML) is a specification from the Object Management Group (OMG). More information can be found on: http://www.uml.org/.
UML Profile is an important mechanism to extend UML with specialized concepts.

2.7 TwinCAT

Look at the reference implementation to see a detail description of how to set up the TwinCAT environment and how to run the PLC code. Don't forget to make the visualisation screen visible. See figure 2-3.

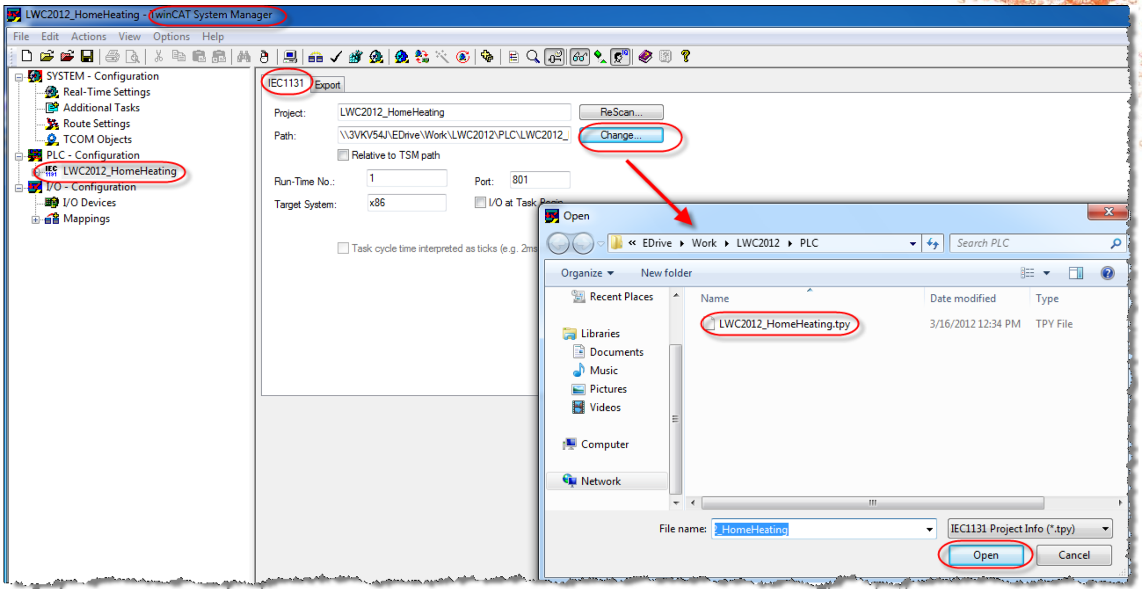


Loading library 'C:\TWINCAT\PLC\LIB\STANDARD.LIB'

Figure 2-3: Make UI visible within TwinCAT environment

Note:

To get the PLC project working in your environment, you probably need to adjust the location of where the TwinCAT System Manager can find the PLC project.
To do this (see figure 2-4):
Double click on the LWC2012_HomeHeating.tsm file. This will start up the TwinCAT System Manager. In the System Manager, select the LWC2012_HomeHeating entry.
Click on the Change… button and select the LWC2012_HomeHeating.tpy file and click Open.
System Manager will automatically rescan the project. This should be sufficient.



Project: LWC2012_HomeHeating
Path: \\3VKV54J\EDrive\Work\LWC2012\PLC\LWC2012_
Relative to TSM path
Run-Time No.: 1
Port: 801
Target System: x86
I/O at Task Begin
Task cycle time interpreted as ticks (e.g. 2ms

« EDrive ▸ Work ▸ LWC2012 ▸ PLC
Search PLC

Name	Date modified	Type
LWC2012_HomeHeating.tpy	3/16/2012 12:34 PM	TPY File

File name: 2_HomeHeating
IEC1131 Project Info (*.tpy)

Figure 2-4 Re-connect TwinCAT System Manager

3 LWC2012 CHALLENGE

3.1 Overview

Our solution comprises several parts:

- A **meta-model** represented by a UML Profile, including specific shapes for concepts to define the structural elements.
- A **model** representing a home heating case as presented in the LWC2012 challenge. The model contains both structural definition (based on the above mentioned profile) as well as the behaviour specification (represented by state machines).
- A set of **generator templates** (in Xpand) that produce pieces of PLC code that represent the structure and behaviour as defined in the model and allow executing the control function of the PLC
- A set of **basic building blocks and viewing screen** (PLC code) provided as reference implementation. These building blocks are the basic framework upon the generated code executed and is developed manually.

These parts are described in the following chapters.

We initially planned to also work on a set of generator templates (in Xpand) that would produce pieces of PLC code that implement a simple simulation of the represented model. However, we didn’t found the time to do this.

3.2 Meta Model

3.2.1 Concepts

The set of concepts represent our DSL. We gave it the name: Piping & Instrumentation Language System (PILS[1]).

We implemented a limited set of concepts from the Piping & Instrumentation domain. These concepts are:

Concept	Description
PILSModel	The root element of any model.
Pipe	Represents the connection between any of the other concepts. A Pipe has two ends which must both be connected. Direction between the two ends is considered not to be relevant.
MeasuringInstrument	Represents a sensor. Measures some physical characteristic about the substance flowing through the element to which the measuring instrument is connected.
CentralHeatingUnit	Represents the component that heats up the water by using a burner. It embeds sensors that detect whether the burner is on or off, and what the temperature is the burner is generating.
HeatExchanger	Represents the concept where heat is exchanged from the closed system of pipes with heated water to something outside this system. Possible kind of HeatExchangers are: - Radiator (exchanging heat to the room)

[1] A little joke: Pils is the dutch word for beer

Concept	Description
	- Boiler (exchanging heat the hot water boiler)
Source (System end)	Represents a connection with the world (outside the model) where gasses or fluids enter the model.
Sink (System end)	Represents a connection with the world (outside the model) where gasses or fluids leave the model.
Joint	Represents a point where multiple pipes are connected together.
Regular Pump	Represents a fluid pump.
TwoWayValve	Represents a valve with 2 possible states: open or closed
ThreeWayValve	Represents a valve with 3 possible states: open to left connection, open to right connection or open to left and right (middle position)
Thermostat	Represents the room thermostat. Used for reference purpose only so that in state machines actual temperature values assumed to be measured by the room thermostat can be referenced.
Controller	Represents independent part of the PLC controller. Used as the owning element of the state machines that describes the behaviour. By using more Controllers, independent state machines can be modelled.

The meta-model consists of several parts:

- A **UML Profile**: Define the stereotypes that are available to design your model.
- A **framework of classes** that implement meta model concepts.
- **Constraints checks** that verify a model against the implicit and explicit constraints defined by the meta-model.

The fact that the meta-model is described in multiple parts means that there is not one single part that describes the meta-model completely. This in itself is an issue and must be remembered when dealing with UML Profiles as meta-modelling tool.

Next chapters go in more detail in each of these parts.

3.2.2 UML Profile

3.2.2.1 General introduction

Definition of a UML Profile in EA is done by creating a new package of stereotype <<profile>>. using the Profile toolbox. See figure 3-1.

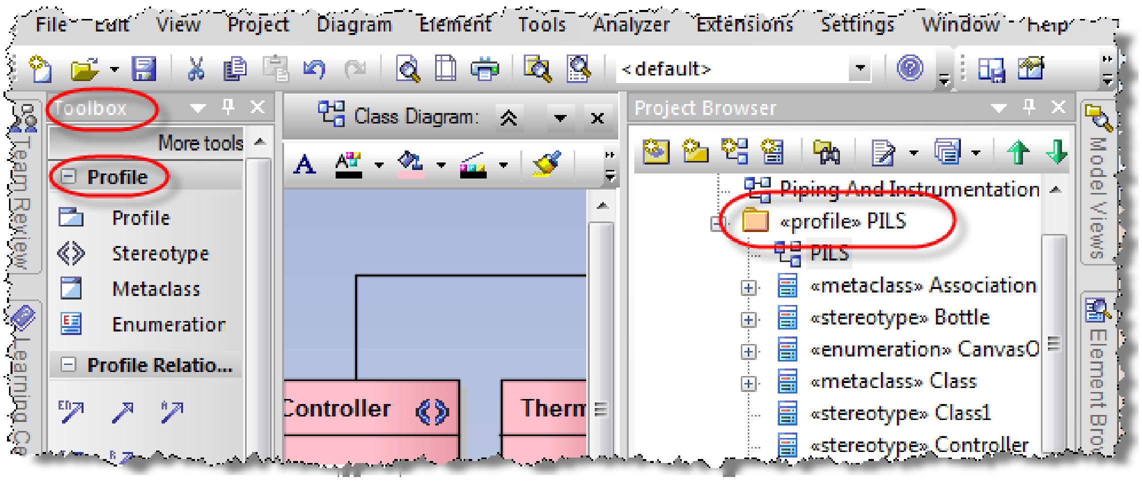


Figure 3-1: Profile tool box

Next, each concept is created as a <<stereotype>>, which need to extend a UML metaclass concept. For example, figure 3-2 shows the *Controller* stereotype extending from the *Class* metaclass. Meta-classes can be added by dragging the Metaclass entry from the toolbox on the diagram.

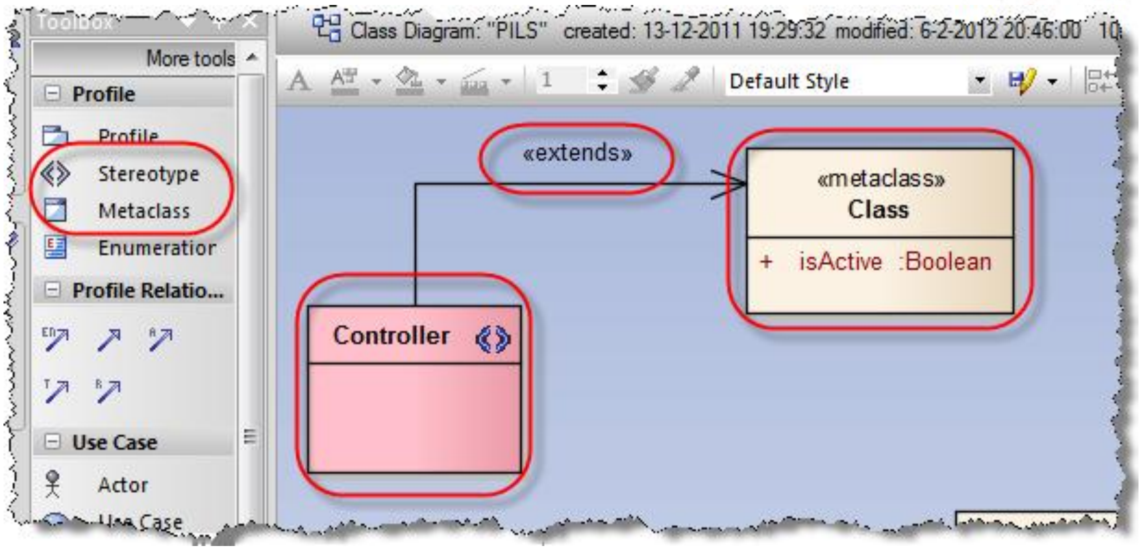


Figure 3-2: Controller stereotype

For some concepts it is required to be able to specify additional information when using the concepts in the model. For example, for a Pipe you would like to define the length and the diameter. This is achieved by adding attributes to the stereotype, which become so-called Tagged Values in the model. See figure 3-3.

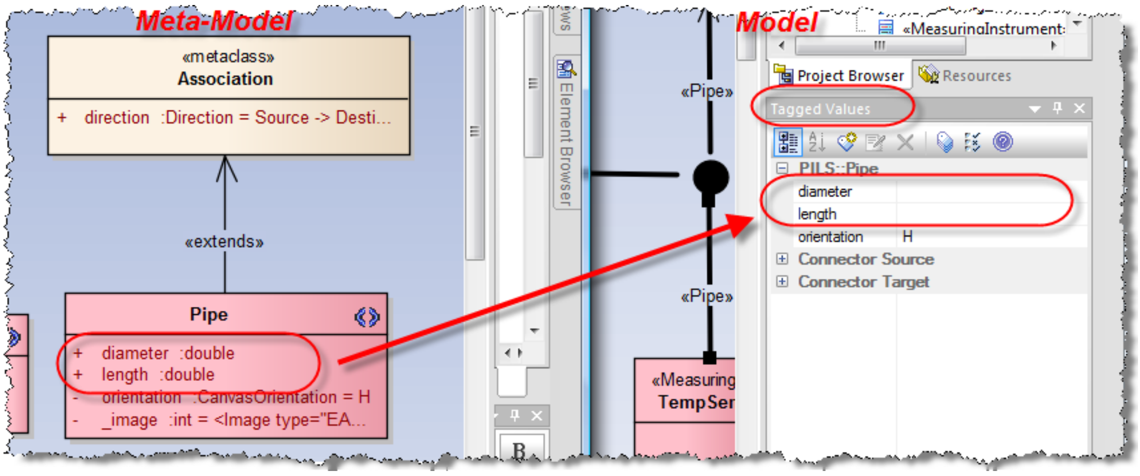
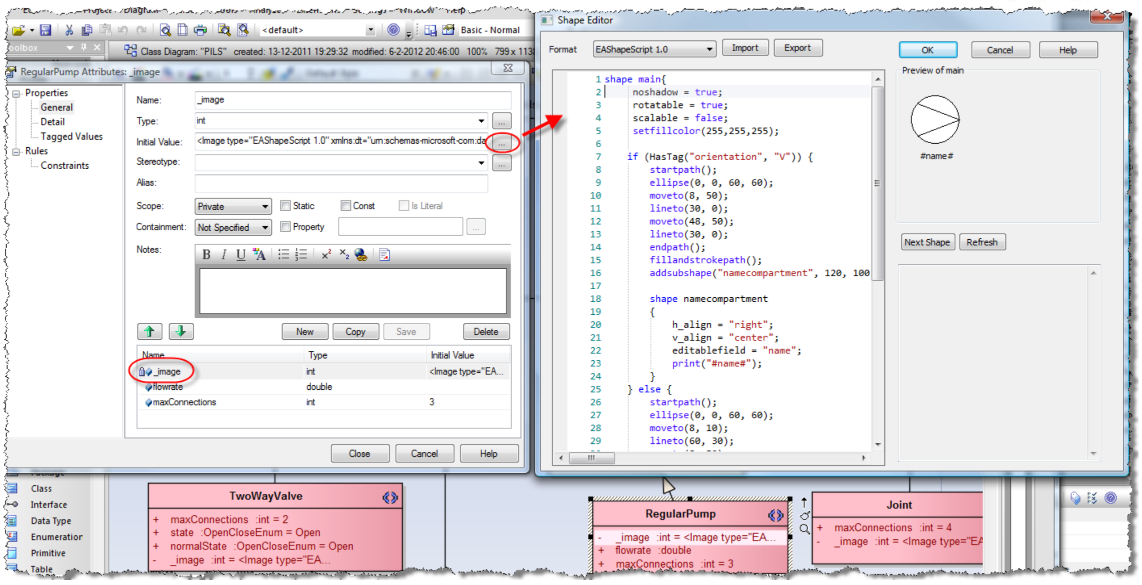


Figure 3-3: Adding tagged values

Figure 3-3 also shows that Pipe extends the meta-class *Association*. So, everywhere where in a UML class diagram an association can be used, it is also possible to use the Pipe stereotype.

A special attribute (_image) is used to define a new shape for the stereotype. If there is no _image attribute, the shape related to the extended meta-class will be used. The Help information of EA provides detailed information about how to create shapes and the available functions. Shapes are defined by a kind of script written in a small language defined by EA. (See also: http://www.sparxsystems.com/enterprise_architect_user_guide/8.0/modeling_languages/usingtheshapescript.html).

Figure 3-4 represents the shape for a pump.



RegularPump Attributes: _image

Name:	_image
Type:	int
Initial Value:	<Image type="EAShapeScript 1.0" xmlns:dt="urn:schemas-microsoft-com:da
Stereotype:	
Alias:	
Scope:	Private
Containment:	Not Specified

Name	Type	Initial Value
_image	int	<Image type="EA...
flowrate	double	
maxConnections	int	3

Shape Editor — Format: EAShapeScript 1.0

```
shape main{
     noshadow = true;
     rotatable = true;
     scalable = false;
     setfillcolor(255,255,255);

    if (HasTag("orientation", "V")) {
        startpath();
        ellipse(0, 0, 60, 60);
        moveto(8, 50);
        lineto(30, 0);
        moveto(48, 50);
        lineto(30, 0);
        endpath();
        fillandstrokepath();
        addsubshape("namecompartment", 120, 100

        shape namecompartment
        {
            h_align = "right";
            v_align = "center";
            editablefield = "name";
            print("#name#");
        }
    } else {
        startpath();
        ellipse(0, 0, 60, 60);
        moveto(8, 10);
        lineto(60, 30);
```

TwoWayValve
+ maxConnections :int = 2
+ state :OpenCloseEnum = Open
+ normalState :OpenCloseEnum = Open
- _image :int = <Image type="EA...

RegularPump
- _image :int = <Image type="EA...
+ flowrate :double
+ maxConnections :int = 3

Joint
+ maxConnections :int = 4
- _image :int = <Image type="EA

Figure 3-4: Shape script of Pump stereotype

If you look closely to the script, you see that it is possible to reference tagged values and use the actual content to control the script. We use the tagged value *orientation* to allow changing the orientation in the model, to get some nicer looking models.

The PPL2010 tutorial from sheet 29 on, gives more details on how to create and use UML Profiles in EA.

When you're done with defining the profile, it must be exported to an xml file (figure 3-5). This file can then be imported in a model file which makes the stereotypes defined in the profile available in the tool box (figure 3-6).

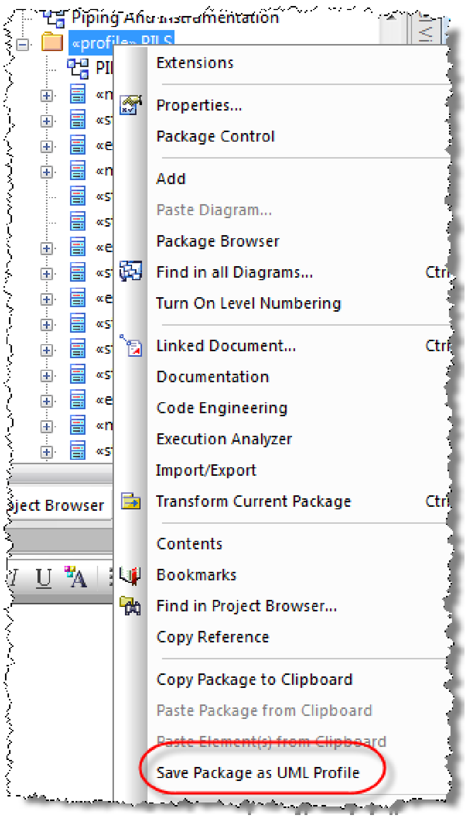


Figure 3-5: Save profile in XML file (from Profile repository)

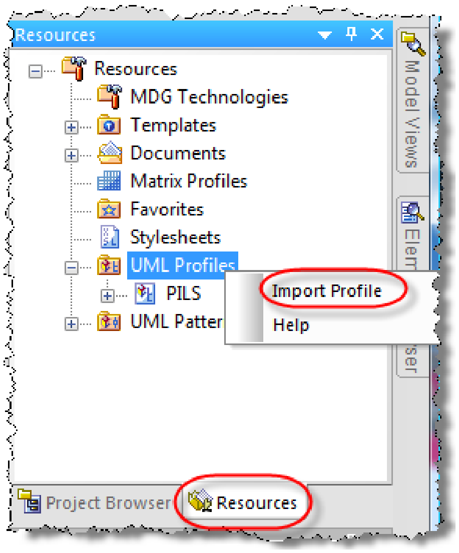


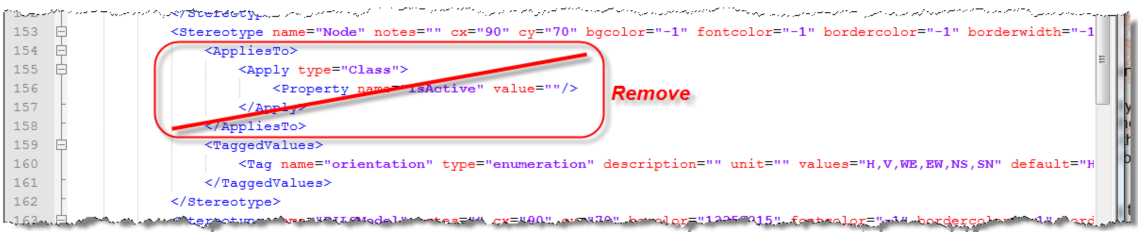
Figure 3-6: Import profile(into model repository)

Note:
There is one noteworthy issue in EA. In our meta model we introduced some layering through abstract stereotypes. The abstract stereotypes allow us to group common attributes. These abstract stereotypes need to be extended from a meta class otherwise the generation

environment can’t handle them correctly (We didn’t investigate what component of the tool chain exactly causes the problem.)
However, EA will only show in your model tool box *non-abstract stereotypes which directly extend from the meta class*. Thus to show our ‘leaf stereotypes’ we also need to extend the leaf elements from a meta class. Stereotypes that have more than one *extend* relationship to the same meta class will show up twice (or more) in a little drop down box in your model tool box. This is somehow weird. And we have also experienced incidental crashes of EA.

The solution is to remove the extend relationship of the stereotype Node before importing the profile into the model. This can be done in 2 ways:

- First: Temporarily remove the extend relationship in the profile between Node and Class before saving the profile to an XML file. And add the extend relationship again before closing the EA file.
- Second: Manually change the export profile XML file. Look for the entry of the stereotype Node and remove the tag AppliesTo. See figure 3-7.



```
<Stereotype name="Node" notes="" cx="90" cy="70" bgcolor="-1" fontcolor="-1" bordercolor="-1" borderwidth="-1
    <AppliesTo>
        <Apply type="Class">
            <Property name="isActive" value=""/>
        </Apply>
    </AppliesTo>
    <TaggedValues>
        <Tag name="orientation" type="enumeration" description="" unit="" values="H,V,WE,EW,NS,SN" default="H
    </TaggedValues>
</Stereotype>
```

Figure 3-7: Modify profile xml file to prevent stereotype to appear in dropdown box.

As said before: For the generation the double extend relationship must be present!

Figure 3-8 shows the RegularPump stereotype being derived from a Pump stereotype, which again is derived from the Node stereotype, but also having its own extend relationship to meta class Class. Note that to keep the diagram readable, extend relationships from the leaf elements are hidden on the diagram.

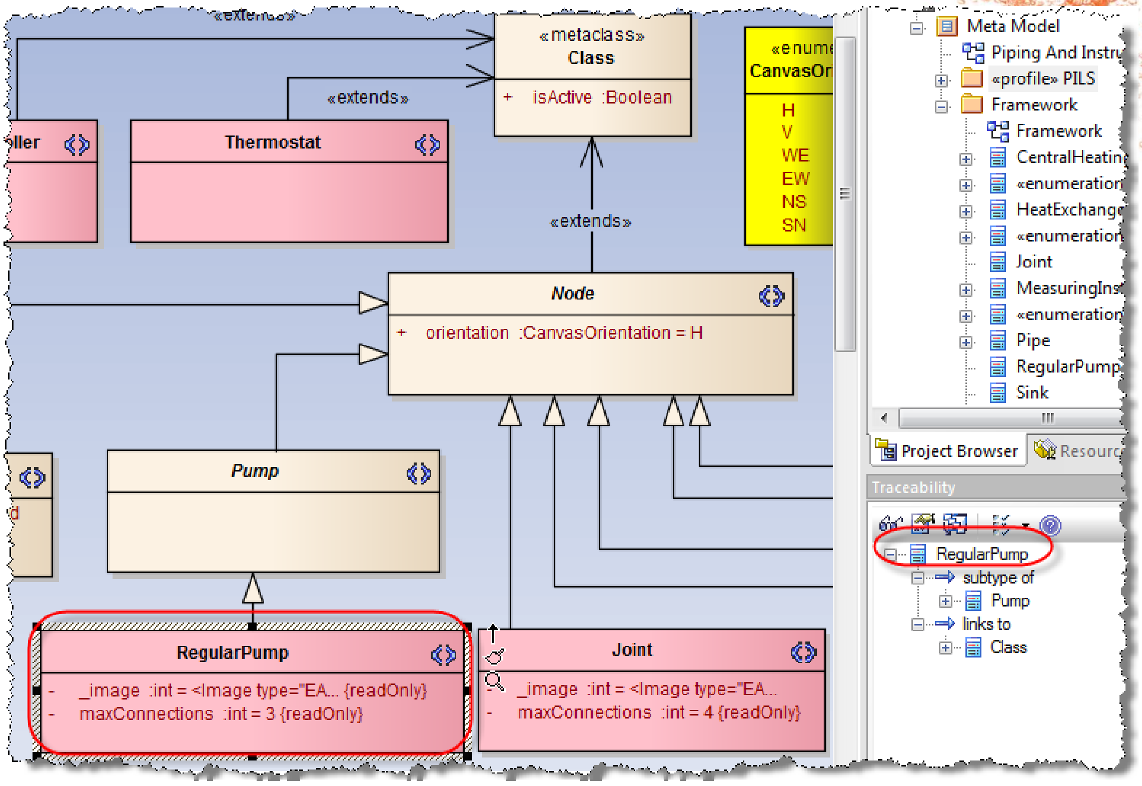


Figure 3-8: Leaf classes have 2 extend relationships (of which only one is drawn)

3.2.2.2 Profile: PILS

The UML Profile is defined in the file PILS.eap in the package Meta Model/PILS.
We have defined several attributes (tagged values) for some of the stereotypes. You can also see examples of attributes which are of enumeration type and have a fixed list of allowed values. Figures 3-9, 3-10 represent the complete profile definition.

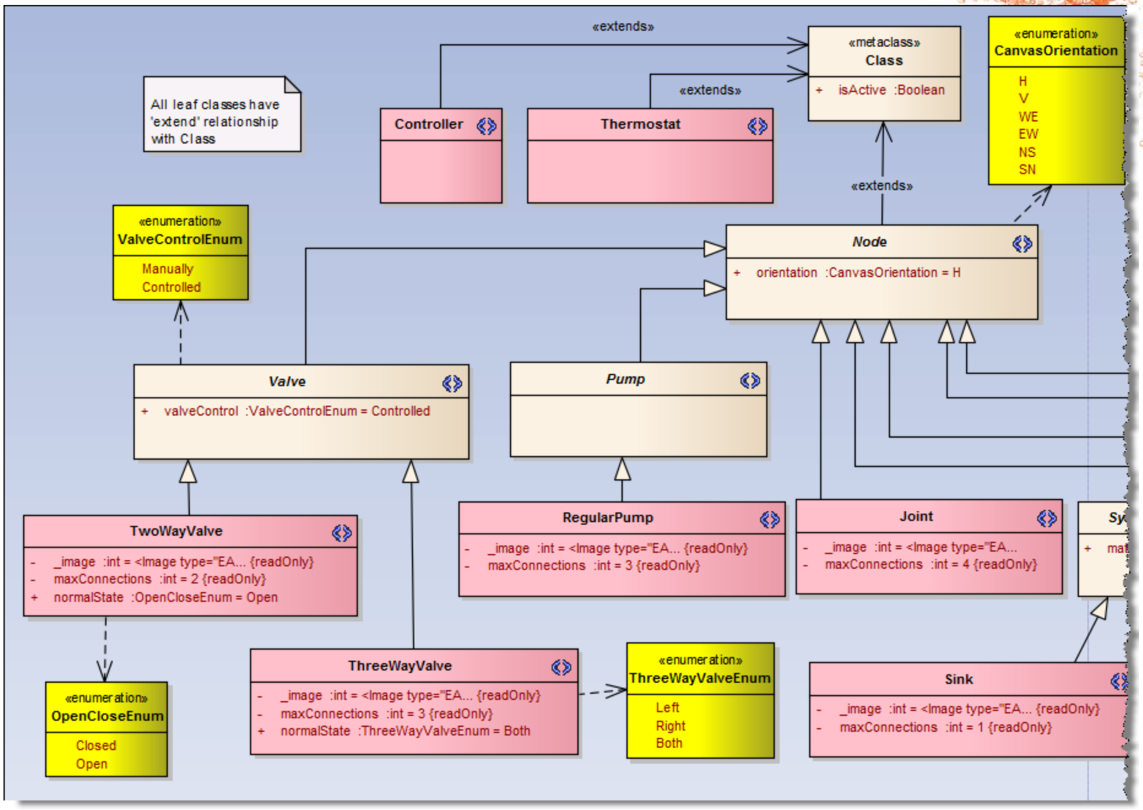

«extends»
«metaclass» Class
+ isActive :Boolean
«enumeration» CanvasOrientation
H
V
WE
EW
NS
SN
All leaf classes have 'extend' relationship with Class
Controller
Thermostat
«extends»
«extends»
«enumeration» ValveControlEnum
Manually
Controlled
Node
+ orientation :CanvasOrientation = H
Valve
+ valveControl :ValveControlEnum = Controlled
Pump
TwoWayValve
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 2 {readOnly}
+ normalState :OpenCloseEnum = Open
RegularPump
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 3 {readOnly}
Joint
- _image :int = <Image type="EA...
- maxConnections :int = 4 {readOnly}
«enumeration» OpenCloseEnum
Closed
Open
ThreeWayValve
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 3 {readOnly}
+ normalState :ThreeWayValveEnum = Both
«enumeration» ThreeWayValveEnum
Left
Right
Both
Sink
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 1 {readOnly}

Figure 3-9: PILS Profile (part 1)

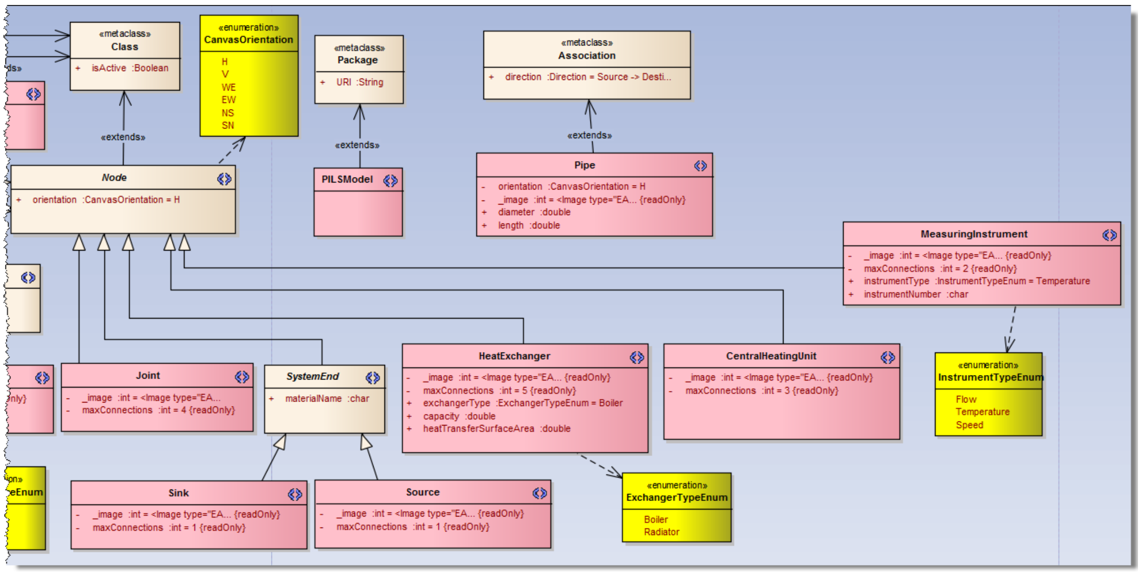

«metaclass» Class
+ isActive :Boolean
«enumeration» CanvasOrientation
H
V
WE
EW
NS
SN
«metaclass» Package
+ URI :String
«metaclass» Association
+ direction :Direction = Source -> Desti...
«extends»
«extends»
«extends»
Node
+ orientation :CanvasOrientation = H
PILSModel
Pipe
- orientation :CanvasOrientation = H
- _image :int = <Image type="EA... {readOnly}
+ diameter :double
+ length :double
MeasuringInstrument
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 2 {readOnly}
+ instrumentType :InstrumentTypeEnum = Temperature
+ instrumentNumber :char
Joint
- _image :int = <Image type="EA...
- maxConnections :int = 4 {readOnly}
SystemEnd
+ materialName :char
HeatExchanger
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 5 {readOnly}
+ exchangerType :ExchangerTypeEnum = Boiler
+ capacity :double
+ heatTransferSurfaceArea :double
CentralHeatingUnit
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 3 {readOnly}
«enumeration» InstrumentTypeEnum
Flow
Temperature
Speed
Sink
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 1 {readOnly}
Source
- _image :int = <Image type="EA... {readOnly}
- maxConnections :int = 1 {readOnly}
«enumeration» ExchangerTypeEnum
Boiler
Radiator

Figure 3-10: PILS Profile (part 2)

3.2.3 Framework

For the framework you could consider developing a platform independent version and a platform specific version. This is especially interesting when the generation needs to target multiple platforms. For the LWCW2012 challenge we only generate to one target. To save time we only developed a platform specific (= PLC) design of a framework.

This framework contains the building blocks that can be instantiated and instances can be connected with each other to reflect the setup as defined by the model. The generator will generate these instantiations and wiring, for as far as needed.

This framework exposes characteristics of the meta model concepts that were not visible in the UML profile. Examples of these characteristics are:

- the available operations, and,
- the properties that change at run-time and are publicly available.

These characteristics can be used in the definition of the state machines!

See figure 3-11.

Note that not all meta model concepts are available in the framework. Concepts which have no influence and/or are not needed for definition of and subsequent implementation of the control algorithms are not made available. For example, the concept Pipe is not known to the state machines or controller software in the PLC. The fact that a pump and a valve are connected via a pipe is a physical fact but is of no interest to the controller.

However, the simulation framework would need to know about pipes and connections to realize realistic behaviour!

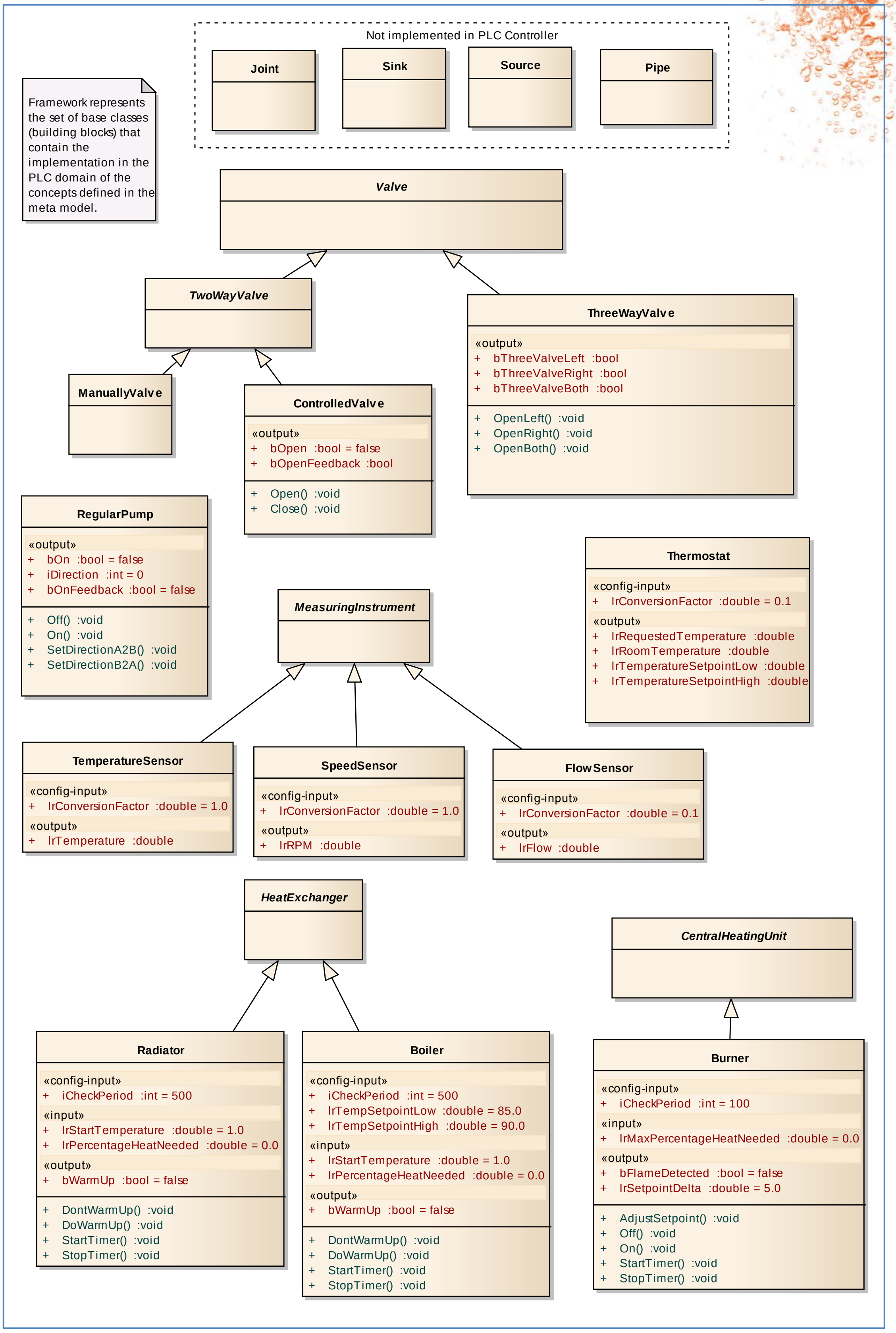


Figure 3-11: Framework (part of meta-model) design

3.2.4 Constraints Checks

These still need to be defined.
Example of constraints to check:

- Unique name of all model elements
- Adherence to maxConnections tagged value.

Constraint checks are implemented as part of the generator setup in so-called check files (extension .chk).

3.3 Model

3.3.1 General

The model is described in the file: CentralHeating.eap, package HomeHeating.
It consists of 2 parts:

- The structural model, defined using the PILS profile,
- The behavioural model, defined using UML State Machines.

Remember that you need to import the PILS profile before you can use it.

3.3.2 Structure

Figure 3-12 shows the model.

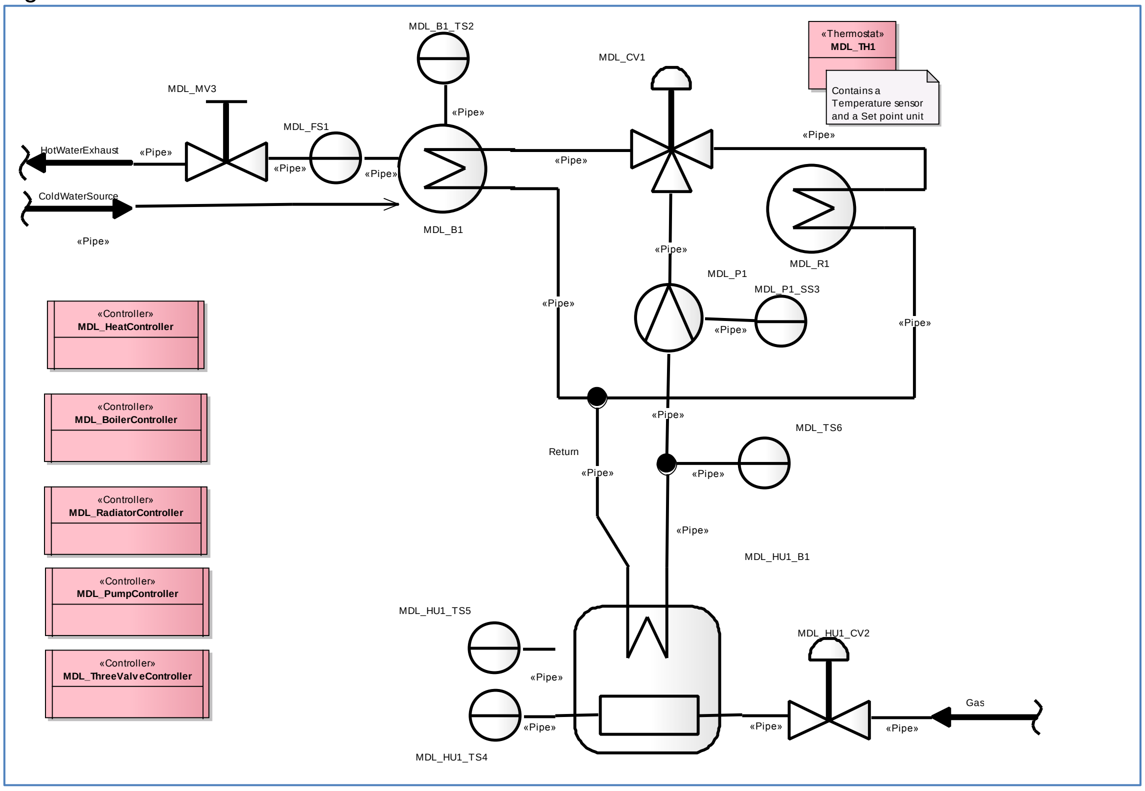
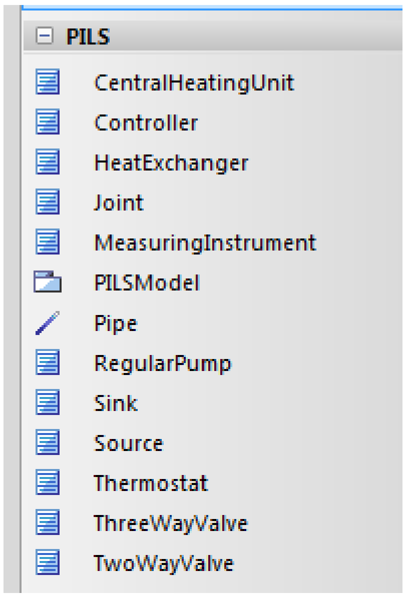


Figure 3-12: Model of Central home heating system

The different elements of the model are instantiated by selecting the respective stereotype in the toolbox and pasting it on the canvas. Figure 3-13 shows the toolbox.



PILS

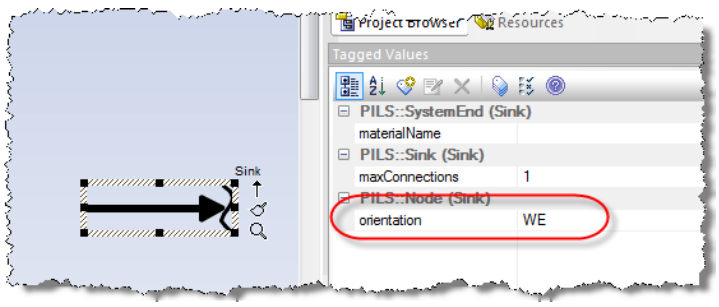
- CentralHeatingUnit
- Controller
- HeatExchanger
- Joint
- MeasuringInstrument
- PILSModel
- Pipe
- RegularPump
- Sink
- Source
- Thermostat
- ThreeWayValve
- TwoWayValve

Figure 3-13: PILS Toolbox

When you place an element on the canvas, it will show up with a specific shape. For some shapes it is possible to change the orientation of the drawing on the canvas. This is done so you can make some nicer looking models. Changing the orientation is done by assigning the appropriate value to a specific tagged value (*orientation*). If not already open, you can open the tagged value window by pressing Ctrl+Shift+6.
Although all elements have this tagged value, not all elements react on it (yet). This is work in progress. (Or see whether you can add it yourself to the profile definition☺).

Source and Sink will react on WE, EW settings.

Figure 3-14 shows the effect of changing the orientation for a sink.



Tagged Values

PILS::SystemEnd (Sink)	
materialName	
PILS::Sink (Sink)	
maxConnections	1
PILS::Node (Sink)	
orientation	WE

Figure 3-14 Sink element with west to east orientation.

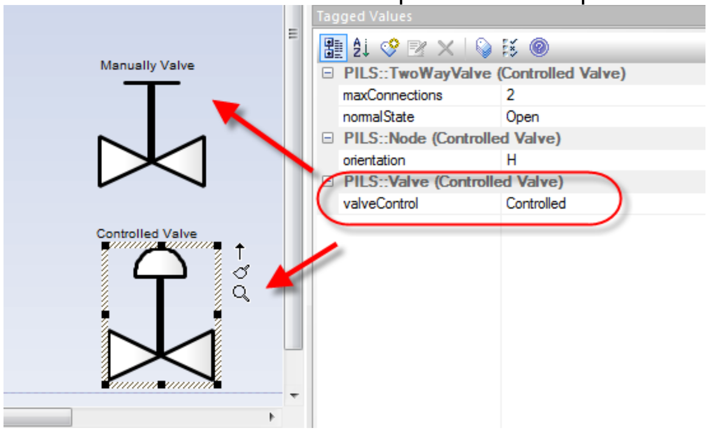
Matching the pipes exactly to the connection points of the other elements is manually done.

The naming of the elements is not restricted, but you should be aware of the following:

- The names are, as is, used for generating the code for the PLC.

- Referencing instances from within the state machine requires using exactly the same names. There is (unfortunately) no automatic checking or selection possible within EA. Constraint checks can be added to verify this constraint before actual generation.
- The naming we use is based on conventions within in of our projects but has no direct relationship with this case. All our names start with MDL_.

Valves:
Valves have a tagged value to define the mode of control: manually or (automatically) controlled. This value has also impact on the shape that is drawn. See figure 3.15:

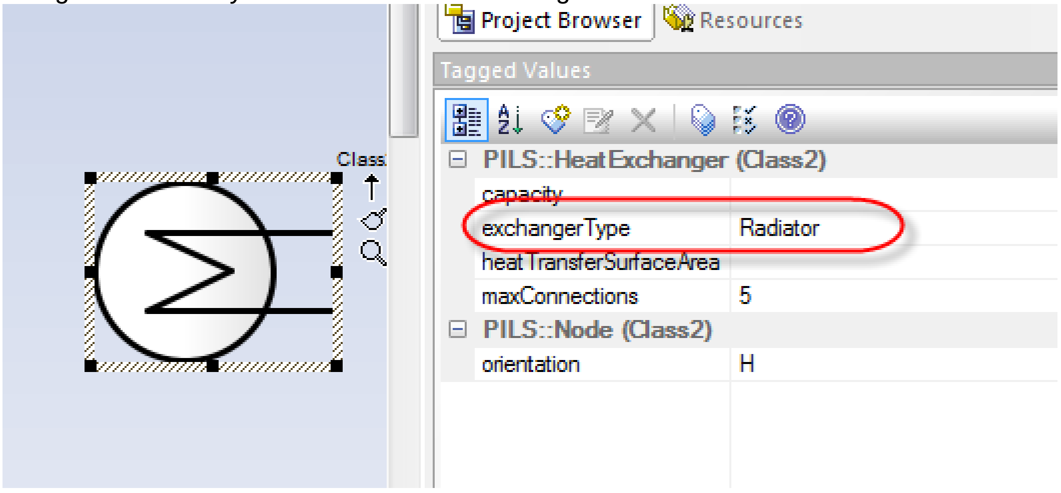


PILS::TwoWayValve (Controlled Valve)	
maxConnections	2
normalState	Open
PILS::Node (Controlled Valve)	
orientation	H
PILS::Valve (Controlled Valve)	
valveControl	Controlled

Figure 3-15: Changing control mode of valve

Manually controlled valves are assumed to be controlled by the operator/user and not the PLC.

HeatExchanger:
HeatExchanger has a tagged value that define the kind of heat exchanger. Two types are recognized currently: boiler and radiator. See figure 3-16:



PILS::HeatExchanger (Class2)	
capacity	
exchangerType	Radiator
heatTransferSurfaceArea	
maxConnections	5
PILS::Node (Class2)	
orientation	H

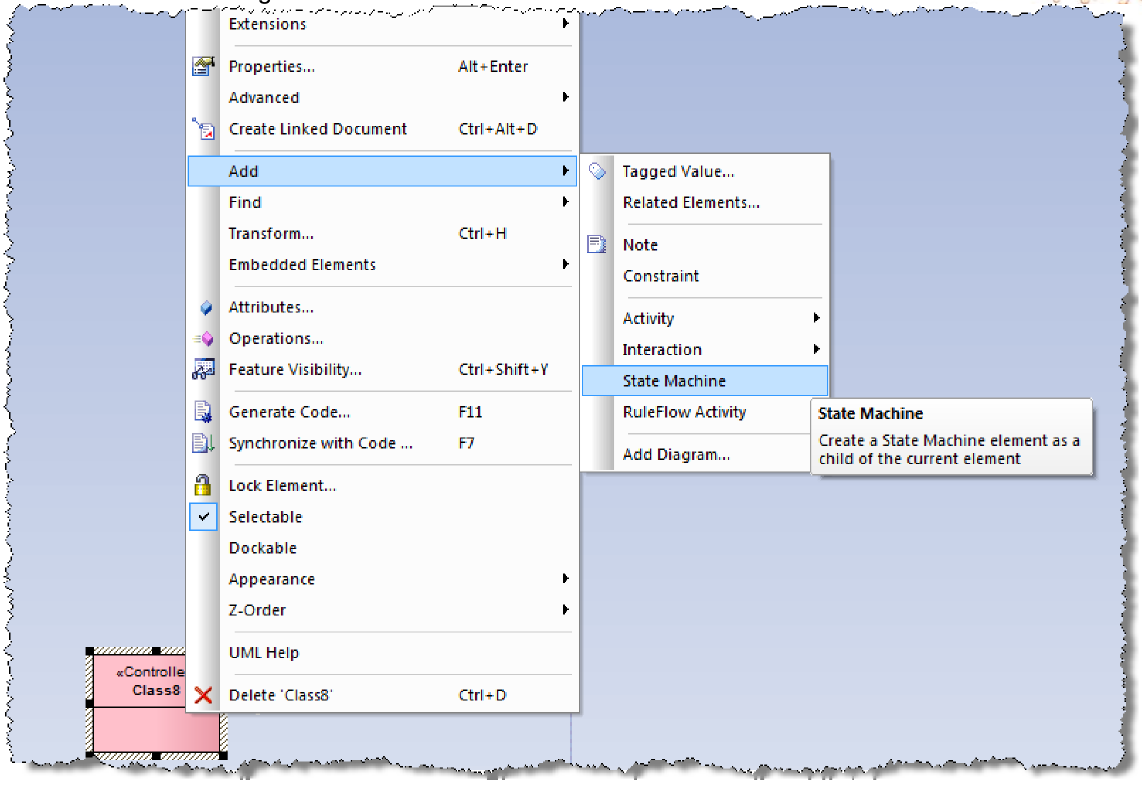
Figure 3-16: Changing type of Heat Exchanger

The different types don't have different shapes but are translated to different classes (function blocks) by the generator.

Most other tagged values are actually ignored by the generator as it is now.

3.3.3 Behaviour

The generator will only look for state machines which belong to <<Controller>> type of instances. To add a state machine to a <<Controller>> right click the instance, select Add – State Machine. See figure 3-17.



State Machine
Create a State Machine element as a child of the current element

Figure 3-17: Adding a state machine to a Controller instance

To open the state machine diagram, double click on it in the Project Browser. See figure 3-18.

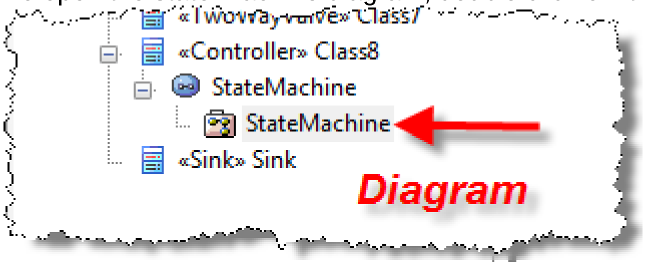


Figure 3-18: Opening State Machine diagram

The state machine generator doesn't support nested states.

Within the context of the state machine you have the possibility:

- To reference the model elements by name
- To reference the input and output attributes as defined in chapter 3.2.3 (for as far as a concept has additional attributes)
- To reference the operations as defined in chapter 3.2.3. (again for as far as a concept has additional operations.

You must enter this information as text, there is no automatic checking/enforcement by EA (unfortunately). And the entered information must be correct, because it is directly copied to the generated PLC code.

The expressions that are entered as guards or actions must also be correct PLC code. They too are copied to the generated code.

We have decided to split the control behaviour into 5 parallel state machines, each controlling a small piece of the total system (see also the description of the reference implementation). Figure shows what part of the model is controlled by each controller and its contained state machine.

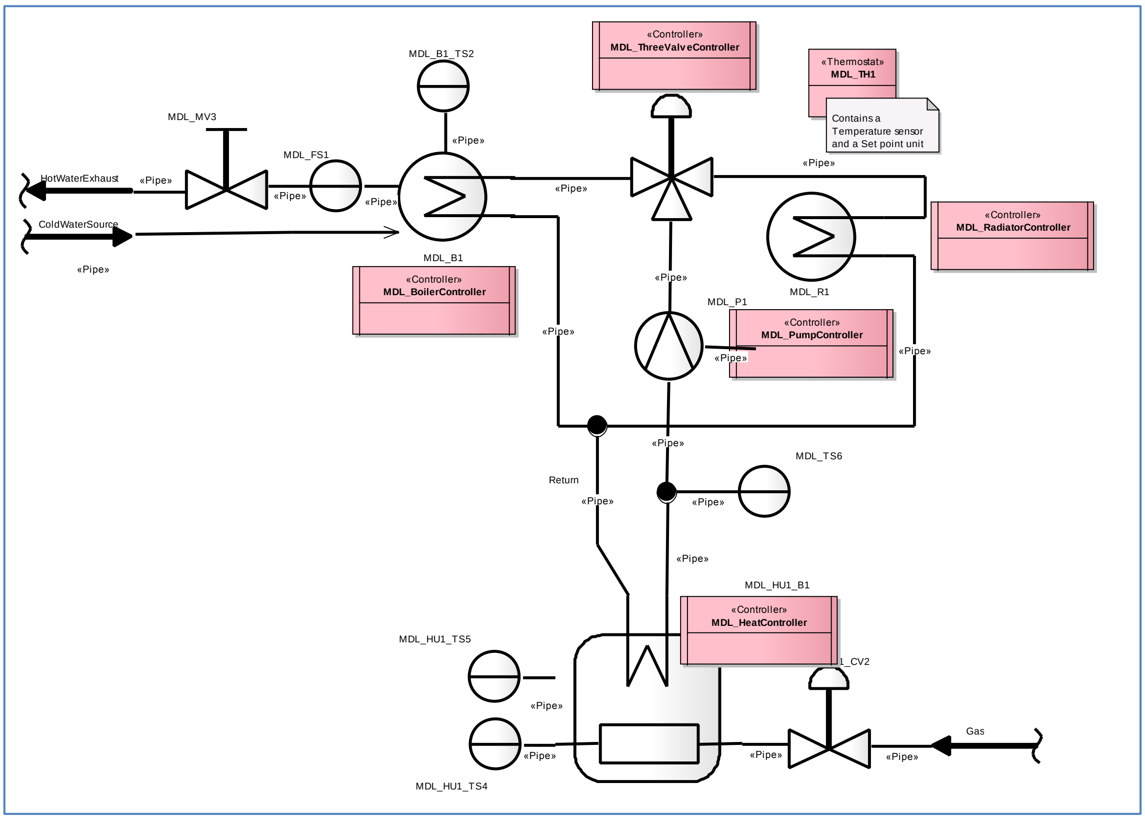


Figure 3-19: 5 Controllers each responsible for a part of the model

Following figures show the 5 state machines.

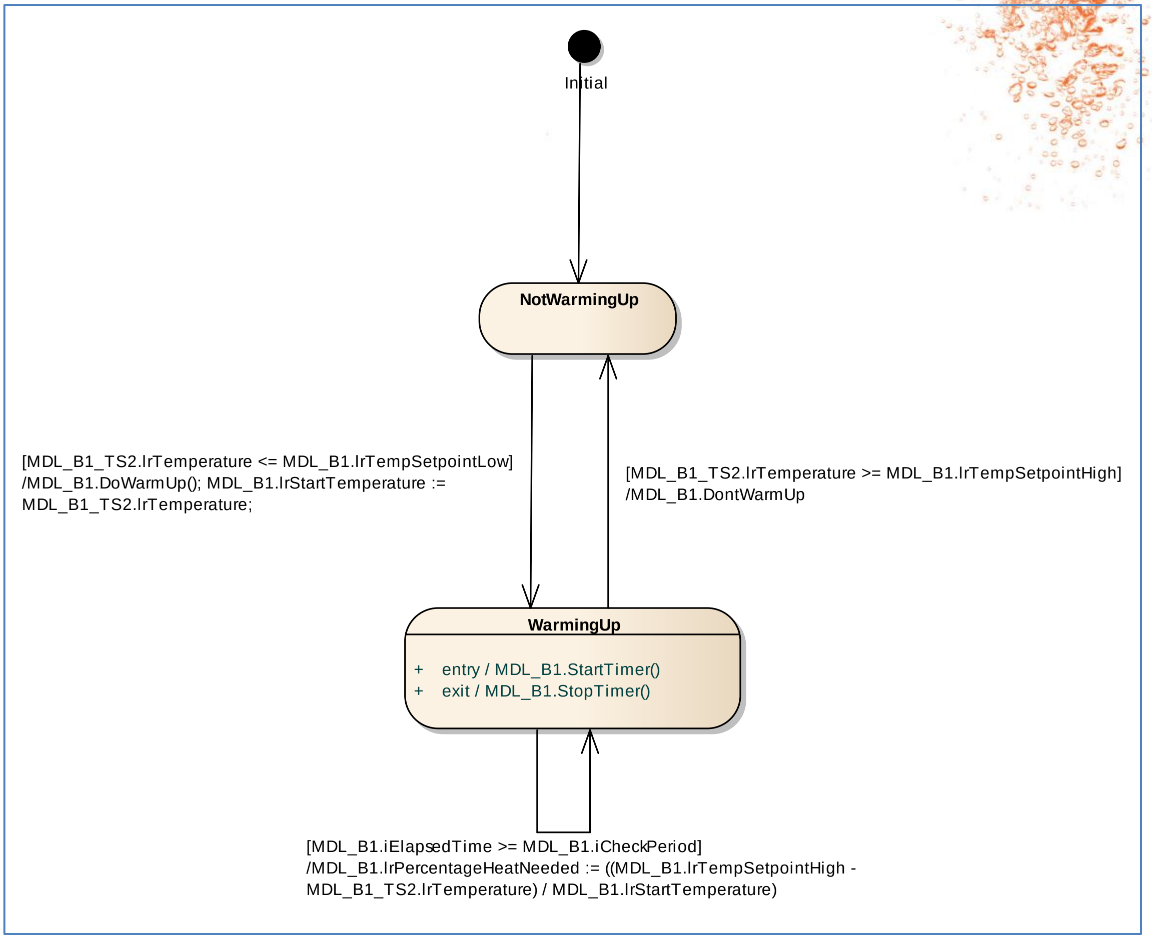

Initial
NotWarmingUp
[MDL_B1_TS2.lrTemperature <= MDL_B1.lrTempSetpointLow] /MDL_B1.DoWarmUp(); MDL_B1.lrStartTemperature := MDL_B1_TS2.lrTemperature;
[MDL_B1_TS2.lrTemperature >= MDL_B1.lrTempSetpointHigh] /MDL_B1.DontWarmUp
WarmingUp
+ entry / MDL_B1.StartTimer()
+ exit / MDL_B1.StopTimer()
[MDL_B1.iElapsedTime >= MDL_B1.iCheckPeriod] /MDL_B1.lrPercentageHeatNeeded := ((MDL_B1.lrTempSetpointHigh - MDL_B1_TS2.lrTemperature) / MDL_B1.lrStartTemperature)

Figure 3-20: SM Boiler Controller

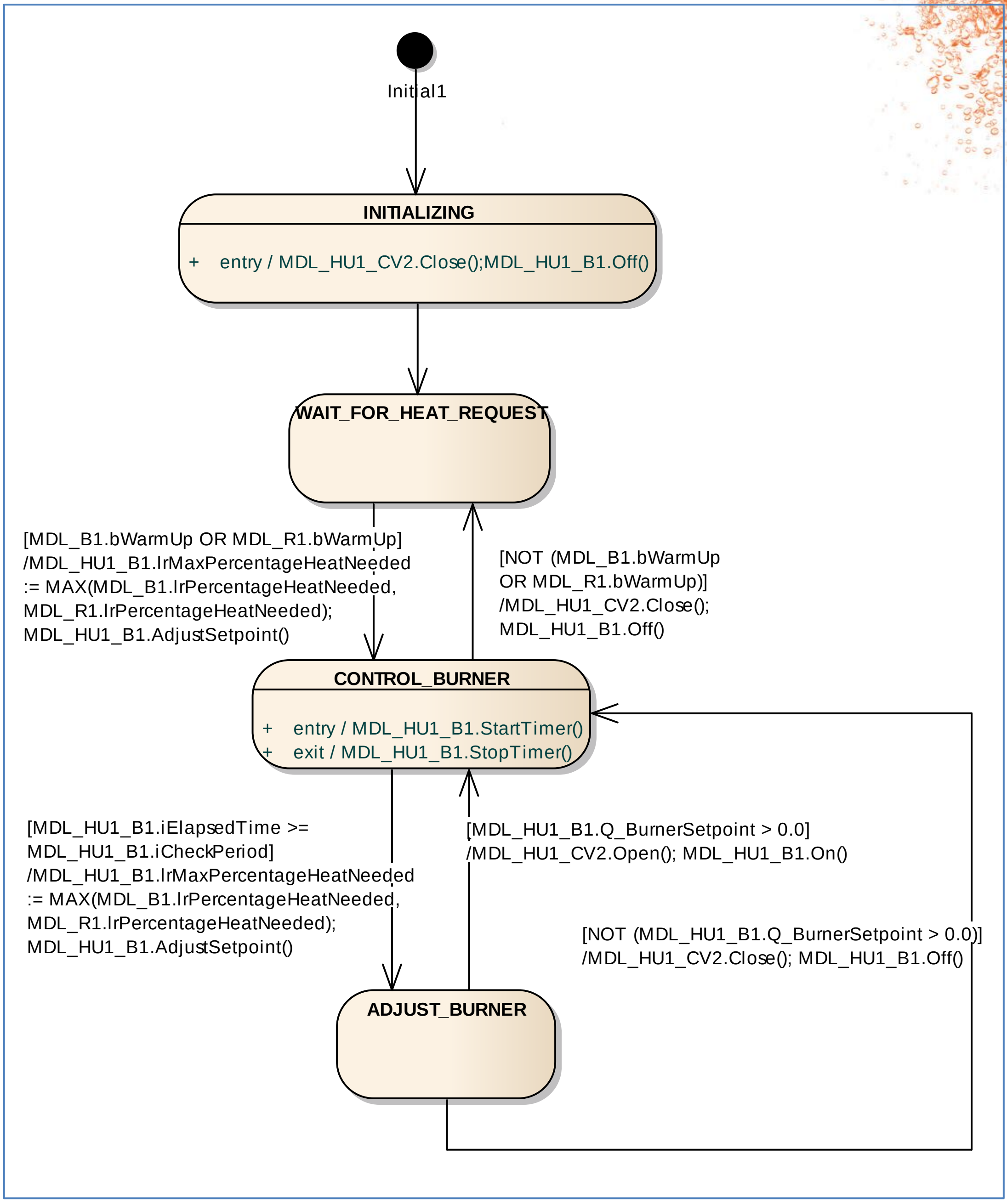

Initial1
INITIALIZING
+ entry / MDL_HU1_CV2.Close();MDL_HU1_B1.Off()
WAIT_FOR_HEAT_REQUEST
[MDL_B1.bWarmUp OR MDL_R1.bWarmUp] /MDL_HU1_B1.lrMaxPercentageHeatNeeded := MAX(MDL_B1.lrPercentageHeatNeeded, MDL_R1.lrPercentageHeatNeeded); MDL_HU1_B1.AdjustSetpoint()
[NOT (MDL_B1.bWarmUp OR MDL_R1.bWarmUp)] /MDL_HU1_CV2.Close(); MDL_HU1_B1.Off()
CONTROL_BURNER
+ entry / MDL_HU1_B1.StartTimer()
+ exit / MDL_HU1_B1.StopTimer()
[MDL_HU1_B1.iElapsedTime >= MDL_HU1_B1.iCheckPeriod] /MDL_HU1_B1.lrMaxPercentageHeatNeeded := MAX(MDL_B1.lrPercentageHeatNeeded, MDL_R1.lrPercentageHeatNeeded); MDL_HU1_B1.AdjustSetpoint()
[MDL_HU1_B1.Q_BurnerSetpoint > 0.0] /MDL_HU1_CV2.Open(); MDL_HU1_B1.On()
[NOT (MDL_HU1_B1.Q_BurnerSetpoint > 0.0)] /MDL_HU1_CV2.Close(); MDL_HU1_B1.Off()
ADJUST_BURNER

Figure 3-21: SM Heat Controller

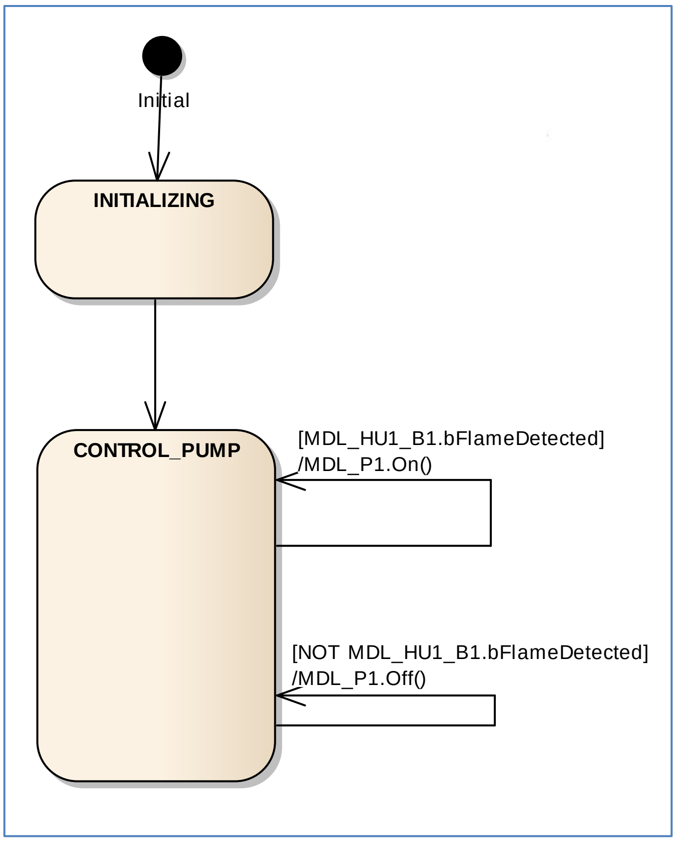


Figure 3-22: SM Pump Controller

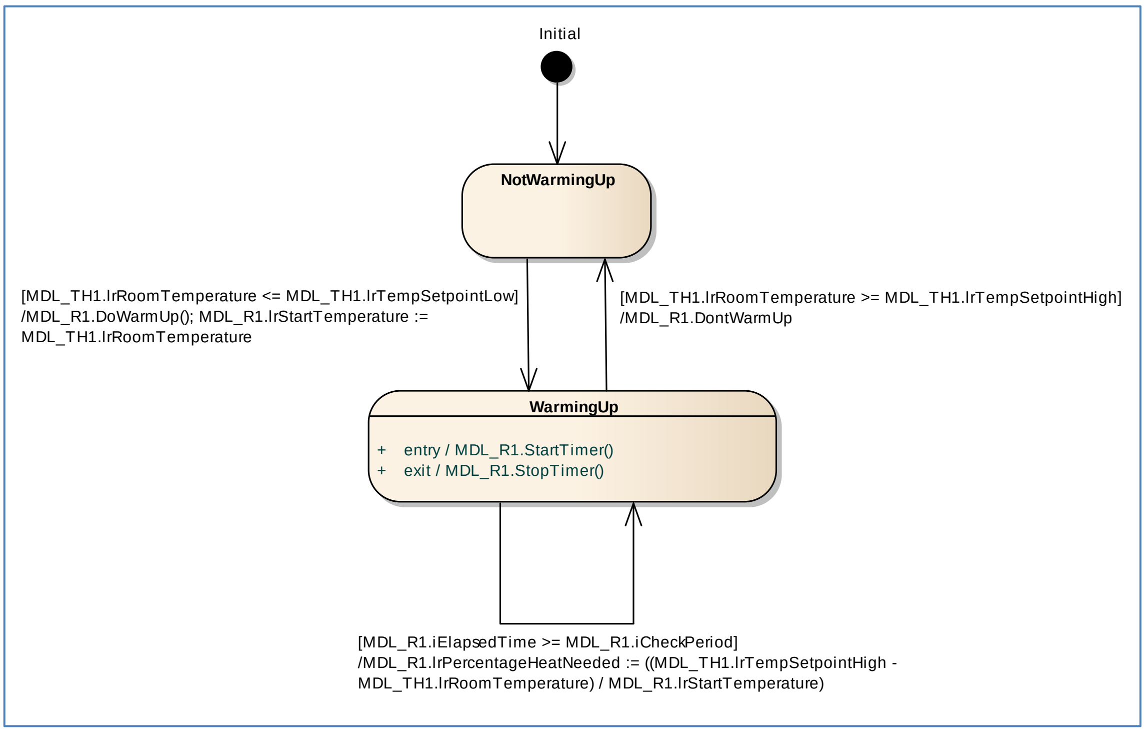

Initial
NotWarmingUp
[MDL_TH1.lrRoomTemperature <= MDL_TH1.lrTempSetpointLow]
/MDL_R1.DoWarmUp(); MDL_R1.lrStartTemperature :=
MDL_TH1.lrRoomTemperature
[MDL_TH1.lrRoomTemperature >= MDL_TH1.lrTempSetpointHigh]
/MDL_R1.DontWarmUp
WarmingUp
+ entry / MDL_R1.StartTimer()
+ exit / MDL_R1.StopTimer()
[MDL_R1.iElapsedTime >= MDL_R1.iCheckPeriod]
/MDL_R1.lrPercentageHeatNeeded := ((MDL_TH1.lrTempSetpointHigh -
MDL_TH1.lrRoomTemperature) / MDL_R1.lrStartTemperature)

Figure 3-23: SM Radiator Controller

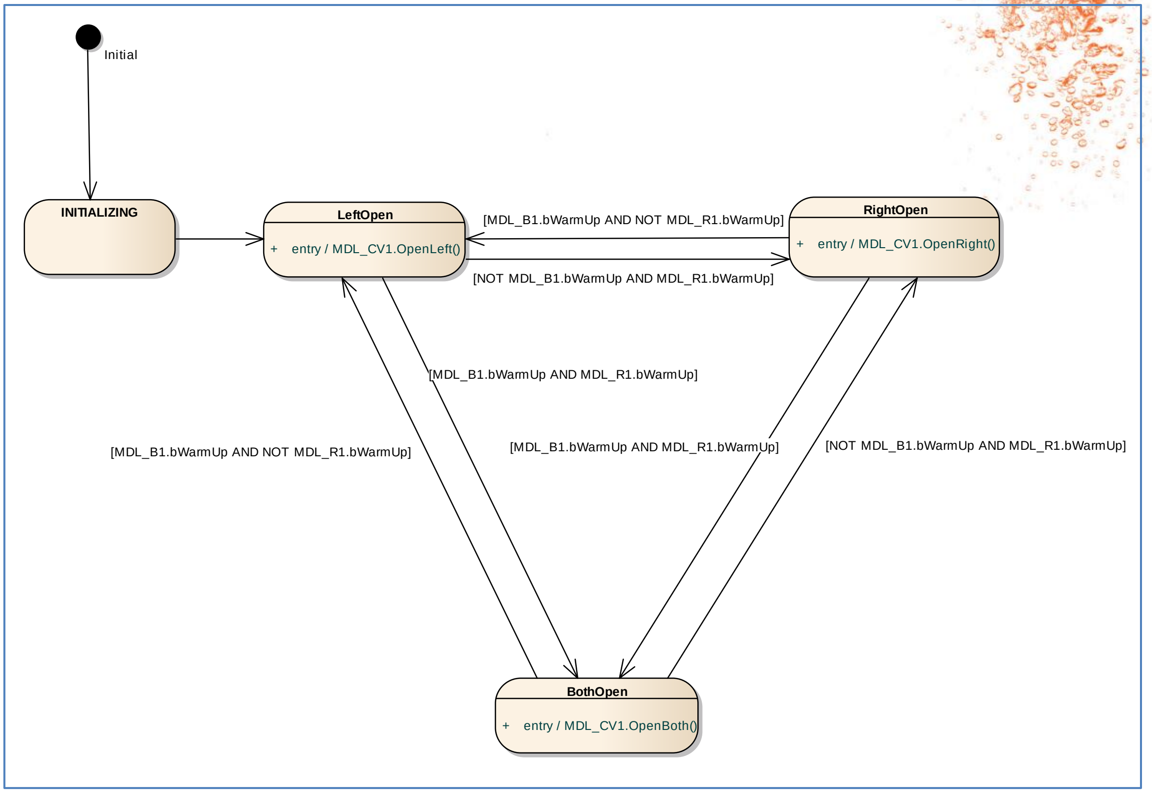

Initial
INITIALIZING
LeftOpen
+ entry / MDL_CV1.OpenLeft()
RightOpen
+ entry / MDL_CV1.OpenRight()
BothOpen
+ entry / MDL_CV1.OpenBoth()
[MDL_B1.bWarmUp AND NOT MDL_R1.bWarmUp]
[NOT MDL_B1.bWarmUp AND MDL_R1.bWarmUp]
[MDL_B1.bWarmUp AND MDL_R1.bWarmUp]
[MDL_B1.bWarmUp AND NOT MDL_R1.bWarmUp]
[MDL_B1.bWarmUp AND MDL_R1.bWarmUp]
[NOT MDL_B1.bWarmUp AND MDL_R1.bWarmUp]

Figure 3-24: SM Three Valve Controller

3.4 Generator

The PPL2010 tutorial describes the basic setup and way of working with EA and Eclipse/Xpand. Look there if you want to know the exact details. Otherwise you can just import the provided generator project into a new workspace with Eclipse.

Look at the generator templates themselves to see how we resolved the generator part. The generator is rather limited in size.

We didn't add generation templates for simulation of the model. There was no time for it.

The generator output consists of several files containing code snippets which need to be pasted in the correct places within the PLC development environment. As stated before, in a real production environment we add automatic merge facilities.

3.4.1 Setting up the generator project

Download the generator zip file (also includes the meta model and model EA files).
Extract the content of the zip file a separate folder which becomes a new workspace within Eclipse. See figure 3-25.

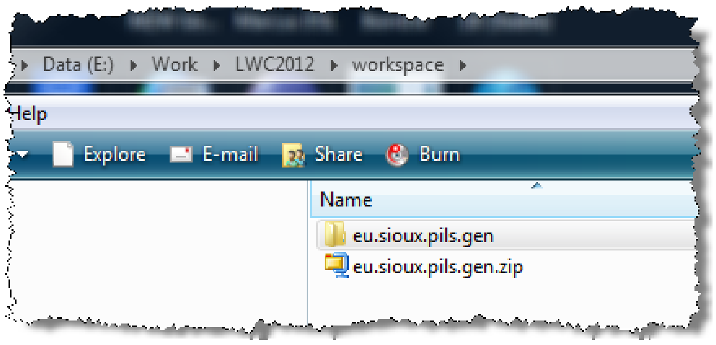


Figure 3-25: Download and unzip eclipse project

Start up Eclipse (assuming Eclipse is already installed) and select the workspace folder (figure 3-26).

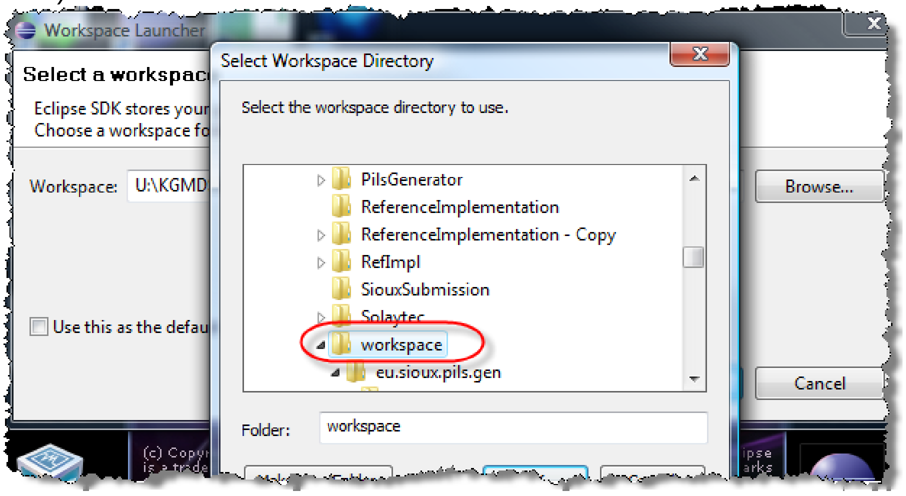


Figure 3-26: Select workspace

Click away the welcome tab.
Import the project into the workspace: Click File – Import, open the General tab and click “Existing Projects into Workspace”. Click Next. Click Browse (see figure 3-27).

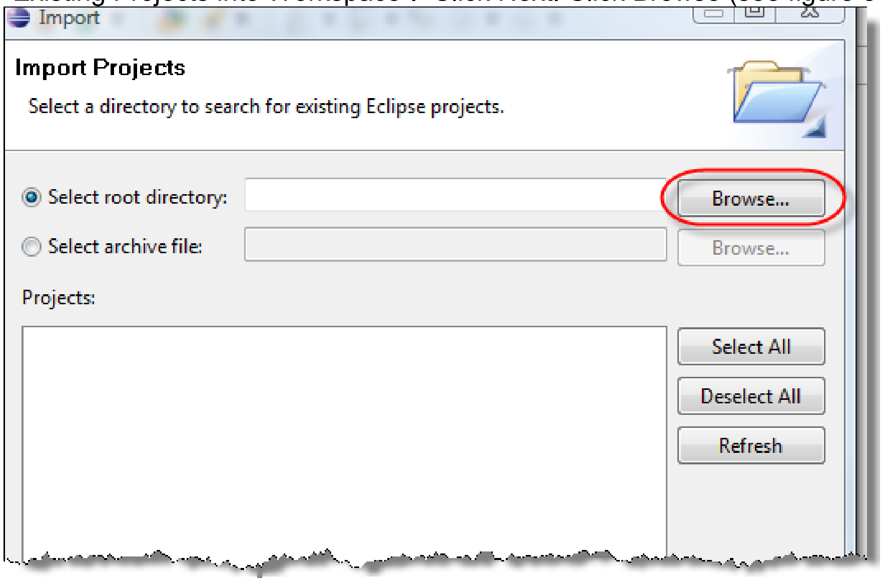


Figure 3-27: Open workspace to import project

And select nothing, but just click Ok.
The project should show up now in the list (figure 3-28).

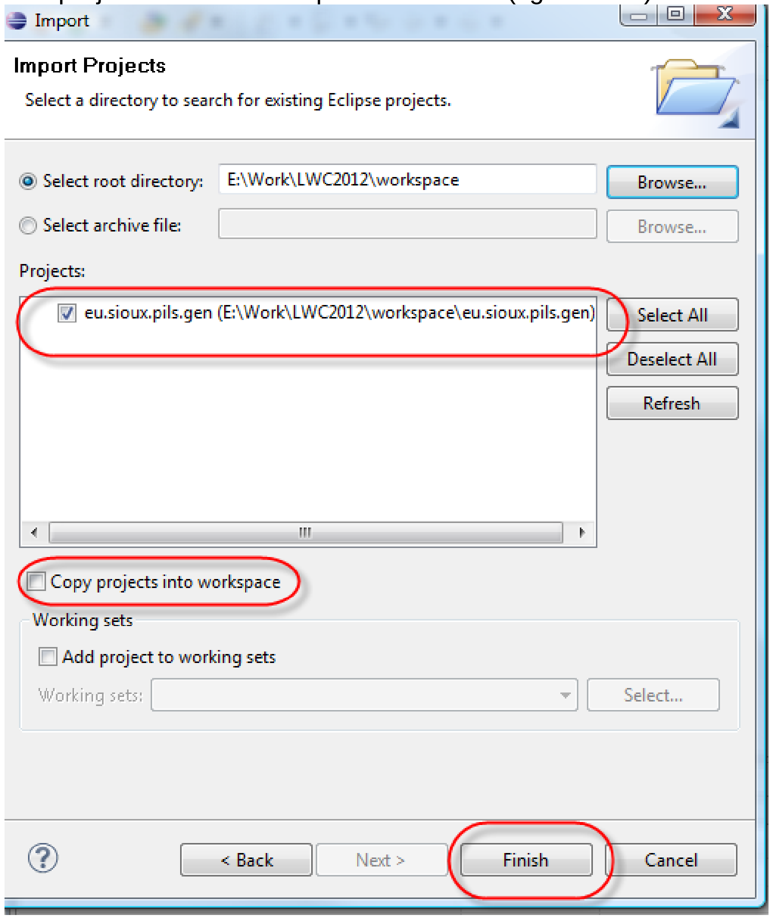

Import
Import Projects
Select a directory to search for existing Eclipse projects.
Select root directory:
E:\Work\LWC2012\workspace
Browse...
Select archive file:
Browse...
Projects:
eu.sioux.pils.gen (E:\Work\LWC2012\workspace\eu.sioux.pils.gen)
Select All
Deselect All
Refresh
Copy projects into workspace
Working sets
Add project to working sets
Working sets:
Select...
< Back
Next >
Finish
Cancel

Figure 3-28: Project selected for import.

Click Finish, and after a few moments the project should show up in the Package Explorer. Figure 3-29 gives an overview of the content of the project.

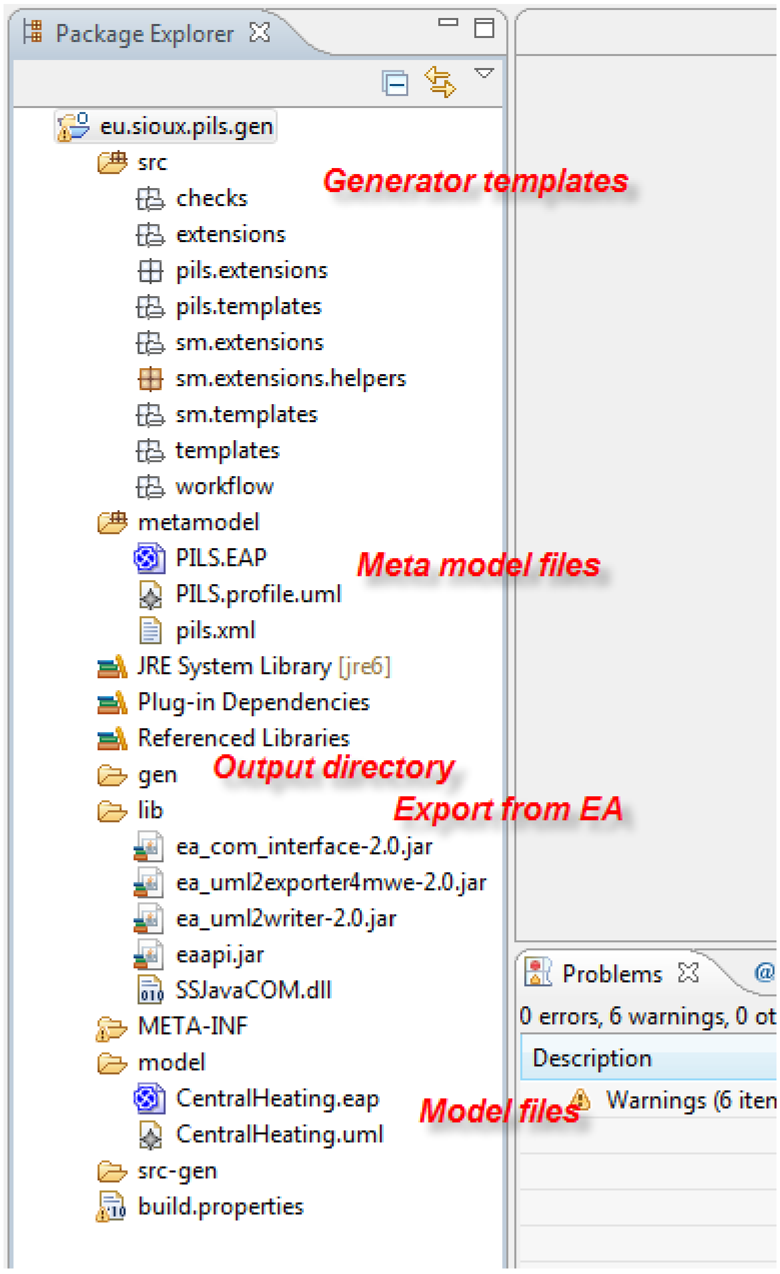


Figure 3-29: Content of generator project

3.4.2 Generation

Generation is controlled by 2 MWE files: one for exporting the PILS profile and one for exporting the model and generating code.
These workflow files are located in the src/workflow folder. Both have an associated properties file that allows setting some properties. They should be fine as downloaded. If you want to test out another model EA file you need to make adaptations to them.
Workflows:

- MetaModelToUml2.mwe: Export Profile from EA into Eclipse.
- ModelToPLC.mwe: Export Model from EA into Eclipse and generate PLC code.

Run these workflows: right click on the files and select Run as - MWE workflow. See figure 3-30.

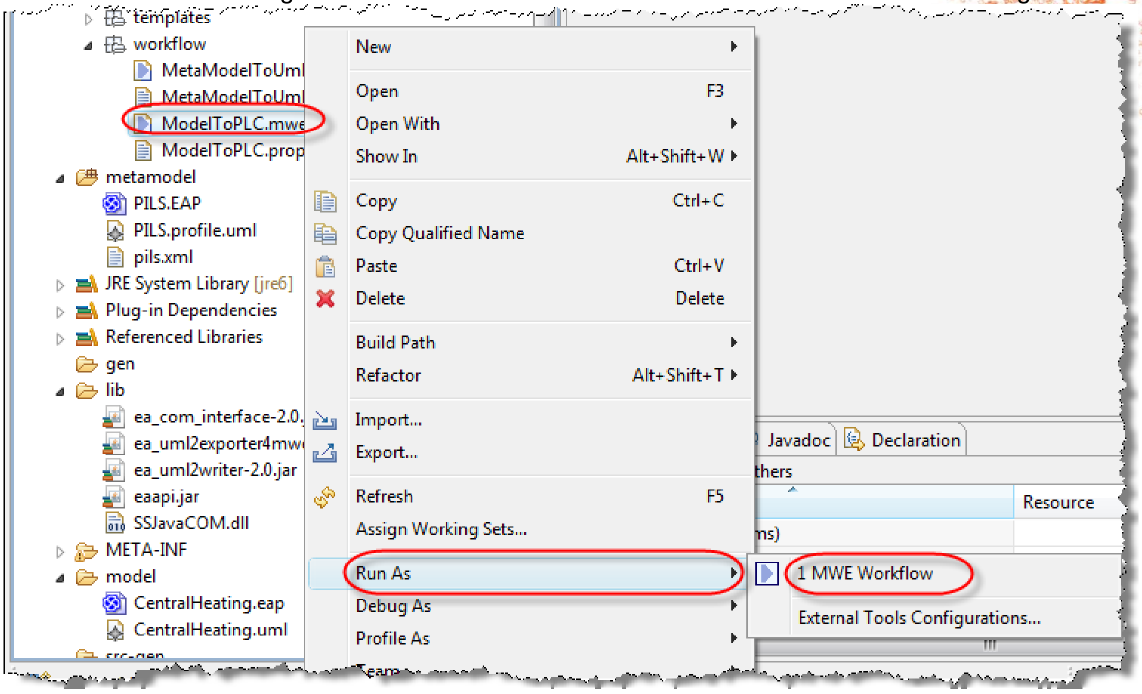


Figure 3-30: Running workflow

Note that it is possible to run the workflow outside Eclipse via a bat-file, which allows it to be executed as part of an automatic build process. It is also possible to initiate the generation from within EA, without starting up Eclipse.
We didn't work out these possibilities further.

Generation starts in the file src/templates/Root.xpt. From there you can follow the flow of generation.

Note:
The output console for running the ModelToPLC.mwe will show some errors with respect to *StereotypeType*. You can safely ignore these errors. The generation still goes fine.

3.4.3 Copying to PLC development environment

After successful generation the folder *gen* contains the generated stuff.

The structural code is placed in 2 files: one for the normal controller part and one for the simulation (we generate this part – but don't generate any further simulation specific code).

The state machine code is put into separate folder, one for each state machine found.

See figure.

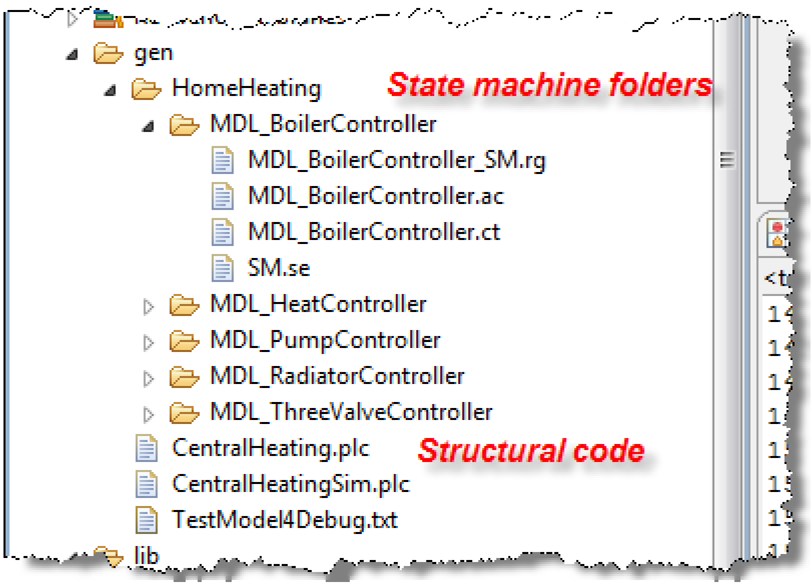
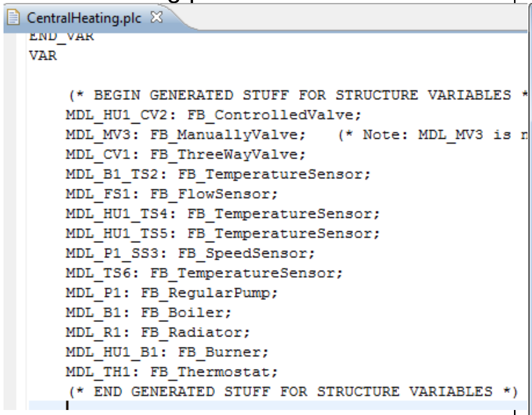
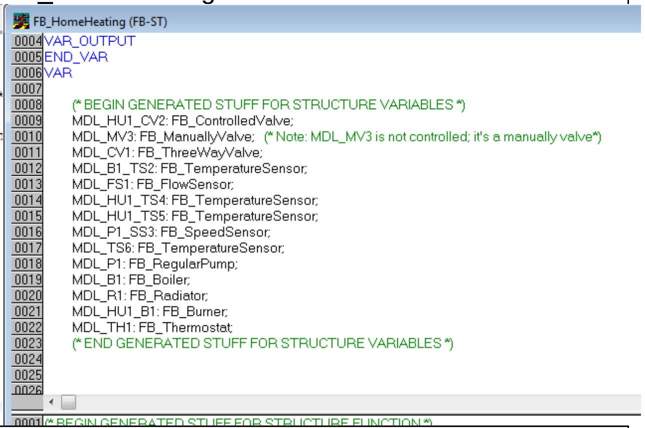
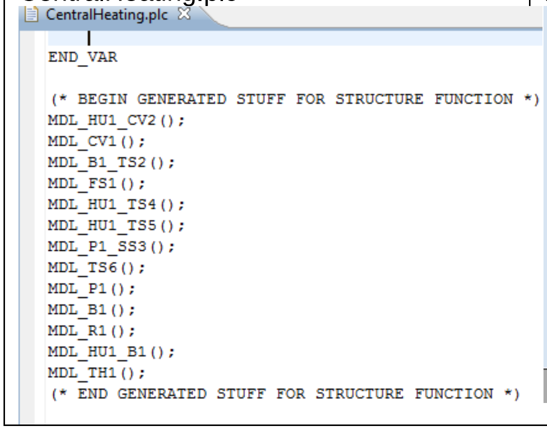
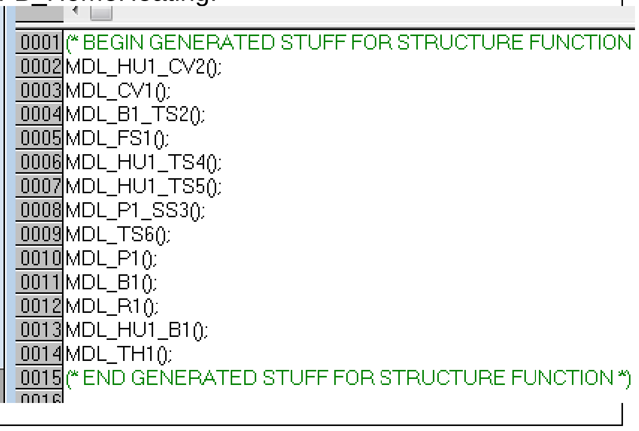


Figure 3-31: Generation output

Next you need to copy the generated code into the PLC environment. This takes a couple of minutes.

Following table explains where to put each piece of code.

Generation	***PLC environment***
CentralHeating.plc 	FB_HomeHeating: 
CentralHeating.plc 	FB_HomeHeating: 

CentralHeating.plc

```
END_VAR
VAR

    (* BEGIN GENERATED STUFF FOR STRUCTURE VARIABLES *
    MDL_HU1_CV2: FB_ControlledValve;
    MDL_MV3: FB_ManuallyValve;    (* Note: MDL_MV3 is r
    MDL_CV1: FB_ThreeWayValve;
    MDL_B1_TS2: FB_TemperatureSensor;
    MDL_FS1: FB_FlowSensor;
    MDL_HU1_TS4: FB_TemperatureSensor;
    MDL_HU1_TS5: FB_TemperatureSensor;
    MDL_P1_SS3: FB_SpeedSensor;
    MDL_TS6: FB_TemperatureSensor;
    MDL_P1: FB_RegularPump;
    MDL_B1: FB_Boiler;
    MDL_R1: FB_Radiator;
    MDL_HU1_B1: FB_Burner;
    MDL_TH1: FB_Thermostat;
    (* END GENERATED STUFF FOR STRUCTURE VARIABLES *)
```

FB_HomeHeating (FB-ST)

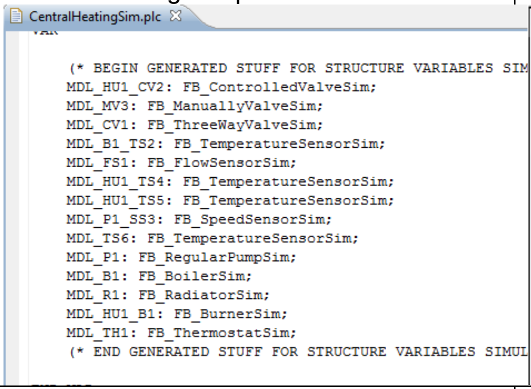
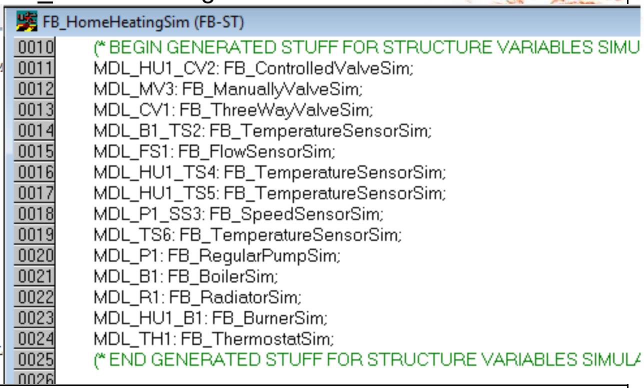
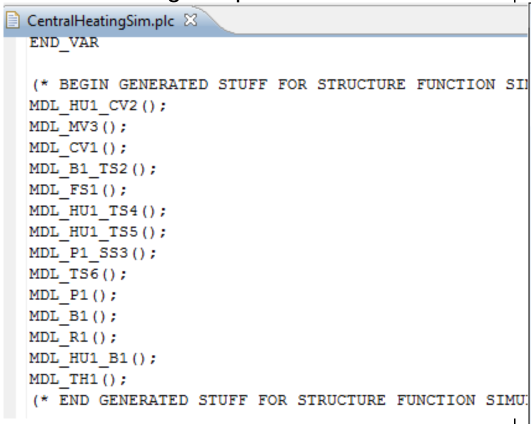
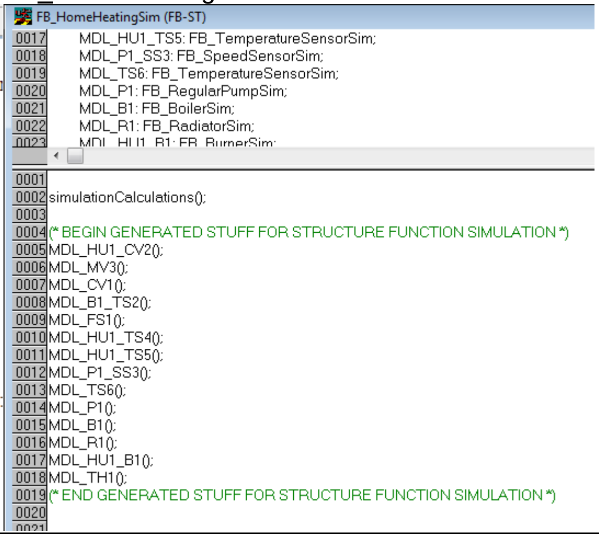
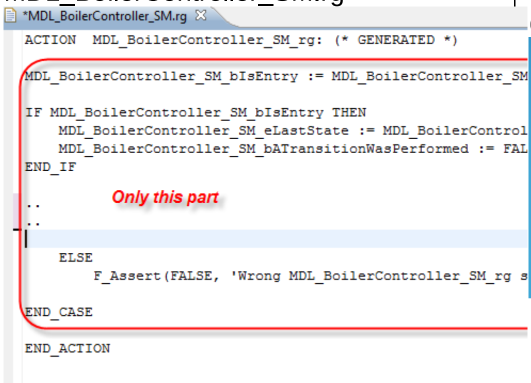
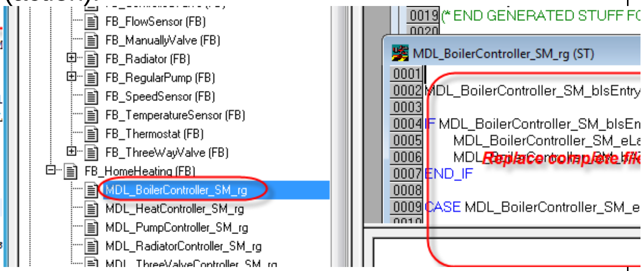
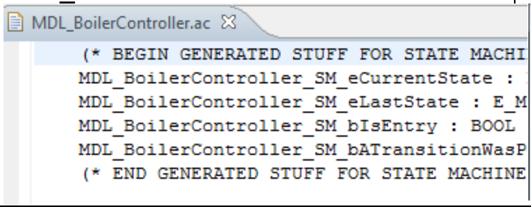
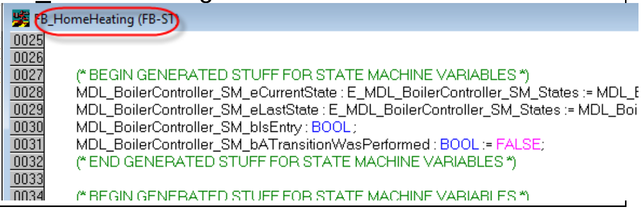
```
VAR_OUTPUT
END_VAR
VAR

    (* BEGIN GENERATED STUFF FOR STRUCTURE VARIABLES *)
    MDL_HU1_CV2: FB_ControlledValve;
    MDL_MV3: FB_ManuallyValve;  (* Note: MDL_MV3 is not controlled; it's a manually valve*)
    MDL_CV1: FB_ThreeWayValve;
    MDL_B1_TS2: FB_TemperatureSensor;
    MDL_FS1: FB_FlowSensor;
    MDL_HU1_TS4: FB_TemperatureSensor;
    MDL_HU1_TS5: FB_TemperatureSensor;
    MDL_P1_SS3: FB_SpeedSensor;
    MDL_TS6: FB_TemperatureSensor;
    MDL_P1: FB_RegularPump;
    MDL_B1: FB_Boiler;
    MDL_R1: FB_Radiator;
    MDL_HU1_B1: FB_Burner;
    MDL_TH1: FB_Thermostat;
    (* END GENERATED STUFF FOR STRUCTURE VARIABLES *)
```

CentralHeating.plc

```
END_VAR

(* BEGIN GENERATED STUFF FOR STRUCTURE FUNCTION *)
MDL_HU1_CV2();
MDL_CV1();
MDL_B1_TS2();
MDL_FS1();
MDL_HU1_TS4();
MDL_HU1_TS5();
MDL_P1_SS3();
MDL_TS6();
MDL_P1();
MDL_B1();
MDL_R1();
MDL_HU1_B1();
MDL_TH1();
(* END GENERATED STUFF FOR STRUCTURE FUNCTION *)
```

FB_HomeHeating:

```
(* BEGIN GENERATED STUFF FOR STRUCTURE FUNCTION
MDL_HU1_CV2();
MDL_CV1();
MDL_B1_TS2();
MDL_FS1();
MDL_HU1_TS4();
MDL_HU1_TS5();
MDL_P1_SS3();
MDL_TS6();
MDL_P1();
MDL_B1();
MDL_R1();
MDL_HU1_B1();
MDL_TH1();
(* END GENERATED STUFF FOR STRUCTURE FUNCTION *)
```

Generation	***PLC environment***
CentralHeatingSim.plc	FB_HomeHeatingSim
	
CentralHeatingSim.plc	FB_HomeHeatingSim
	
For each state machine – taking BoilerController as an example:	
MDL_BoilerController_SM.rg	FB_HomeHeating : MDL_BoilerController_SM_rg (action):
	
MDL_BoilerController.ac	FB_HomeHeating
	

CentralHeatingSim.plc

```
    (* BEGIN GENERATED STUFF FOR STRUCTURE VARIABLES SIM
    MDL_HU1_CV2: FB_ControlledValveSim;
    MDL_MV3: FB_ManuallyValveSim;
    MDL_CV1: FB_ThreeWayValveSim;
    MDL_B1_TS2: FB_TemperatureSensorSim;
    MDL_FS1: FB_FlowSensorSim;
    MDL_HU1_TS4: FB_TemperatureSensorSim;
    MDL_HU1_TS5: FB_TemperatureSensorSim;
    MDL_P1_SS3: FB_SpeedSensorSim;
    MDL_TS6: FB_TemperatureSensorSim;
    MDL_P1: FB_RegularPumpSim;
    MDL_B1: FB_BoilerSim;
    MDL_R1: FB_RadiatorSim;
    MDL_HU1_B1: FB_BurnerSim;
    MDL_TH1: FB_ThermostatSim;
    (* END GENERATED STUFF FOR STRUCTURE VARIABLES SIMUL
```

FB_HomeHeatingSim (FB-ST)

```
(* BEGIN GENERATED STUFF FOR STRUCTURE VARIABLES SIMU
MDL_HU1_CV2: FB_ControlledValveSim;
MDL_MV3: FB_ManuallyValveSim;
MDL_CV1: FB_ThreeWayValveSim;
MDL_B1_TS2: FB_TemperatureSensorSim;
MDL_FS1: FB_FlowSensorSim;
MDL_HU1_TS4: FB_TemperatureSensorSim;
MDL_HU1_TS5: FB_TemperatureSensorSim;
MDL_P1_SS3: FB_SpeedSensorSim;
MDL_TS6: FB_TemperatureSensorSim;
MDL_P1: FB_RegularPumpSim;
MDL_B1: FB_BoilerSim;
MDL_R1: FB_RadiatorSim;
MDL_HU1_B1: FB_BurnerSim;
MDL_TH1: FB_ThermostatSim;
(* END GENERATED STUFF FOR STRUCTURE VARIABLES SIMULA
```

CentralHeatingSim.plc

```
END_VAR

 (* BEGIN GENERATED STUFF FOR STRUCTURE FUNCTION SI
MDL_HU1_CV2();
MDL_MV3();
MDL_CV1();
MDL_B1_TS2();
MDL_FS1();
MDL_HU1_TS4();
MDL_HU1_TS5();
MDL_P1_SS3();
MDL_TS6();
MDL_P1();
MDL_B1();
MDL_R1();
MDL_HU1_B1();
MDL_TH1();
 (* END GENERATED STUFF FOR STRUCTURE FUNCTION SIMU
```

FB_HomeHeatingSim (FB-ST)

```
MDL_HU1_TS5: FB_TemperatureSensorSim;
MDL_P1_SS3: FB_SpeedSensorSim;
MDL_TS6: FB_TemperatureSensorSim;
MDL_P1: FB_RegularPumpSim;
MDL_B1: FB_BoilerSim;
MDL_R1: FB_RadiatorSim;
MDL_HU1_B1: FB_BurnerSim;
```

```

simulationCalculations();

(* BEGIN GENERATED STUFF FOR STRUCTURE FUNCTION SIMULATION *)
MDL_HU1_CV2();
MDL_MV3();
MDL_CV1();
MDL_B1_TS2();
MDL_FS1();
MDL_HU1_TS4();
MDL_HU1_TS5();
MDL_P1_SS3();
MDL_TS6();
MDL_P1();
MDL_B1();
MDL_R1();
MDL_HU1_B1();
MDL_TH1();
(* END GENERATED STUFF FOR STRUCTURE FUNCTION SIMULATION *)
```

*MDL_BoilerController_SM.rg

```
ACTION  MDL_BoilerController_SM_rg: (* GENERATED *)

MDL_BoilerController_SM_bIsEntry := MDL_BoilerController_SM

IF MDL_BoilerController_SM_bIsEntry THEN
    MDL_BoilerController_SM_eLastState := MDL_BoilerControl
    MDL_BoilerController_SM_bATransitionWasPerformed := FAL
END_IF

..
..

    ELSE
        F_Assert(FALSE, 'Wrong MDL_BoilerController_SM_rg s

END_CASE

END_ACTION
```

```
(* END GENERATED STUFF FO
```

MDL_BoilerController_SM_rg (ST)

```

MDL_BoilerController_SM_bIsEntry

IF MDL_BoilerController_SM_bIsEn
    MDL_BoilerController_SM_eLa
    MDL_BoilerController_SM_bA
END_IF

CASE MDL_BoilerController_SM_e
```

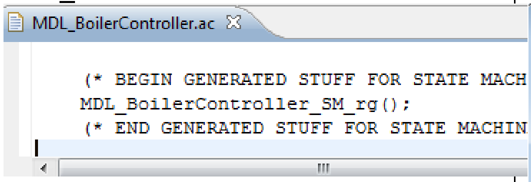
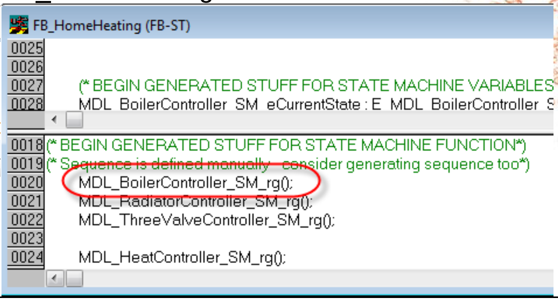
MDL_BoilerController.ac

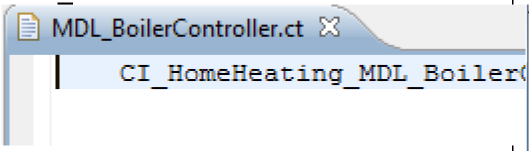
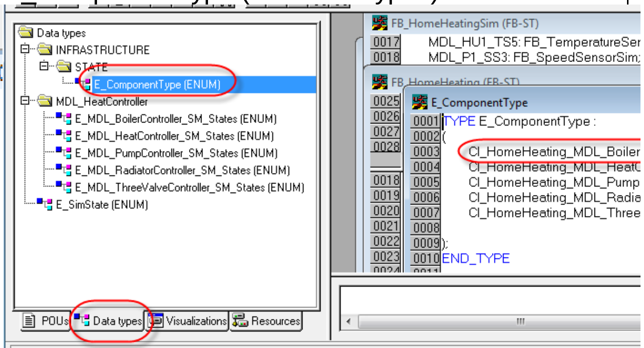
```
    (* BEGIN GENERATED STUFF FOR STATE MACHI
    MDL_BoilerController_SM_eCurrentState :
    MDL_BoilerController_SM_eLastState : E_M
    MDL_BoilerController_SM_bIsEntry : BOOL
    MDL_BoilerController_SM_bATransitionWasP
    (* END GENERATED STUFF FOR STATE MACHINE
```

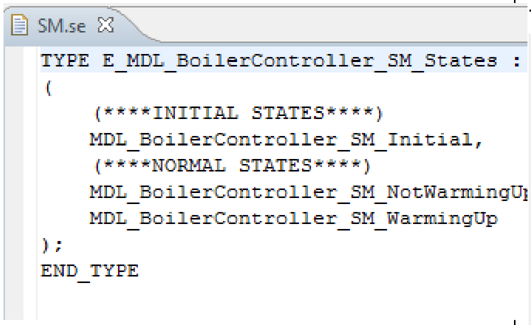
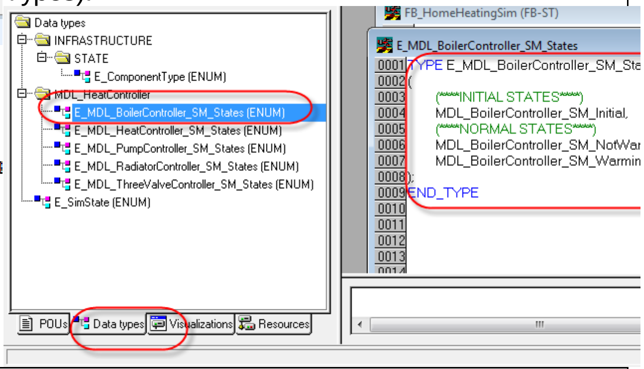
FB_HomeHeating (FB-ST)

```


(* BEGIN GENERATED STUFF FOR STATE MACHINE VARIABLES *)
MDL_BoilerController_SM_eCurrentState : E_MDL_BoilerController_SM_States := MDL_B
MDL_BoilerController_SM_eLastState : E_MDL_BoilerController_SM_States := MDL_Boi
MDL_BoilerController_SM_bIsEntry : BOOL;
MDL_BoilerController_SM_bATransitionWasPerformed : BOOL := FALSE;
(* END GENERATED STUFF FOR STATE MACHINE VARIABLES *)

(* BEGIN GENERATED STUFF FOR STATE MACHINE VARIABLES *)
```

Generation	***PLC environment***

MDL_BoilerController.ac

MDL_BoilerController.ac
`(* BEGIN GENERATED STUFF FOR STATE MACH`
`MDL_BoilerController_SM_rg();`
`(* END GENERATED STUFF FOR STATE MACHIN`
	FB_HomeHeating
	
	FB_HomeHeating (FB-ST)
	`(* BEGIN GENERATED STUFF FOR STATE MACHINE VARIABLES`
	`MDL_BoilerController_SM_eCurrentState : E_MDL_BoilerController_S`
	`(* BEGIN GENERATED STUFF FOR STATE MACHINE FUNCTION*)`
	`(* ~~Sequence is defined manually~~ - consider generating sequence too*)`
	`MDL_BoilerController_SM_rg();`
	`MDL_RadiatorController_SM_rg();`
	`MDL_ThreeValveController_SM_rg();`
	`MDL_HeatController_SM_rg();`

MDL_BoilerController.ct

MDL_BoilerController.ct
`CI_HomeHeating_MDL_Boiler(`
	E_ComponentType (tab Data Types):
	
	Data types
	INFRASTRUCTURE
	STATE
	E_ComponentType (ENUM)
	MDL_HeatController
	E_MDL_BoilerController_SM_States (ENUM)
	E_MDL_HeatController_SM_States (ENUM)
	E_MDL_PumpController_SM_States (ENUM)
	E_MDL_RadiatorController_SM_States (ENUM)
	E_MDL_ThreeValveController_SM_States (ENUM)
	E_SimState (ENUM)
	POUs | Data types | Visualizations | Resources
	FB_HomeHeatingSim (FB-ST)
	`MDL_HU1_TS5: FB_TemperatureSer`
	`MDL_P1_SS3: FB_SpeedSensorSim;`
	FB_HomeHeating (FB-ST)
	E_ComponentType
	`TYPE E_ComponentType :`
	`(`
	`    CI_HomeHeating_MDL_Boiler`
	`    CI_HomeHeating_MDL_HeatC`
	`    CI_HomeHeating_MDL_Pump`
	`    CI_HomeHeating_MDL_Radia`
	`    CI_HomeHeating_MDL_Three`
	`);`
	`END_TYPE`

SM.se:

SM.se
`TYPE E_MDL_BoilerController_SM_States :`
`(`
`    (****INITIAL STATES****)`
`    MDL_BoilerController_SM_Initial,`
`    (****NORMAL STATES****)`
`    MDL_BoilerController_SM_NotWarmingU`
`    MDL_BoilerController_SM_WarmingUp`
`);`
`END_TYPE`
	E_MDL_BoilerController_SM_States (tab Data Types):
	
	Data types
	INFRASTRUCTURE
	STATE
	E_ComponentType (ENUM)
	MDL_HeatController
	E_MDL_BoilerController_SM_States (ENUM)
	E_MDL_HeatController_SM_States (ENUM)
	E_MDL_PumpController_SM_States (ENUM)
	E_MDL_RadiatorController_SM_States (ENUM)
	E_MDL_ThreeValveController_SM_States (ENUM)
	E_SimState (ENUM)
	POUs | Data types | Visualizations | Resources
	FB_HomeHeatingSim (FB-ST)
	E_MDL_BoilerController_SM_States
	`TYPE E_MDL_BoilerController_SM_Sta`
	`(`
	`    (****INITIAL STATES****)`
	`    MDL_BoilerController_SM_Initial,`
	`    (****NORMAL STATES****)`
	`    MDL_BoilerController_SM_NotWar`
	`    MDL_BoilerController_SM_Warmin`
	`);`
	`END_TYPE`

Running the PLC code is described in the reference implementation presentation. Please look there.