



Grant Agreement: 644047

Integrated Tool chain for model-based design of CPSs



SysML and FMI in INTO-CPS

Deliverable Number: D4.1c

Version: 0.7

Date: 2015

Public Document

Contributors:

Etienne Brosse, ST

Imran Quadri, ST

Editors:

Etienne Brosse, ST

Reviewers:

Richard Payne, UNEW

Nuno Amálio, UY

Alie El-Din Mady, UTRC

Consortium:

Aarhus University	AU	Newcastle University	UNEW
University of York	UY	Linköping University	LIU
Verified Systems International GmbH	VSI	Controllab Products	CLP
ClearSy	CLE	TWT GmbH	TWT
Agro intelligence	AI	United Technologies	UTRC
Softeam	ST		

Document History

Ver.	Date	Author	Description
0.1	22/05/2015	Etienne Brosse	Initial document version
0.2	1/11/2015	Imran Quadri	Initial document version- 2 nd draft
0.3	5/11/2015	Etienne Brosse	Add Mapping section and review
0.4	12/11/2015	Etienne Brosse	Proof review
0.5	12/11/2015	Imran Quadri	3 rd draft
0.6	24/11/15	Etienne Brosse	Proof review
0.7	29/11/15	Imran Quadri	Draft to address review comments

Abstract

This deliverable defines how INTO-CPS/SysML profile is utilized into the INTO-CPS platform at the end of the first year of the project. The deliverable describes FMI and its usage followed by how FMI is used in the context of INTO-CPS SysML profile along with the mapping between INTO-CPS/SysML profile and FMI. Finally, the document indicates the FMU concepts are modelled using INTO-CPS profile.

Contents

1	Introduction.....	6
2	FMI.....	6
2.1	FMI for Model-Exchange and Co-Simulation.....	7
3	INTO-CPS Model-Based Approach.....	9
3.1	UML and SysML	9
3.2	Modelio Modelling Environment	10
4	FMI and SysML for INTO-CPS.....	10
4.1	FMI ModelDescription.xml	10
4.2	INTO-CPS SysML to FMI Mapping.....	12
4.3	INTO-CPS SysML Modelling	15
4.3.1	INTO-CPS Architecture Structure Diagram (ASD)	15
4.3.2	INTO-CPS Connection Diagram (CD).....	16
5	Conclusion	17
6	References.....	18

1 Introduction

The INTO-CPS project focuses on the “model-based design” of Cyber-Physical Systems (CPSs). A model signifies a representation of some reality or systems with an accepted level of abstraction, i.e., all unnecessary details of the system are omitted for the sake of simplicity, formality, comprehensibility, etc. In INTO-CPS, the emphasis is on the utilisation of heterogeneous modelling, as the overall model in INTO-CPS is actually a multi-model, where each model can be described using a different modelling notation/language, possibly from a different computational paradigm or domain. In INTO-CPS models are categorized into continuous time (CT) and discrete event models (DE) as defined in D3.1a [1].

In the INTO-CPS multi-model approach, both holistic and design architecture approaches are advocated. A holistic architecture describes a conceptual model that highlights the main elements of the system and the way these elements are connected with each other, taking a holistic view of the overall system. The aim is to identify the main functional elements of the system reflecting the semantics and structure of the domain of application. The design architecture emphasises a decomposition of a system into subsystems, where each subsystem is modelled separately in isolation using a special notation and tool designed for the domain of the subsystem. The INTO-CPS SysML profile is designed to enable the specification of CPS design architectures [2].

The INTO-CPS Toolchain is a collection of software tools, based centrally around Functional Mockup Interface (FMI)-compatible co-simulation that supports the collaborative development of CPSs. The INTO-CPS Application interfaces tools in the INTO-CPS toolchain to query model information which can be used in collaborative simulations (co-simulations) by the INTO-CPS Co-simulation Orchestration Engine (COE) [2]. Central to the INTO-CPS Tool Chain is the use of the Functional Mockup Interface (FMI) standard [3].

Co-simulation is the simultaneous, collaborative, execution of multi-models and allowing information to be shared between them. The multi-model may be CT-only, DE-only or a combination of both. The Co-simulation Orchestration Engine (COE) combines existing co-simulation solutions and scales them to the CPS level, allowing CPS multi-models to be evaluated through co-simulation.

The aim of this deliverable is to take as input the INTO-CPS SysML profile developed in D2.1 [2] for FMI co-modelling and provide details on how to translate the INTO-CPS SysML models containing CPS components to be automatically transformed into FMI, and vice versa.

2 FMI

The Functional Mockup Interface (FMI) is a tool-independent standard to support both model exchange and co-simulation of dynamic models using a combination of XML-files and compiled C-code [4]. Part of the FMI standard for model exchange specification is a model description file; an XML file that supplies a description of all properties of a model (for example input/output variables). A Functional Mockup Unit (FMU) implements FMI. Data exchange between FMUs and the synchronisation of all simulation solvers [4] is controlled by a Master Algorithm (MA).

An FMU is in fact the executable that implements the interface. During FMU export an FMU archive is generated from a systems model, while during FMU import a systems model is generated from an FMU archive.

An FMU contains the following information:

1. A **Model Description** XML file: The model description file that contains information about the CPS model, for example variable definitions, attributes—and other more general model information, such as model name and FMI version. The advantages are:
 - a. There is no overhead for model execution; and
 - b. Tools independency: Tools can read this information with their preferred language (for example C++, Java etc.).
2. **Model equations**: A CPS model can be described using equations (differential, algebraic or discrete equations). These equations are represented by a small set of C functions. The source code and/or binary code for one or more platforms (such as Windows/Linux) is present in the FMU. One FMU can contain binaries for more than one platform and/or platform version.
3. **Optional resource files**: The FMU can contain other optional files that might be useful for the model, such as documentation files (HTML), model icon (bitmap file), maps and tables (read by model during initialization), and other libraries or DLLs that need to be used in the model.

2.1 FMI for Model-Exchange and Co-Simulation

The FMI standards currently specify two types of protocols:

1. FMI for Model Exchange (import and export), and
2. FMI for Co-Simulation (master and slave).

For FMI Model Exchange Import, the subsystem model is exported from a simulation tool in the form of an FMU archive containing the necessary FMU information (model description file, optional C source code, etc.); while in the FMI Model Exchange Export, the subsystem model is imported into the simulation system for system simulation. See Figure 1 and Figure 2 for an illustration of the FMI Model Exchange.

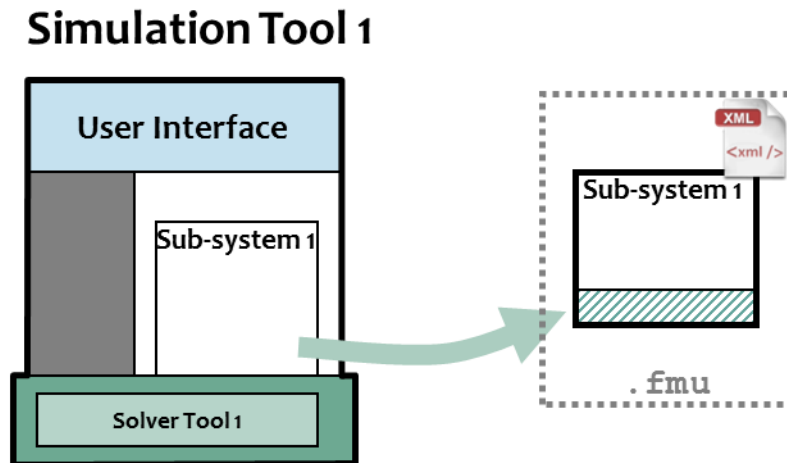


Figure 1: FMI Model Exchange Import

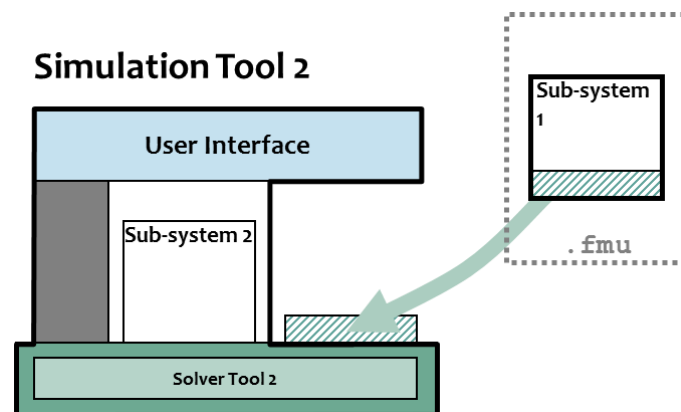


Figure 2: FMI Model Exchange Export

The FMI for Co-Simulation is to couple two or more simulation tools in a co-simulation environment, such as the INTO-CPS COE [2].

The main difference between these two protocols is that in Model Exchange the FMU is simulated using the importing tool's solver, while in Co-Simulation the FMU is shipped with its own solver.

It should be noted that FMI for Model Exchange allows several FMUs to be connected together. In both Model Exchange and Co-Simulation the FMUs can potentially be very large, containing tens of thousands of variables.

3 INTO-CPS Model-Based Approach

3.1 UML and SysML

In order to be interpretable by a machine, the expression, with which a model is represented, is pre-defined. This is achieved by a **metamodel**. A metamodel is a collection of concepts and relations for describing a model using a model description language; and is used for defining the syntax of a model.

Each model that is designed according to a given metamodel is said to conform to its metamodel at a higher level. This relation is analogous to a text and its language grammar. Here level does not signify an abstraction level, but a definition level. A metamodel itself is also a model, thus it also conforms to another metamodel. However, in order to define a model, it is not convenient to define an infinite succession of metamodels, with each one conforming to another at a higher level. One formal solution to this issue is the definition of a metamodel, which conforms to itself, i.e., it can be expressed only by using the concepts it defines. Currently, widely used metamodels, such as Ecore [5] and MOF [6], are examples of such kind of metamodels or metametamodels.

UML is considered as one of the main unified visual modelling languages in Model-Based Design [7]. The UML metamodel was standardized in 1997 by the Object Management Group (OMG). Since its standardization, UML has been widely accepted and adopted in industry and academia. UML now provides support for a wide variety of modelling domains, including real-time system modelling. It has been proposed to answer the requirements of modelling specification, communication, documentation, etc. It integrates the advantages of component re-use, unified modelling of heterogeneous system, different facet modelling of a system, etc. The proposition of UML is based on several languages, such as Booch and OOSE, which had a great influence on the object-based modelling approach. Consequently, UML is very similar to object-based languages. As UML is widely utilized in industry and academia for modelling purposes, a large number of tools have been developed for its support.

Unfortunately, the success of UML has its drawbacks, resulting in a bloated and complex language. Its expressivity and precision are not always well defined in certain cases for the specification of some specific systems; such as CPSs.

System Modeling Language (SYSML) [8] is the first UML standard for system engineering proposed by OMG that aims at describing complex systems. SysML allows describing of the traceability requirements, and provides means to express the behaviour and composition of the system blocks.

The key contributions of the SysML standard are as follows:

- **Architecture organization:** The modelling concepts related to expressing architectural aspects.
- **Blocks and Flows:** Blocks allow representing complex systems in a composed manner. Flows in SysML enable modelling of data/control flow as well as physical flows, such as electrical flows.
- **Behaviour:** SysML refines the UML common behaviour concepts (such as state machines, activities among others) for modelling continuous systems.
- **Requirements:** System requirements can also be modelled via SysML. These requirements can be presented either in graphical or tabular form and help with model traceability.

- Parametrics: SysML allows designers to describe analytical relations and constraints in a graphical manner.

The INTO-CPS SysML profile, defined in [2], proposes the following structuring of a CPS: Systems are decomposed into ‘subsystems’. These subsystems may be composed of cyber or physical components. Using the INTO-CPS tool chain, each subsystem corresponds to a FMI model description, and therefore an individual model in a multi-model.

3.2 Modelio Modelling Environment

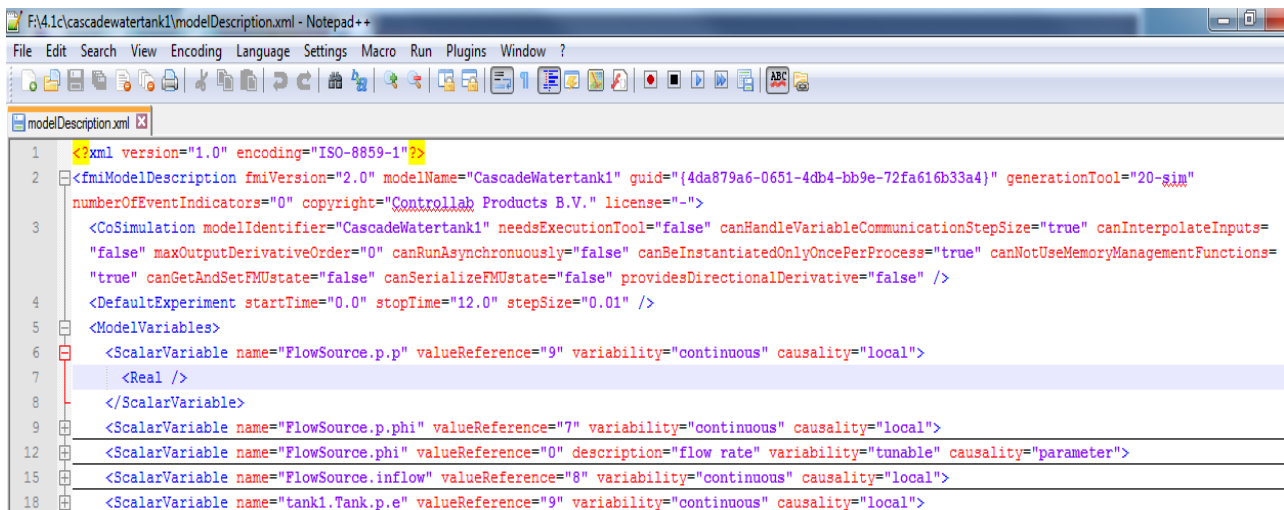
Modelio is an open source modelling tool [9] supporting UML standard and its extensions, such as SysML. Designers are thus able to simultaneously depict and specify several aspects of the system from requirements to the hardware/software architecture through use case specification, and system functional design. Modelio also enables transversal features such as automatic documentation generation and traceability/impact analysis which are helpful during system life cycle management. Customisation mechanisms provided by Modelio allow the definition of dedicated working environments. Additionally, Modelio supports the notion of World Wide modelling, which enables model fragments to be distributed and reused worldwide.

This deliverable makes use of the Modelio environment to a) define and develop the SysML profile and b) carry out the automatic transformation of SysML models into FMI, and vice versa.

4 FMI and SysML for INTO-CPS

4.1 FMI ModelDescription.xml

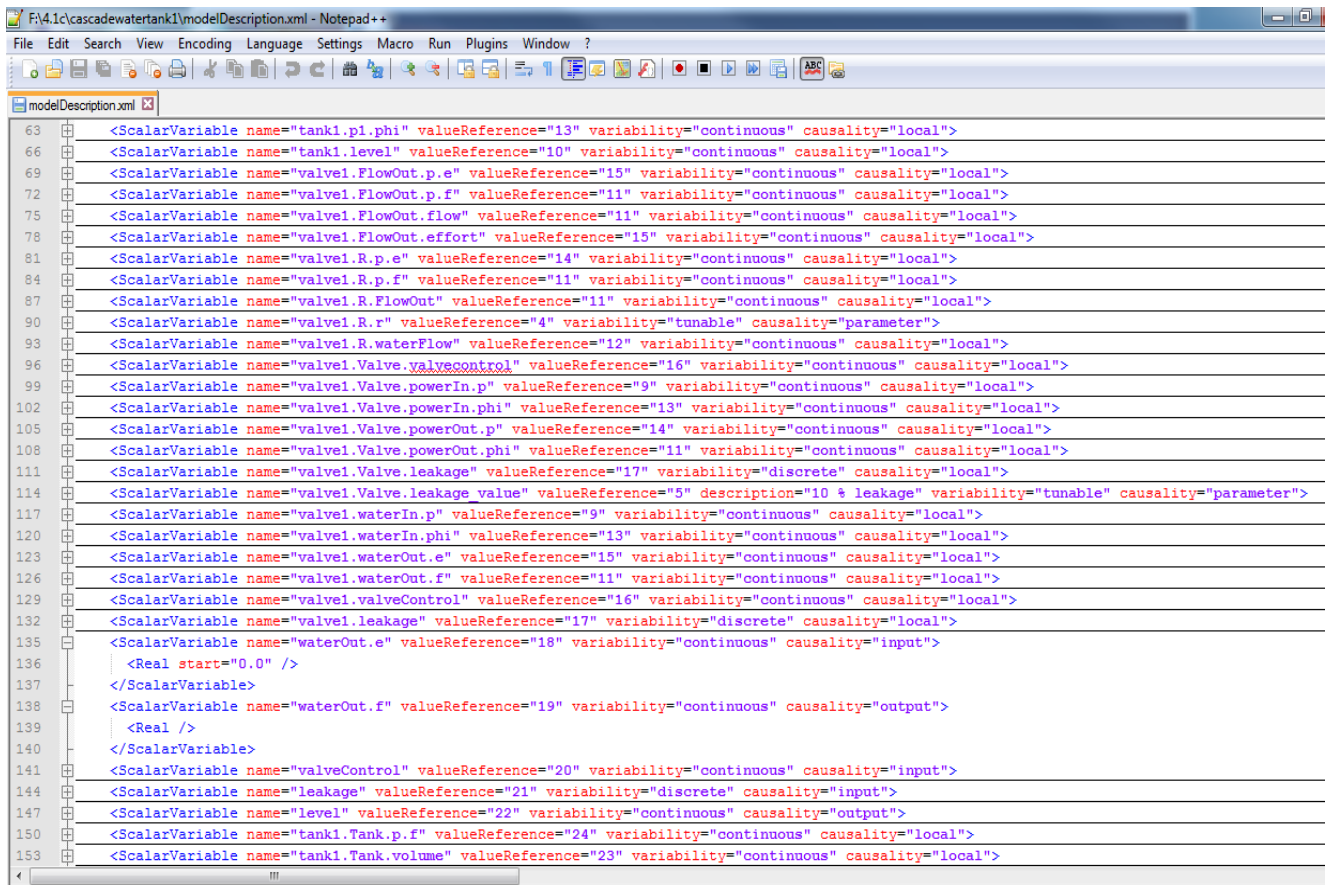
The following section provides an example of a *ModelDescription.xml* file related to cascading water tank developed in 20-sim [14] and illustrates the FMI syntax.



```

1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <fmiModelDescription fmiVersion="2.0" modelName="CascadeWatertank1" guid="{4da879a6-0651-4db4-bb9e-72fa616b33a4}" generationTool="20-sim"
   numberOfEventIndicators="0" copyright="Controllab Products B.V." license="-">
3    <CoSimulation modelIdentifier="CascadeWatertank1" needsExecutionTool="false" canHandleVariableCommunicationStepSize="true" canInterpolateInputs=
   "false" maxOutputDerivativeOrder="0" canRunAsynchronously="false" canBeInstantiatedOnlyOncePerProcess="true" canNotUseMemoryManagementFunctions=
   "true" canGetAndSetFMUstate="false" canSerializeFMUstate="false" providesDirectionalDerivative="false" />
4    <DefaultExperiment startTime="0.0" stopTime="12.0" stepSize="0.01" />
5    <ModelVariables>
6      <ScalarVariable name="FlowSource.p.p" valueReference="9" variability="continuous" causality="local">
7        <Real />
8      </ScalarVariable>
9      <ScalarVariable name="FlowSource.p.phi" valueReference="7" variability="continuous" causality="local">
10     </ScalarVariable>
12     <ScalarVariable name="FlowSource.phi" valueReference="0" description="flow rate" variability="tunable" causality="parameter">
13     </ScalarVariable>
15     <ScalarVariable name="FlowSource.inflow" valueReference="8" variability="continuous" causality="local">
16     </ScalarVariable>
18     <ScalarVariable name="tank1.Tank.p.e" valueReference="9" variability="continuous" causality="local">

```



```

63 <ScalarVariable name="tank1.pl.phi" valueReference="13" variability="continuous" causality="local">
66 <ScalarVariable name="tank1.level" valueReference="10" variability="continuous" causality="local">
69 <ScalarVariable name="valve.FlowOut.p.e" valueReference="15" variability="continuous" causality="local">
72 <ScalarVariable name="valve.FlowOut.p.f" valueReference="11" variability="continuous" causality="local">
75 <ScalarVariable name="valve.FlowOut.flow" valueReference="11" variability="continuous" causality="local">
78 <ScalarVariable name="valve.FlowOut.effort" valueReference="15" variability="continuous" causality="local">
81 <ScalarVariable name="valve.R.p.e" valueReference="14" variability="continuous" causality="local">
84 <ScalarVariable name="valve.R.p.f" valueReference="11" variability="continuous" causality="local">
87 <ScalarVariable name="valve.R.FlowOut" valueReference="11" variability="continuous" causality="local">
90 <ScalarVariable name="valve.R.r" valueReference="4" variability="tunable" causality="parameter">
93 <ScalarVariable name="valve.R.waterFlow" valueReference="12" variability="continuous" causality="local">
96 <ScalarVariable name="valve.Valve.valveControl" valueReference="16" variability="continuous" causality="local">
99 <ScalarVariable name="valve.Valve.powerIn.p" valueReference="9" variability="continuous" causality="local">
102 <ScalarVariable name="valve.Valve.powerIn.phi" valueReference="13" variability="continuous" causality="local">
105 <ScalarVariable name="valve.Valve.powerOut.p" valueReference="14" variability="continuous" causality="local">
108 <ScalarVariable name="valve.Valve.powerOut.phi" valueReference="11" variability="continuous" causality="local">
111 <ScalarVariable name="valve.Valve.leakage" valueReference="17" variability="discrete" causality="local">
114 <ScalarVariable name="valve.Valve.leakage.value" valueReference="5" description="10 % leakage" variability="tunable" causality="parameter">
117 <ScalarVariable name="valve.waterIn.p" valueReference="9" variability="continuous" causality="local">
120 <ScalarVariable name="valve.waterIn.phi" valueReference="13" variability="continuous" causality="local">
123 <ScalarVariable name="valve.waterOut.e" valueReference="15" variability="continuous" causality="local">
126 <ScalarVariable name="valve.waterOut.f" valueReference="11" variability="continuous" causality="local">
129 <ScalarVariable name="valve.valveControl" valueReference="16" variability="continuous" causality="local">
132 <ScalarVariable name="valve.leakage" valueReference="17" variability="discrete" causality="local">
135 <ScalarVariable name="waterOut.e" valueReference="18" variability="continuous" causality="input">
136 | <Real start="0.0" />
137 </ScalarVariable>
138 <ScalarVariable name="waterOut.f" valueReference="19" variability="continuous" causality="output">
139 | <Real />
140 </ScalarVariable>
141 <ScalarVariable name="valveControl" valueReference="20" variability="continuous" causality="input">
144 <ScalarVariable name="leakage" valueReference="21" variability="discrete" causality="input">
147 <ScalarVariable name="level" valueReference="22" variability="continuous" causality="output">
150 <ScalarVariable name="tank1.Tank.p.f" valueReference="24" variability="continuous" causality="local">
153 <ScalarVariable name="tank1.Tank.volume" valueReference="23" variability="continuous" causality="local">

```

Figure 3: Example of ModelDescription.xml

The ModelDescription.xml contains the definition of the simulation interface including input and output variables as well as some internal variables for monitoring purposes. The interfaces are defined in the XML language.

The main part of the description is the model variables definition. The FMI limits the definition to scalar variables. Thus, all the variables should be flattened and listed as a collection of scalar variables. For example Arrays should be flattened to scalars.

Each scalar variable has:

- **name** – a unique identifier with respect to all other elements of the ModelDescription file;
- **valueReference** – a default starting value (can be for example: Real, Integer or Boolean);
- **description** – a textual description of the scalar variable (optional).
- **variability** – An enumeration that defines the time dependency of the variable, i.e. defines the time instants when a variable can change its value: Can be Discrete, Continuous, Tunable or Fixed
 - *Discrete*: Value is constant between external and internal events (i.e. time, state, steps defined implicitly in the FMU)
 - *Continuous*: no restriction on value changes [Only variable of type Real can be continuous].
 - *Tunable*: Value is constant between external events
 - *Fixed*: Fixed value after initialization

- **causality** – Enumeration that defines the causality of the variable: Can be Parameter, Local, Input or Output
 - *Input*: The variable value is provided from another model or slave.
 - *Output*: The variable value can be used by other model or slave.
 - *Local*: A local scalar variable is not to be used outside of the FMU scope. Its value is calculated from other variables
 - *Parameter*: A parameter value is not changed after initialization. The value remains constant during simulation and is not used in connections to other FMUs. When the causality is set as “parameter”, the variability must be set to either “tunable” or “fixed”.

In INTO-CPS we use SysML to define the composition of the simulations that are defined with FMI interfaces. In the subsequent section, we define the INTO-CPS SysML concepts to FMI 2.0 mapping with the intended goal to automatically generated FMI related to FMUs from an INTO-CPS SysML model and to also enable the reverse process, i.e. import an FMI related to a FMU into Modelio modelling environment in order to automatically generate a corresponding INTO-CPS SysML model.

4.2 INTO-CPS SysML to FMI Mapping

This section describes the mapping between the INTO-CPS SysML concepts and the Functional Mockup Interface (FMI) Model Exchange Version 2.0 aspects. This mapping helps to generate a ModelDescription.xml file from a model based on the INTO-CPS profile defined in [2].

Note: Please note that this is a preliminary mapping and will be enriched in future due to updates in the INTO-CPS profile.

The correspondence between the SysML elements of the INTO-CPS SysML profile and FMI elements is given below in Table 1.

Table 1: Mapping between SysML and FMI Elements

INTO-CPS SysML Element	FMI Element
Block [with INTO-CPS “Component” stereotype] <ul style="list-style-type: none"> • A component can be either a physical, cyber or of sub-component type (itself consisting of physical or cyber components). • A component can have a type = it is either discrete or continuous • A component can have an associated platform = either Open Modelica, 20-Sim or VDM/RT 	FMU
FlowPort with “in” direction	Scalar input variable*
FlowPort with “out” direction	Scalar output variable*
SysML Interface with “StrtType” stereotype applied on a FlowPort with “in” direction	Scalar input variable
SysML Interface with “StrtType” stereotype applied on a FlowPort with “out” direction	Scalar output variable

Attribute	Scalar variable: can be local or parameter
Initial value of attribute or FlowPort	Starting referencing value of scalar variable

*In the INTP-CPS FMI to SysML mapping, 2 mechanisms are supported for expressing the input and output scalar variables.

```

<ScalarVariable name="waterOut.e" valueReference="18" variability="continuous" causality="input">
<ScalarVariable name="waterOut.f" valueReference="19" variability="continuous" causality="output">
<ScalarVariable name="valveControl" valueReference="20" variability="continuous" causality="input">
<ScalarVariable name="leakage" valueReference="21" variability="discrete" causality="input">
<ScalarVariable name="level" valueReference="22" variability="continuous" causality="output">

```

Figure 4: The input/output scalar variables of the Cascade Water Tank example

- 1) First, we support a mechanism whereas a SysML Flow Port is mapped to 1 single input or output scalar variable. This method is often natural for understanding the mapping between SysML and FMI. The issue regarding this approach is that it becomes problematic when a FMU corresponding to a SysML block has a large number of input/output scalar variables. This would also cause issues in connecting one FMU connections to the corresponding connections to another FMU. Thus an approach is needed to group variables and automate connections.

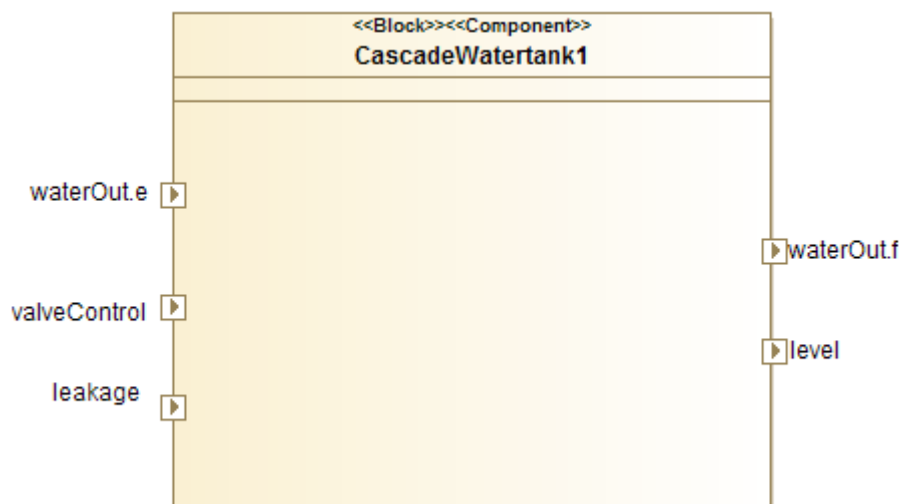


Figure 5: SysML modelling using the first approach

- 2) Second, to resolve the issues arising from the first approach, we would support a mechanism whereas a SysML Flow Port is typed by a SysML Interface with the “StrtType” stereotype, which in its turn includes variables that correspond to a set of either input or output scalar variables. This mechanism is very relevant in complex cases when the list of scalars is long and they should be grouped by their functional purpose. The input/output interfaces can have generic names such as Input Scalars / Output Scalars, containing the respectively named input/output scalar variables.

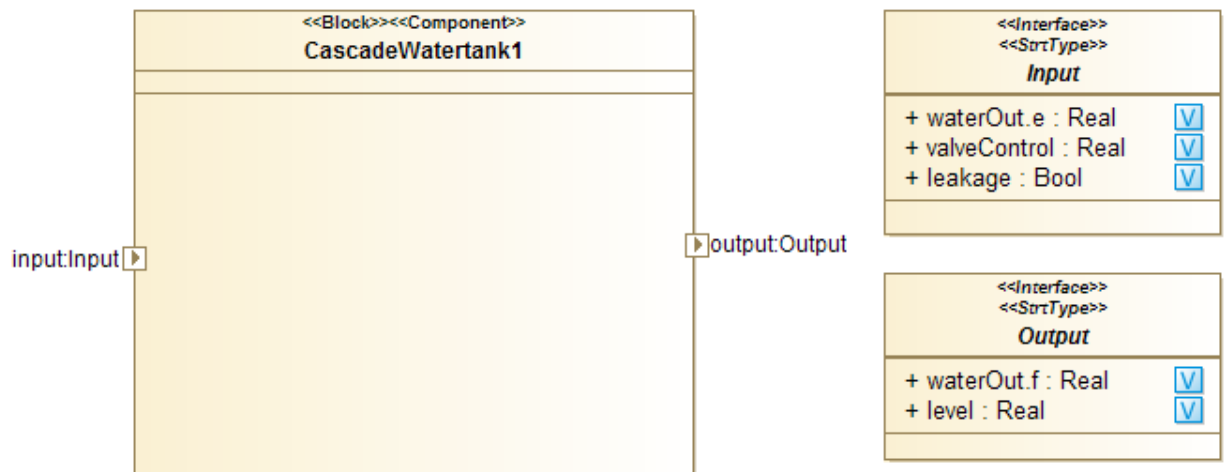


Figure 6: SysML modelling using the second approach

Henceforth we can define the mapping as follows:

- SysML Block: A SysML Block can correspond to the top level FMI element denoting that this is the highest hierarchical element in the model structure. This corresponds to a single FMU.
- FlowPort with 'in' direction: This corresponds to either one FMI Scalar input variable (can be either discrete or continuous) or a group of scalar input variables (via the second approach) when the FlowPort is typed with a “StrtType” interface.
- FlowPort with 'out' direction: This corresponds to either one FMI Scalar output variable (can be either discrete or continuous) or a group of scalar output variables (via the second approach) when the FlowPort is typed with a “StrtType” interface.
- Attribute: This corresponds to a FMI scalar parameter or local variable.
- Initial value of an attribute or FlowPort: This corresponds to the start value (value reference) of a scalar variable. Additionally, using the second approach, the attributes inside a “StrtType” interface can have initial values as well. Details related to StrtType (structural type) can be found in [2].

A SysML attribute pertaining to a FMU can have an initial value defined in the model, as seen in Figure 7. Here, the controller FMU has local variables, which have been set to some initial values.

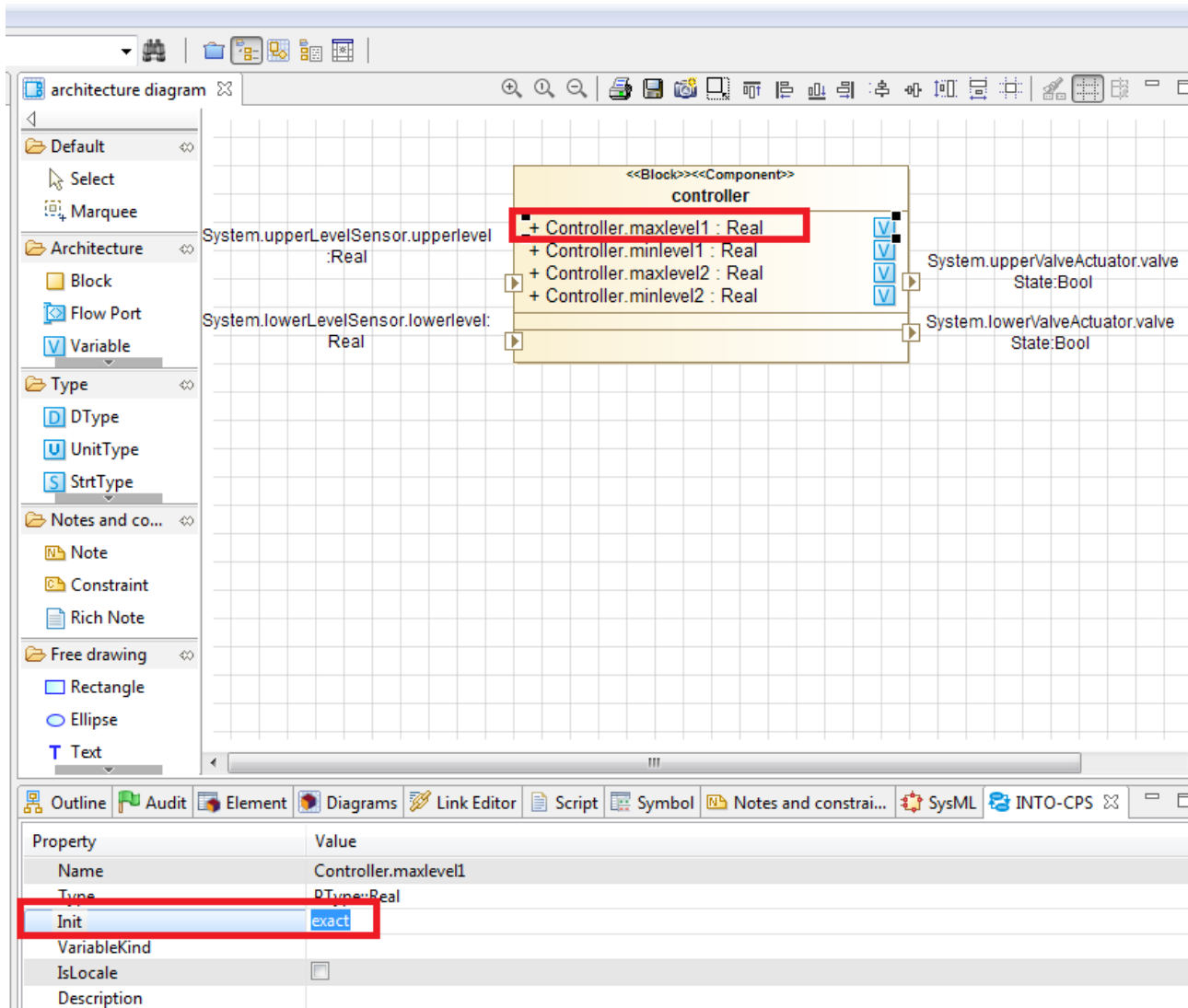


Figure 7: Setting initial values of FMU scalar variables

4.3 INTO-CPS SysML Modelling

Finally, in the context of the INTO-CPS SysML profile, two specific diagrams are used to model the system in question.

4.3.1 INTO-CPS Architecture Structure Diagram (ASD)

A SysML block diagram in the context of INTO-CPS is referred to as Architecture Structure Diagram (ASD), as defined in D2.1a [2].

An ASD is used to model SysML blocks, i.e. the external structure of the FMU along with its input/output ports and possible related interfaces, as presented in [15], and illustrated in Figure 6. An ASD can be specific to a single FMU in the overall design, and is henceforth usually used to model that single FMU in the form of a SysML block. Figure 8 and Figure 9 respectively show two FMUs belonging to a cascading

water tank example developed in 20-Sim. Figure 8 illustrates the SysML block corresponding to the ‘Controller’ FMU and its inputs/outputs; while Figure 9 subsequently displays one of the water tank FMUs that are connected to the controller FMU.

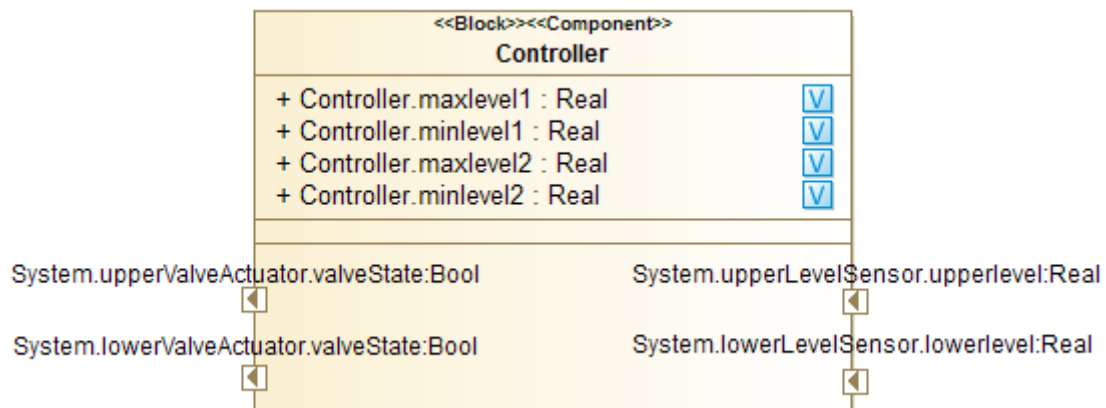


Figure 8: Example of INTO-CPS SysML Architecture Structure Diagram (Controller FMU)

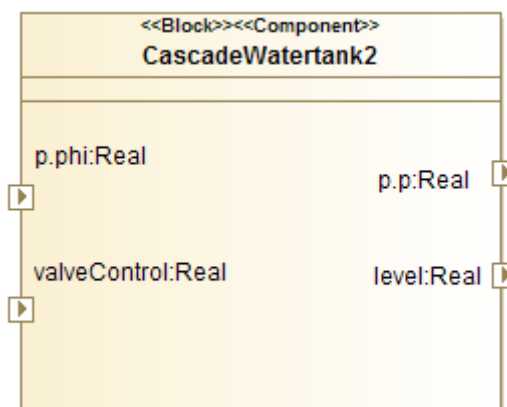


Figure 9: Example of INTO-CPS SysML Architecture Structure Diagram (Controller FMU)

4.3.2 INTO-CPS Connection Diagram (CD)

A SysML internal block diagram in the context of INTO-CPS is referred to as Connection Diagram (CD), as defined in D2.1a [2].

A CD is used to define the interactions between different FMUs. The CD contains instances of FMUs (SysML Block instances) which are connected together by means of connectors to the input/output ports of the FMU instances present in the CD. The FMU instances correspond to the FMUs modelled in the different ASDs, as seen in Figure 8 and Figure 9. The CD also enables to associate existing FMUs to the block instances for eventual simulation [15].

As seen in Figure 10, the CD illustrates the ‘instance’ that indicates the overall system containing FMU instances of ‘Controller’, ‘CascadeWatertank1’ and ‘CascadeWatertank2’; and the instances of these FMUs are connected with each other by means of connectors to their respective input/output ports.

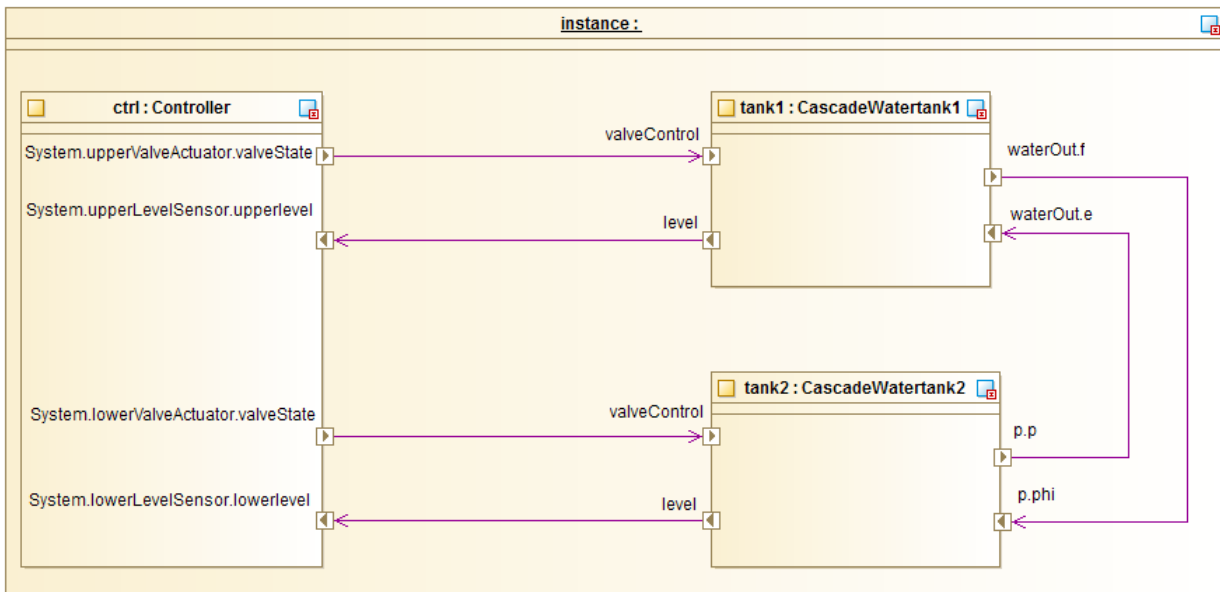


Figure 10: Example of an INTO-CPS SysML Connection Diagram (Cascading Water Tank Example)

5 Conclusion

This deliverable provides an overview of the current INTO-CPS SysML concepts and their mapping to FMI 2.0. The deliverable also provides an overview of the main FMI concepts used in the context of the INTO-CPS project. Finally, the deliverable illustrates how SysML is used in INTO-CPS, by highlighting the notion of Architecture Structure and Connection Diagrams developed in the context of INTO-CPS.

6 References

- [1] INTO-CPS. Deliverable 3.1a: Method Guidelines. 2015
- [2] INTO-CPS. Deliverable 2.1a: Foundations of the SysML Profile for CPS Modelling. 2015
- [3] FMI Standard. FMI 2.0. 2015. Available at: <https://www.fmi-standard.org/>
- [4] TorstenBlochwitz (Ed). Functional mock-up interface for model exchange and co-simulation. 2014. Available at: <https://www.fmi-standard.org/downloads>
- [5] Eclipse. Eclipse Modeling Framework. 2015. Available at: <http://www.eclipse.org/emf>.
- [6] Object Management Group Inc. MOF 2.0 core final adopted specification. 2003. Available at: <http://www.omg.org/cgi-bin/doc?ptc/03-10-04>
- [7] Object Management Group (OMG). Unified Modeling Language 2.4.1. Available: <http://www.omg.org/spec/UML>. 2011.
- [8] Object Management Group (OMG). System modeling language specification v1.3. Available: <http://www.omg.org/spec/SysML/1.3/>. 2012
- [9] Softeam. Modelio Open-Source Modelling Environment. <http://www.modelio.org/>. 2014
- [10] S. Sendall and W. Kozaczynski. Model Transformation: The Heart and Soul of Model-Driven Software Development. IEEE Software, 20(5):42–45, 2003
- [11] T.Mens and P. V. Gorp. A taxonomy of model transformation. In Proceedings of the International Workshop on Graph and Model Transformation (GraMoT 2005), volume 152 of Electronic Notes in Theoretical Computer Science, pages 125–142, March 2006.
- [12] Epsilon. Eclipse Epsilon Project. 2015. Available at: <http://www.eclipse.org/epsilon/download/>
- [13] Iulia Dragomir et al. Safety Contracts for Timed Reactive Components in SysML. SOFSEM 2014: Theory and Practice of Computer Science; Volume 8327. Lecture Notes in Computer Science pp 211-222
- [14] 20-Sim. 2015. Available at: <http://www.20sim.com/>
- [15] INTO-CPS. Deliverable D4.1a. User Manual for the INTO-CPS Tool Chain. 2015