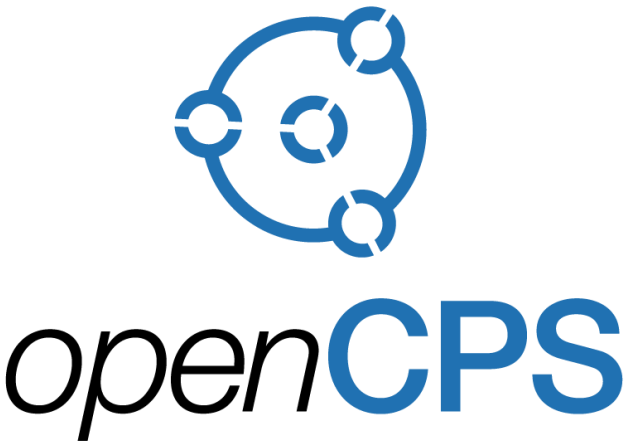
D4.1	Debugging state machines
Access[1]:	**PU**
Type[2]:	**Prototype**
Version:	**1.0**
Due Dates[3]:	**M36**



Open Cyber-Physical System Model-Driven Certified Development

Executive summary[4]:

D4.1 deliverable provides an overview of developments realized on debuggers for Modelica state machines and UML state machines. These supports for state machines debugging are respectively developed in OpenModelica and Papyrus Moka.

[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]	Jeremie Tatibouet	CEA	T4.1 leader	X
Contributing Author(s)[7]	Adeel Asghar	RISE SICS East AB	WP4 member	X
Internal Reviewer(s)[8]	Sébastien REVOL	CEA	WP2 leader	

Document History:

Version	Date	Reason for Change	Status[9]
0.1	15/11/2018	First Draft Version	Draft
0.2	16/11/2018	Added OpenModelica contribution	Draft
0.3	20/11/2018	Added CEA contribution	Draft
1.0	30/11/2018	Reviewed by CEA	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**
OMEdit	OpenModelica Connection Editor
PSSM	Precise Semantics for UML State Machines

1 INTRODUCTION

In the context of T4.1, debuggers are developed in both OpenModelica and Papyrus Moka [1]. These debuggers are intended to enable debugging of both Modelica state machines and UML state machines. The capability of debugging such behavioral models consists in enabling the user to suspend the execution of state machines, inspect contextual variables values, let the state machine execution to progress in a step by step mode, etc.

State machines specified and debugged in OpenModelica are different from those specified in Papyrus and debugged in Moka. Indeed, the language used to specify these two kinds of state machines are different and have different semantics.

1. Modelica state machines are similar to StateCharts [2]. They extend on synchronous language extension, supports hierarchic and parallel composition of states, immediate and delayed transitions. In the past, the library-based approach was used (e.g., StateGraph and StateGraph2 library) which requires the user to write the custom code so is not very convenient and powerful.
2. UML state machines derive from Harel state charts and ROOM [3] state machines. They execute according to a RTC (a.k.a, Run to Completion) semantics. The principle is simple: the state machine has an event pool, events occurrences available in that pool are dispatched in order, if an event occurrence can be accepted then it triggers a RTC step. Semantics of UML state machines are formalized and normative. They are fully specified in PSSM (Precise Semantics for UML State Machines) [4].

This is the final iteration of the deliverable. Section 2.1 provides and overview of the prototyping work on the OpenModelica side while section 2.2 focus on the developments realized on the Papyrus & Moka side. Finally, section 3 provides pointers to install both tools.

2 IMPLEMENTATION

2.1 OpenModelica

Support for Modelica state machines was added in the Modelica Language Specification v3.3. OMEdit uses the specification standard as a basis for the graphical support Figure 1. Modelica models can be defined as states as shown in the Figure 2.

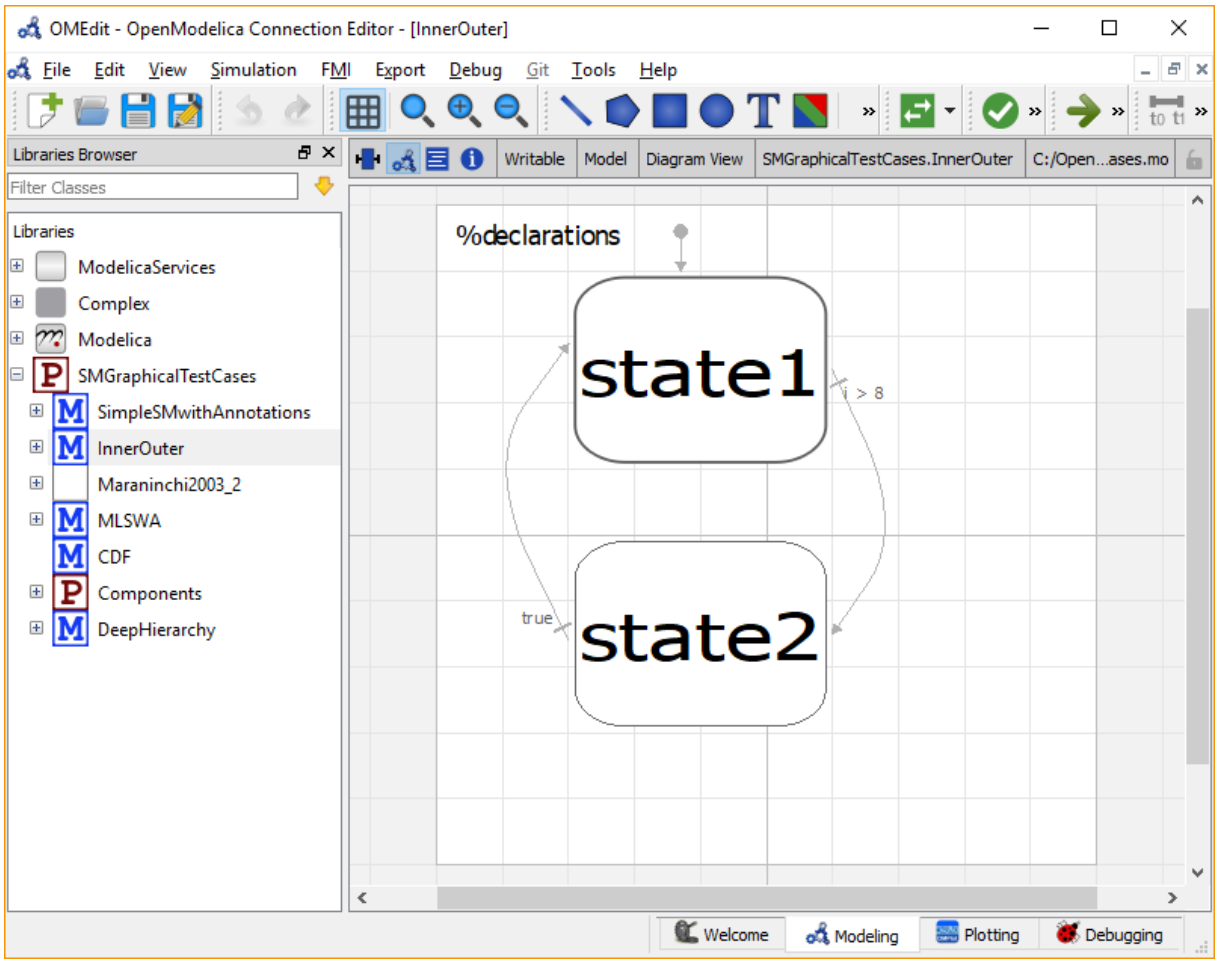


Figure 1. State machines in OMEdit.

2.1.1 Modeling

2.1.1.1 Creating a New Modelica State Class

Creating a new Modelica state class in OMEdit is rather straightforward. Choose any of the following methods,

- Select File > New Modelica Class from the menu.
- Click on New Modelica Class toolbar button.
- Click on the Create New Modelica Class button available at the left bottom of Welcome Perspective.
- Press Ctrl+N.

Additionally, make sure you check the State checkbox.

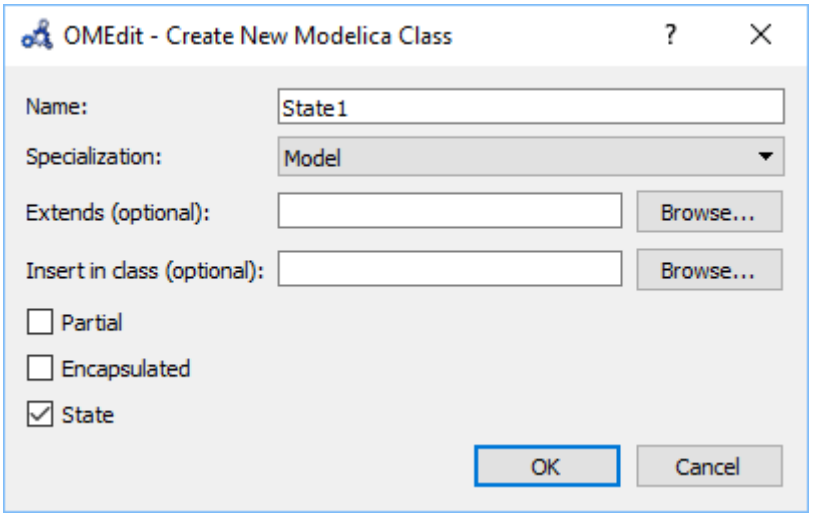


Figure 2. Creating a new Modelica state.

2.1.1.2 Transitions

In order to make a transition from one state to another the user first needs to enable the transition mode ( ) from the toolbar.

Move the mouse over the state. The mouse cursor will change from arrow cursor to cross cursor. To start the transition press left button and move while keeping the button pressed. Now release the left button. Move towards the end state and click when cursor changes to cross cursor.

A Create Transition dialog box will appear which allows you to set the transition attributes. Cancelling the dialog will cancel the transition.

Double click the transition or right click and choose Edit Transition to modify the transition attributes.

Model/Block instances are represented as states and allows to and from transitions. Each transition has “from” and “to” block instances and a condition that defines when to fire the transition. The “immediate” attribute defines whether it is strong or weak transition.

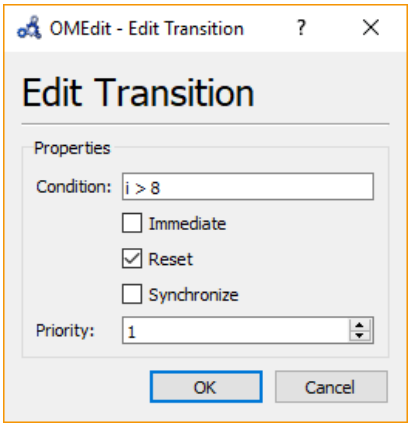


Figure 3. Transition attributes.

2.1.2 Debugger

Modelica state machines debugger is implemented as a visualization, which allows the user to run the state machines simulation as an animation Figure 4.

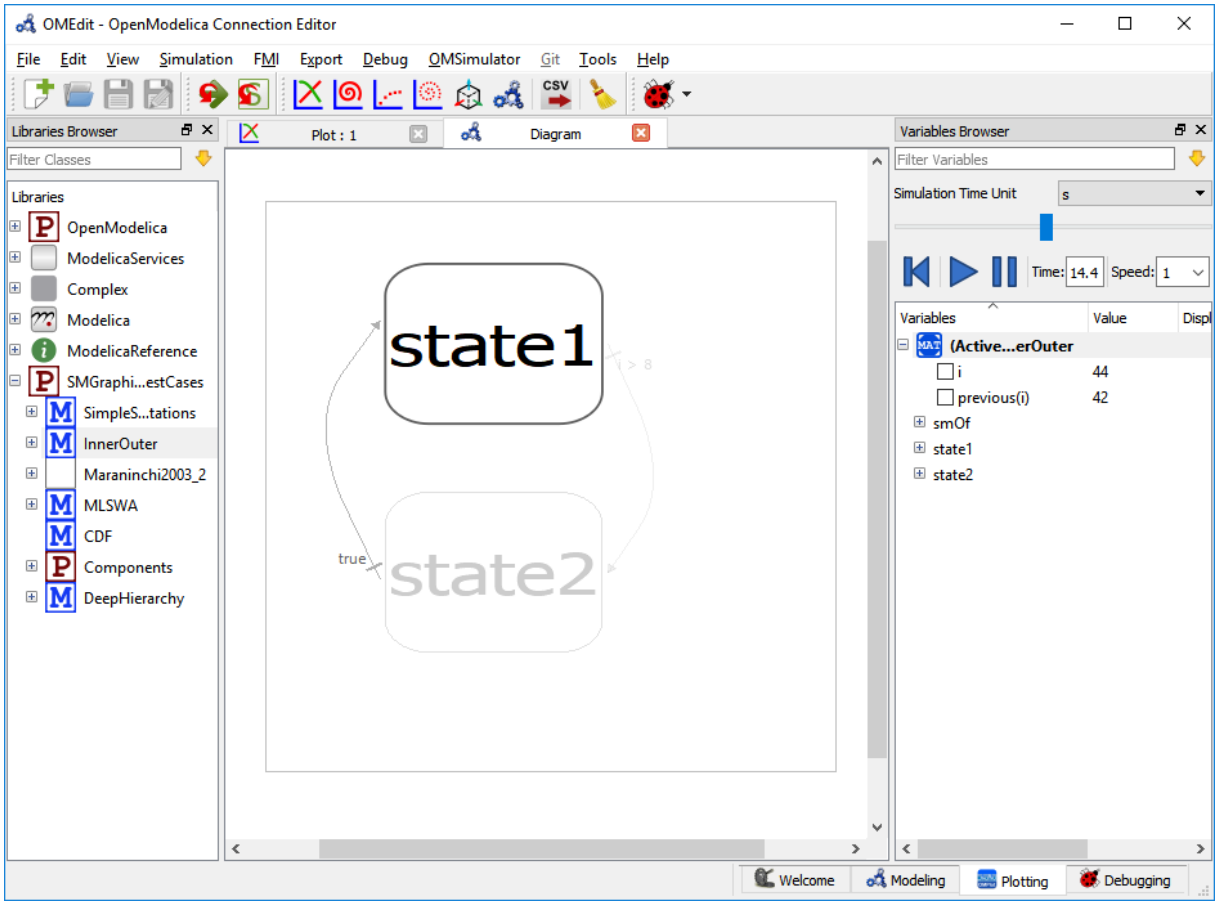


Figure 4. State machine debugger in OMEdit.

A special *Diagram Window* is developed to visualize the active and inactive states. The active and inactive value of the states are stored in the OpenModelica simulation result file [5]. After the successful simulation, of the state machine model, OMEdit reads the start, stop time values, and initializes the visualization controls accordingly.

The controls allows the easy manipulation of the visualization,

- Rewind – resets the visualization to start.
- Play – starts the visualization.
- Pause – pauses the visualization.
- Time – allows the user to jump at any specific time.
- Speed – speed of the visualization.
- Slider – controls the time.

The visualization is based on the simulation result file. All three formats of the simulation result file are supported i.e., mat, csv and plt where mat is a matlab file format, csv is a comma separated file and plt is an ordered text file.

It is only possible to debug one state machine at a time. This is achieved by marking the result file *active* in the *Variables Browser*. The visualization only read the values from the *active* result file. It is possible to simulate several state machine models. In that case, the user will see a list of result files in the *Variables Browser*. The user can switch between different result files by right clicking on the result file and selecting "***Set Active***" in the context menu.

2.2 Papyrus & Moka

In 2017, CEA put the focus on the development of a debug feature for UML state machines models conforming to PSSM. This year, CEA has focused on the consolidation of the tool dedicating to state machine modeling. In addition, CEA remained strongly involved at OMG to ensure the finalization of the PSSM specification. These two axis of work are presented in subclauses 2.2.1 and 2.2.2.

2.2.1 UML State Machines Modeling

The Papyrus team took advantage of this deliverable to fix a set of issues preventing the user from specifying valid state machines. Issues that have been fixed in the context of the state machine diagram are the following:

1. Bug 481499 – Prevent the capability to add regions within a final state
2. Bug 521260 – Prevent deletion of a transition when its kind is updated to internal
3. Bug 528502 – Disable "RemoveOrphanViewPolicy" in state machine diagram

In addition to the improvements made on the modeler side, CEA has focused on the update of the Precise Semantics of UML State Machines specification (a.k.a., PSSM). The provided updates are presented in subclause 2.2.2.

2.2.2 UML State Machines Semantics

PSSM is a specification of the standard semantics of a large subset of UML state machines. This specification is composed of the following artifacts:

1. The syntax model: a model of the syntactic subset for which semantics are defined
2. The semantic model: a model of the visitors capturing semantics of syntactic elements
3. The test suite model : a model describing a set of tests that a tool must be able to pass in order to claim to conform to the semantics defined in this specification
4. The specification document: a document presenting the syntactic subset, the semantic model as well the test suite.
   The PSSM initial version (i.e., 1.0b) was released in February 2017. Since this initial release, many feedbacks (e.g., issues in tests, need alignment with other executable specifications, etc.) were provided to the PSSM finalization task force (FTF). Based on these feedbacks, the FTF (co-chaired by CEA and Model Driven Solutions) submitted a finalized version of the specification. This version includes a large set of issue resolutions contributing to make the standard more robust. All issue resolutions submitted to update PSSM are presented in a report that is provided as an annex [6] of this deliverable (see file *ptc-18-11-01-PSSM_FTF_Report.pdf*). This report is going to be evaluated for acceptance by the OMG architecture board on December 10th, 2018.

3 AVAILABILITY OF THE PROTOTYPE

3.1 OpenModelica

The graphical support for state machines is part of OpenModelica since version 1.12.0, the debugging of state machines is a rather new development and is available through the nightly builds. The full functionality of state machines including the debugging will be part of the upcoming 1.13 release.

All releases can be downloaded through https://openmodelica.org.

3.2 Papyrus & Moka

The version of Papyrus and Moka including the very last issue resolution both for the modeling and semantics aspects are available at the following URLs:

1. https://hudson.eclipse.org/papyrus/job/Papyrus-Master/lastSuccessfulBuild/artifact/repository/
2. https://hudson.eclipse.org/papyrus/view/Moka/job/papyrus-moka-master/lastSuccessfulBuild/artifact/releng/org.eclipse.papyrus.moka.p2/target/repository/

REFERENCES

[1] CEA, «Papyrus Moka,» Online]. Available: https://wiki.eclipse.org/Papyrus/UserGuide/ModelExecution#Moka_Overview.

[2] D. Harel, «Statecharts: A visual formalism for complex systems,» *Science of Computer Programming*, vol. 8, pp. 231-274, 1987.

[3] B. Selic, «Tutorial: Real-TIme Object-Oriented Modeling (ROOM),» chez *Real-Time Technology and Applications Symposium*, Brookline, 1996.

[4] OMG, «Precise Semantics for UML State Machines,» OMG, 2016.

[5] B. Thiele, A. Pop et P. Fritzson, «Flattening of Modelica State Machines: A Practical Symbolic Representation,» chez *Proceedings of the 11th International Modelica Conference*, Versailles, France, 2015.

[6] OMG, «Report of the PSSM 1.0 FTF 2 Finalization Task Force to the OMG Platform Technical Committee,» 2018.

[7] S. A. Asghar et S. Tariq, «Design and Implementation of a User Friendly OpenModelica Graphical Connection Editor,» Linkoping University, 2010.