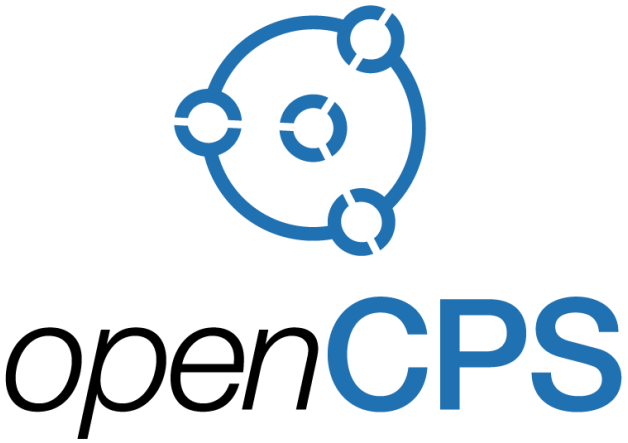
Annex 6 for D2.3	Scripting support for Papyrus
Access[1]:	**PU**
Type[2]:	**Prototype**
Version:	**2.0**
Due Dates[3]:	**M24, M36**



Open Cyber-Physical System Model-Driven Certified Development

Executive summary[4]:

Annex 6 for D2.3 focuses on the Python scripting support developed into Papyrus, allowing a tight integration of Papyrus SysML editor with OMSimulator, for configuring, and launching simulations, as well as analysing and displaying simulation results

[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]	Sebastien Revol	CEA	Task leader of T2.3	X
Contributing Author(s)[7]	David Lopez	CEA	Member of T2.3	
Internal Reviewer(s)[8]	Akos Horvath	IncQuery Labs	Member of T2.3	

Document History:

Version	Date	Reason for Change	Status[9]
0.1	30/11/2018	First Draft Version	Draft
1.0	03/12/2018	Final version	Final

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

CONTENTS

ABBREVIATIONS

List of abbreviations/acronyms used in document:

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

1 INTRODUCTION

Out of the box, Papyrus is a general purpose UML and SysML modeling tool. It provides all the diagram kinds defined by those standards and can support any regular system architecture specification methodologies based on them.
However, such methodologies are generally very descriptive, and connections to the lower level steps in the system design flow remain generally very low, leading to duplication of information. This duplication generally implies many additional efforts, such as rewriting, consistency management, and error introduction etc.
The goal of the activities described here is to try to reduce the gap between the system architecture domain and the system simulation domain, by providing consistent environments able to efficiently address both activities.
In that context, the documents D2.3-Annex5-SSP-Support-for-Papyrus.pdf describes how a complete FMU-based simulation architecture can be imported as or generated from a SysML internal block diagram. More particularly, the SSP standard allows to facilitate the switch between system architecture definition environment such as Papyrus, and numeric simulation environment such as OMEdit.
However, in some circumstances, it may be interesting to be able to run system simulations directly from the SysML architecture definition environment. It can help system architect to validate architectural choices or provide an integrated debugging/evaluation facility when building the models.
For that reason we decided to introduce the capability to directly integrate OMSimulator into Papyrus, thus allowing to configure, run and visualize results of simulations.
Moreover, in order to allow end-users to build their own "decision cockpit", we also integrated into Papyrus a dedicated python-based environment called Jupyter Notebook[10]. This tool, allows indeed to rapidly create simple web-based user interfaces to configure inputs and visualize simulation results. The tight integration of Papyrus and OMSimulator offers a complete integrated environment allowing to easily build interactive dashboards.

In the following sections, we are going to present the global architecture of the developed solutions, and illustrate on simple examples how python scripting and interactive Jupyter dashboards can be used.

[10] http://jupyter.org/

2 DEVELOPMENT DESCRIPTION

2.1 Overall architecture

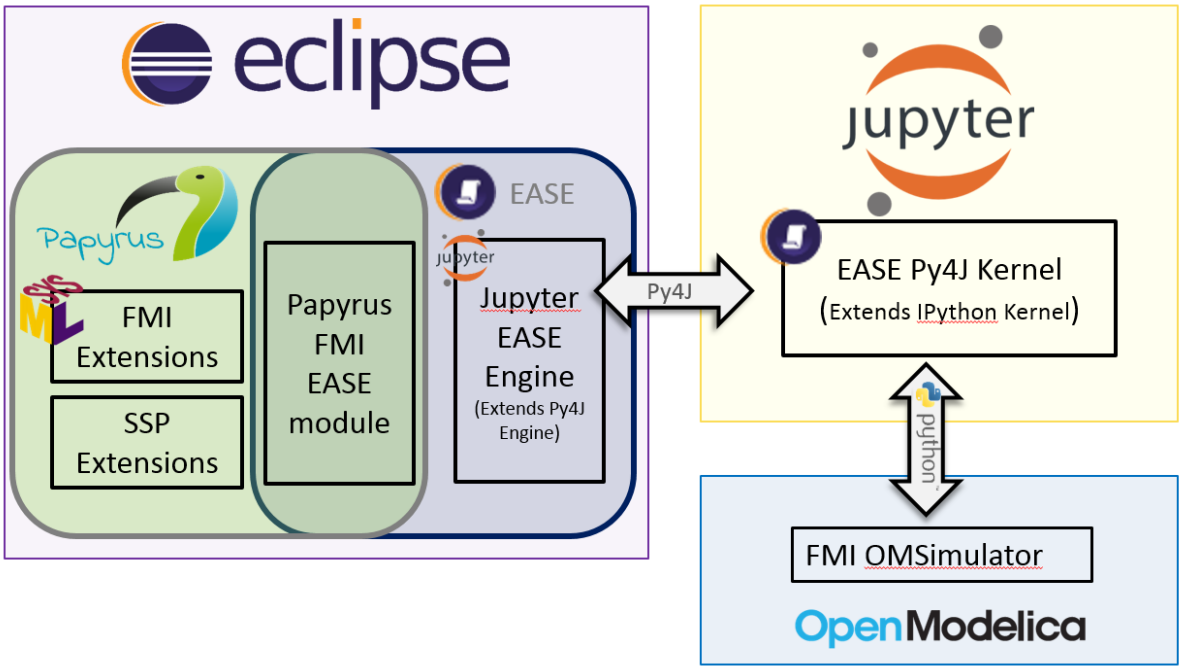


Figure 1 General Architecture of the presented solution

Figure 1 shows the general architecture of the developed solution. It is based on the integration of 3 main different tools :

- **Eclipse/Papyrus** : Eclipse[11] is an open source plugin based platform into which Papyrus[12] is integrated as a specific set of plugins. By default, Papyrus provides standard support of UML and SysML. On top of that, IncQuery Labs and CEA developed *FMI Extensions and SSP Extensions* allowing to transform Papyrus into a FMI/SSP graphical user interface. Moreover CEA also developed a *Papyrus FMI EASE Module* allowing read/write access to SysML models with FMI/SSP from scripting languages like Python or Javascript. Last, CEA also developed a specific *Jupyter EASE Engine* allowing to seamlessly interact with Jupyter Notebook.
- **Jupyter Notebook**, an open-source web application that allows to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. Thanks to our connection with Eclipse/Papyrus, it allows to have a complementary use where Papyrus is the graphical user interface for defining and visualizing simulation composite models, and Jupyter is as an interactive dashboard able to configure and launch simulations, as well as visualize and document simulation results. Jupyter is based on Python scripting language. We developed a specific *EASE Py4J Kernel* in Jupyter, equivalent to the *Jupyter EASE Engine* on Eclipse side.

[11] https://www.eclipse.org/

[12] https://www.eclipse.org/papyrus/

- **OMSimulator**, the main tool developed in OpenCPS. More particularly we access to the tool thanks to its Python API, allowing a very low level control of the simulations.

2.2 FMI and SSP extensions

FMI and SSP extensions are detailed in document D2.3-Annex5-SSP-Support-for-Papyrus.pdf. The main idea is to provide SysML extensions, thanks to the UML *Profile* mechanism to be able to import SSP files and to handle them in SysML Block Definition Diagram (BDD) and Internal Block Diagram (IBD).
Each imported FMU is indeed transformed into a SysML Block definition, whereas the composition of FMU instances, corresponding to SSD files, is realized in a new hierarchical block, whose internal structure is described with an Internal Block Diagram.

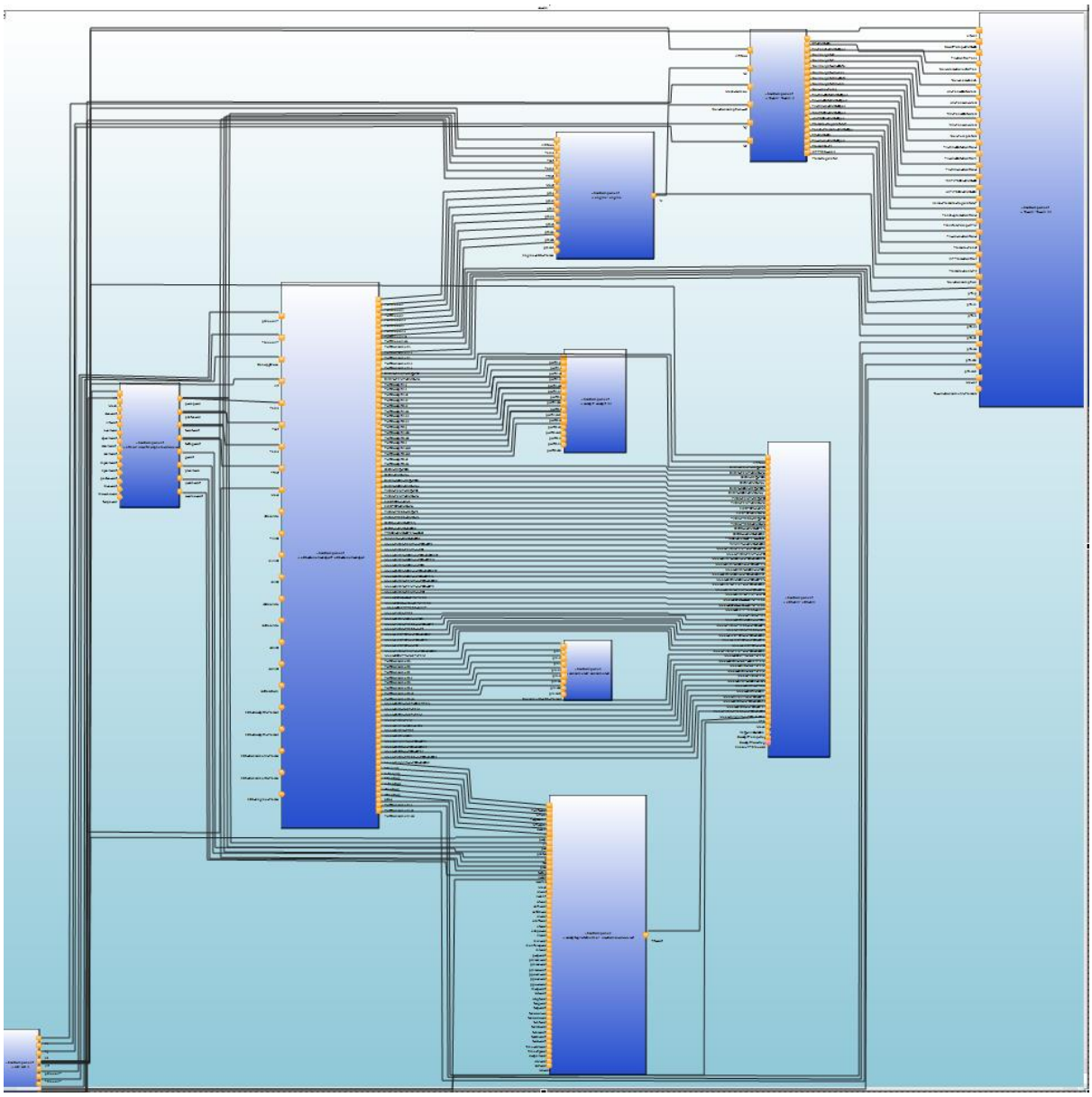


Figure 2 Saab Demonstrator imported into Papyrus

Figure 2 shows the result of the SSP import process of the complete T6.3 Saab Aircraft demonstrator.
The FMU composition was initially done with an OMSimulator Lua script, and then exported as a new SSP file. The file is then imported into Papyrus with the SSP import functionality.

Notice that the layout has been automatically generated thanks to another Eclipse open source extension named Eclipse Layout Kernel[13] (ELK), and more particularly with its Graphical Modeling Framework extension on which Papyrus is based.

2.3 Papyrus FMI EASE scripting module

Scripting proposes very complementary functionalities to Graphical User Interfaces such as Papyrus. Whereas a diagram allows to have a good overview of the complete architecture, or facilitates the assembly/analysis of simple systems, it can become very cumbersome when handling complex systems like described in Figure 2, needing to create hundreds of connections.
On top of Eclipse Advanced Scripting Environment[14] (EASE), we developed a communication bridge allowing to access to Papyrus in-memory models from scripting languages such as Python.
More particularly, we created a programming interface very similar to OMSimulator Python API allowing to create in Papyrus the composition of FMUs.
The scripts can be used to create a new composite model, or to modify/update an existing one. They can also allow the easy creation of model queries, to validate for instance if the models conforms to in-house modeling rules (naming rules, missing connections detection etc.)

```
# name     :Create Sherpa model
#popup : enableFor(org.eclipse.uml2.uml.Class)

loadModule("FMI Module")
loadModule("/System/UI")
include("platform:/plugin/org.eclipse.papyrus.moka.fmi.ease/script/fmi_util.py")

def createModel() :

    papyrusFile = "workspace://SherpaPapyrusTestCase/SherpaTestCase.di"
    openEditor(papyrusFile)
    papyrusSimulatorArchi = getSimulatorHandler(getPapyrusNamedElement(papyrusFile,"SherpaTestCase::Simulator"))

    papyrusSimulatorArchi.addConnection("Supervisor1:Vehicle_Cycle","ControlSystem1:Vehicle_Cycle")
    papyrusSimulatorArchi.addConnection("Supervisor1:Vehicle_Mode_Choice","ControlSystem1:Vehicle_Mode_Choice")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:TorqueBra_me","ControlSystem1:TorqueBra_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:PowerElecElecMac_me","ControlSystem1:PowerElecElecMac_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:PowerMechElecMac_me","ControlSystem1:PowerMechElecMac_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:AngularSpeedElecMac_me","ControlSystem1:AngularSpeedElecMac_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:TorqueElecMac_me","ControlSystem1:TorqueElecMac_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:UElecAux_me","ControlSystem1:UElecAux_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:PwrMaxDechargeBattery_me","ControlSystem1:PwrMaxDechargeBattery_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:SOCBattery_me","ControlSystem1:SOCBattery_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:EmaxBattery_me","ControlSystem1:EmaxBattery_me")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:VehSpeed_me","ControlSystem1:VehSpeed_me")
    papyrusSimulatorArchi.addConnection("ControlSystem1:TorqueBra_sp","OperatingSystem1:TorqueBra_sp")
    papyrusSimulatorArchi.addConnection("ControlSystem1:Battery_sp","OperatingSystem1:Battery_sp")
    papyrusSimulatorArchi.addConnection("ControlSystem1:IElecAux_sp","OperatingSystem1:IElecAux_sp")
    papyrusSimulatorArchi.addConnection("ControlSystem1:TorqueElecMac_sp","OperatingSystem1:TorqueElecMac_sp")
    papyrusSimulatorArchi.addConnection("OperatingSystem1:SOCBattery_me","Supervisor1:Battery_SOC")
papyRun(createModel)
```

Figure 3 FMU composition script

[13] https://www.eclipse.org/elk/
[14] https://www.eclipse.org/ease/

Figure 3 shows an example of FMU composition script. We can see in line 12 that Papyrus simulation architecture is accessed with a call to "*getSimulatorHandler*". A sequence of calls to *"addConnection"* allows then to create SysML connectors between two port instances.

Last we can notice the last line (line 31), that calls the script in Papyrus context with the *"papyrun"* function. In practice, this allows to execute the script within an *undo/redo* transaction, meaning that all the modifications in Papyrus in-memory model performed by the script can be easily undone if wrong, allowing a fast iterative development process of the scripts.

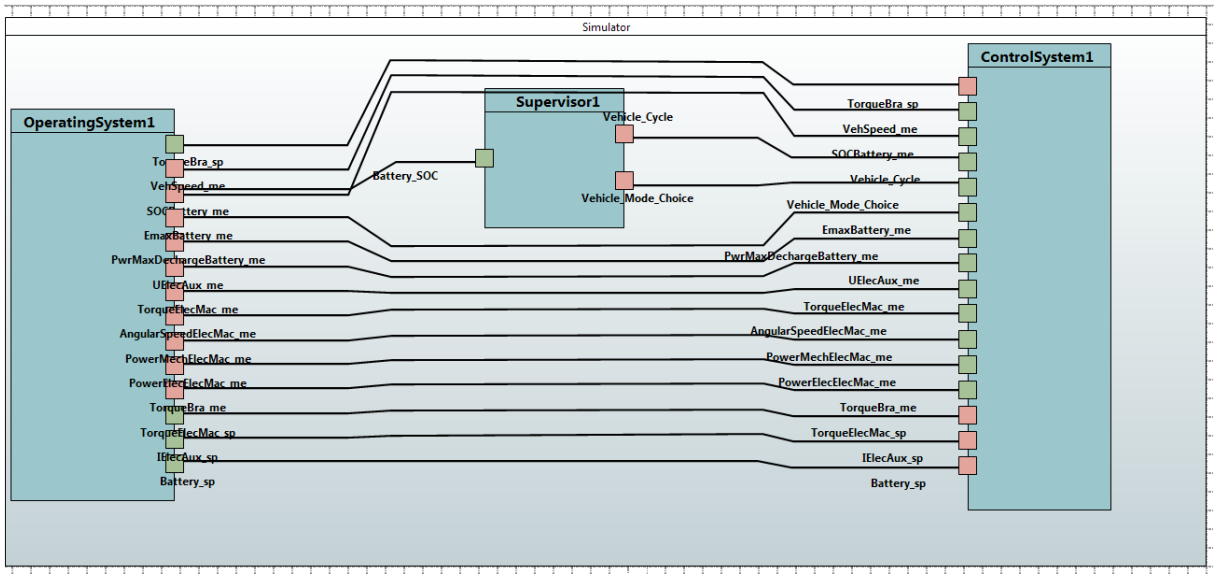


Figure 4 Result of script execution in Papyrus

Figure 4 shows the result of the script of Figure 3 after applying the ELK auto layout functionality.

Thanks to OMSimulator Python API, it also becomes very easy to create dedicated scripts allowing to run simulation within the same scripts.

Figure 5 shows an example of such script, executing the Sherpa test case directly from Papyrus. Lines 21 and 23 show the code to write to initialize OMSimulator, with the call the "*instantiateArchitecture"* function. This function is reading the current model in Papyrus memory, and creates the corresponding model in OMSimulator.
Then regular Python functions from OMSimulator API can be called, such as "*setCommunicationInterval"* (simulation time step setting) *"setResultFile"* or *"simulate"*[15].

[15] The complete python API is described in the following web page: https://openmodelica.org/doc/OMSimulator/html/OMSimulatorPython.html

```
1  # name     : Run Sherpa Simulation
2  #papypopup : enableFor(org.eclipse.uml2.uml.Class)
3
4  import os
5  import sys
6  import csv
7
8  loadModule("FMI Module")
9  loadModule("/System/Resources")
10
11 include("platform:/plugin/org.eclipse.papyrus.moka.fmi.ease/script/fmi_util.py")
12 import OMSimulator
13
14
15 project = getProject("SherpaPapyrusTestCase")
16 projectPath =    project.getRawLocation().makeAbsolute().toOSString()
17
18
19 papyrusSimulatorArchi = getSimulatorHandler(getSelection())
20 identifier = papyrusSimulatorArchi.getName()
21 omSimulator = OMSimulator.OMSimulator()
22
23 instantiateArchitecture(omSimulator, papyrusSimulatorArchi)
24
25 resultFile = os.path.join(projectPath,'simulationResult.mat')
26
27 #Simulation settings
28 omSimulator.setCommunicationInterval(identifier, 1e-3)
29 omSimulator.setResultFile(identifier, resultFile)
30 omSimulator.setStopTime(identifier, 5)
31
32
33 omSimulator.initialize(identifier)
34 omSimulator.simulate(identifier)
35
36 omSimulator.unloadModel(identifier)
37
```

Figure 5 Definition of an OMSimulator script in Papyrus

We can give a particular attention to lines 1 and 2. Those headers are indeed dynamically interpreted by EASE framework, allowing to directly launch the script from a corresponding contextual menu in Papyrus user interface. Line 1 defines the name of the menu "*Run Sherpa Simulation"*, whereas line 2 states that this script should be enabled only when an UML Class is selected in the editor.
The selected Class is then accessed in the script at line 19 thanks to the call to *"getSelection"*.

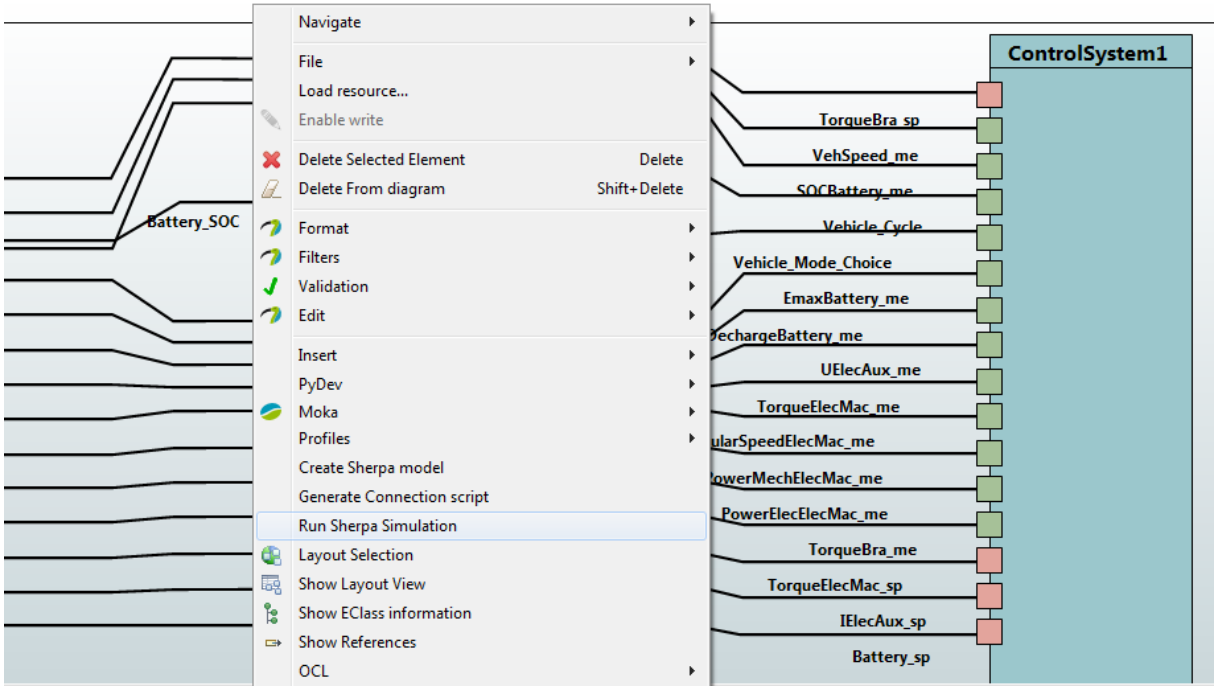


Figure 6 Dynamic menu creation in Papyrus user interface from scripts

2.4 Jupyter integration

As mentioned in section 2.1, Jupyter Notebook is an open-source web application allowing to create and share documents containing live code, equations, visualizations and narrative text. It's written in Python, and provides its own integration of a Python interpreter.
The challenge of our development was to enable the execution of the scripts described in the previous section, accessing both to Papyrus model and OMSimulator, with Jupyter Notebook python interpreter instead of the regular python interpreter provided by EASE.

We managed to do that by the definition of a new EASE scripting engine, the *Jupyter EASE Engine* mentioned in Figure 1. This engine is launching Jupyter Notebook from Eclipse/Papyrus with specific arguments allowing to execute our dedicated *EASE Py4J Kernel* in Jupyter. This new Kernel establishes a socket-based communication with Eclipse at Jupyter start-up, relying on *Py4J*[16] to make Jupyter Python interpreter and Eclipse Java virtual machine working together.

Jupyter notebook is launched in an external web browser that becomes tightly connected to the running eclipse session. It is then possible to pilot Eclipse GUI from this web browser, with commands such as "*openEditor*" allowing to open a Papyrus file and visualize its diagrams, for further access to its internal model elements.

Moreover, Jypter Notebook provides easy to use APIs to create simple graphical user interfaces in the web browser[17]. Thanks to this API, we developed simple examples of functions allowing to call OMSimulator, with graphical sliders enabling the setting of the simulation time step and the simulation end time, as well as the result file name, as shown in Figure 7.

[16] https://www.py4j.org/

[17] https://ipywidgets.readthedocs.io/en/stable/examples/Widget%20Basics.html

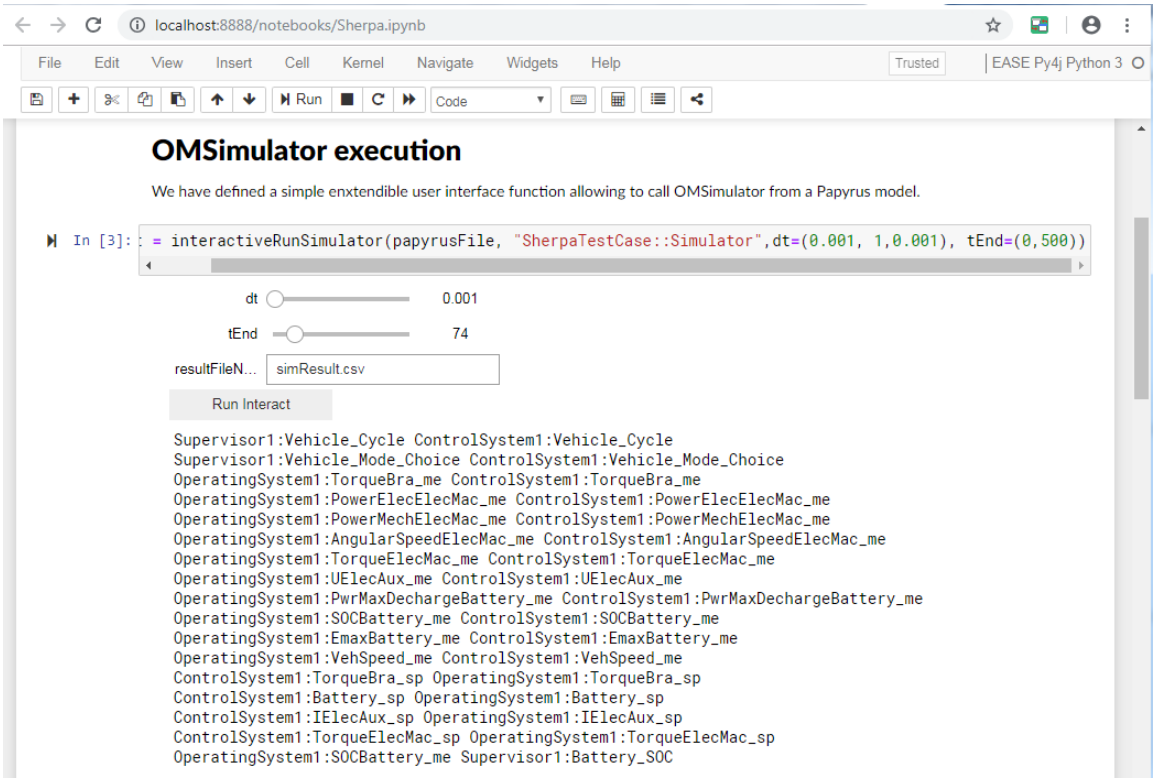


Figure 7 Example of simple user interface in Jupyter

Moreover, Jupyter provides a very good integration of the *matplotlib*[18] plotting and *pandas*[19] data analysis libraries, allowing to easily display and analyze simulation results in the notebook.

Hence, Figure 8 shows how the simulation results of Figure 7 are plot in a simple graph, and Figure 9 shows how a new column, computing the difference between the two previously displayed traces, can be created and displayed.
Last, Figure 10 shows how this new column is then analyzed, with a query selecting only the values where the absolute difference between the traces is greater than 0.6. The result is then displayed in an interactive table, providing additional formatting and sorting functionalities, enabling convenient and powerful data analysis.

[18] https://matplotlib.org/
[19] https://pandas.pydata.org/

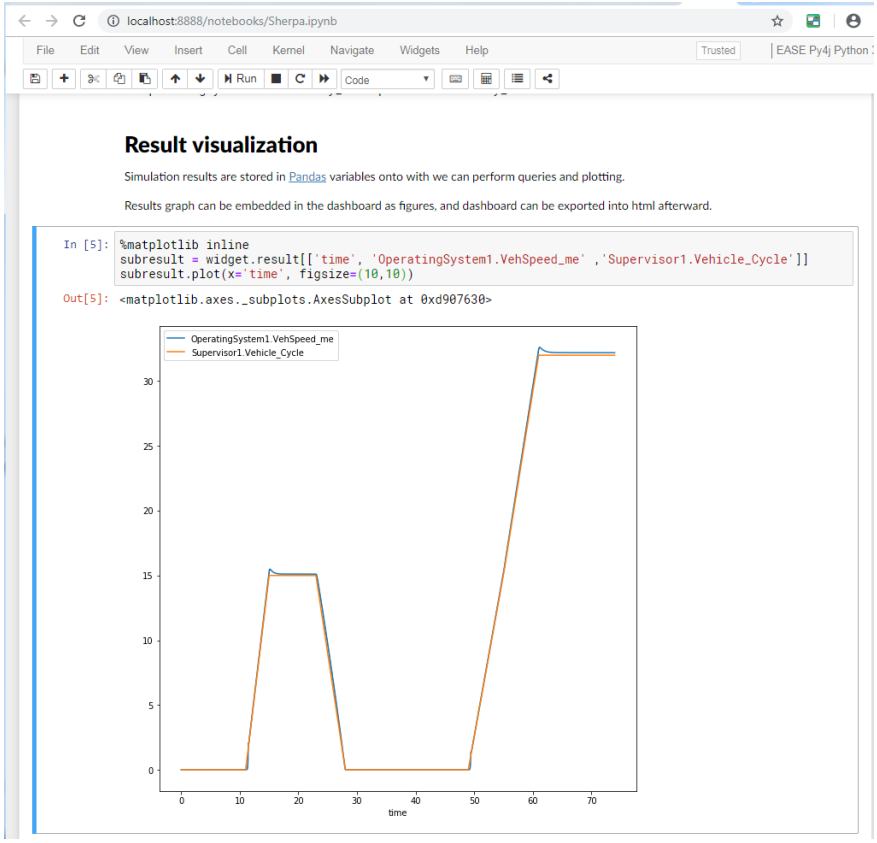


Figure 8 Plotting simulation results

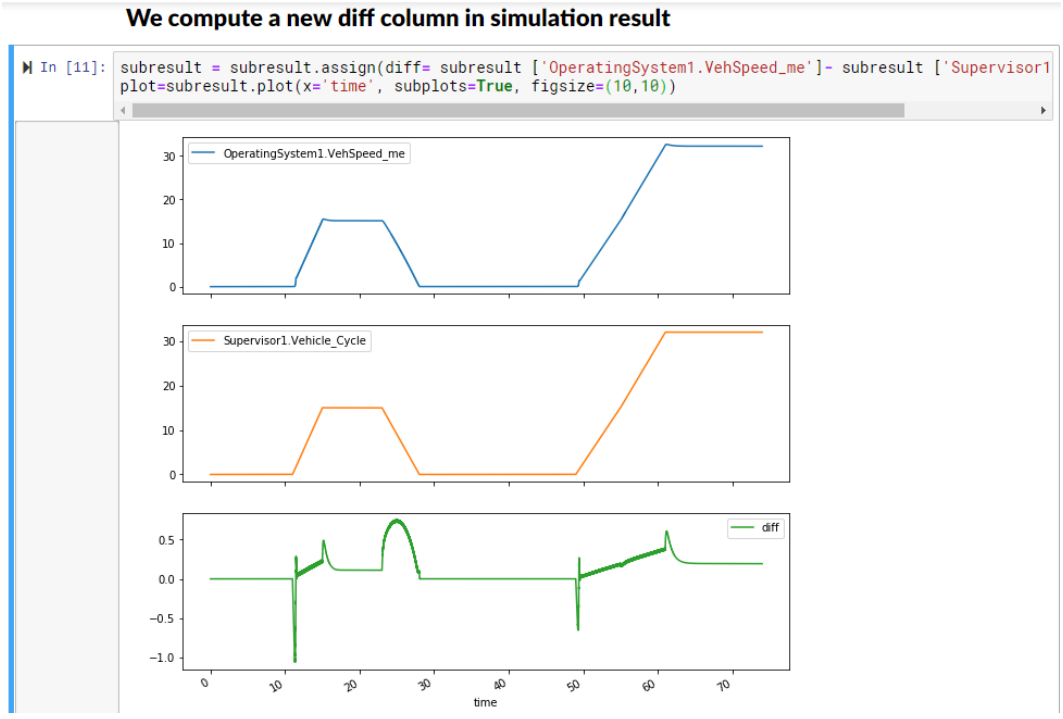


Figure 9 Creation of a new diff column in simulation results

```
In [10]: subresult[np.abs(subresult['diff'])>0.6]
```

index	time	OperatingSystem1.VehSpeed_me	Supervisor1.Vehicle_Cycle	diff
22325	11.163	0.009	0.611	-0.602
22326	11.163	0.009	0.611	-0.602
22327	11.164	0.009	0.615	-0.606
22328	11.164	0.009	0.615	-0.606
22329	11.165	0.009	0.619	-0.610
22330	11.165	0.009	0.619	-0.610
22331	11.166	0.009	0.623	-0.613
22332	11.166	0.009	0.623	-0.613
22333	11.167	0.009	0.626	-0.617
22334	11.167	0.009	0.626	-0.617
22335	11.168	0.009	0.630	-0.621
22336	11.168	0.009	0.630	-0.621
22337	11.169	0.009	0.634	-0.625
22338	11.169	0.009	0.634	-0.625
22339	11.170	0.011	0.637	-0.626
22340	11.170	0.011	0.637	-0.626
22341	11.171	0.011	0.641	-0.630
22342	11.171	0.011	0.641	-0.630
22343	11.172	0.011	0.645	-0.634
22344	11.172	0.011	0.645	-0.634
22345	11.173	0.011	0.649	-0.638
22346	11.173	0.011	0.649	-0.638
22347	11.174	0.011	0.652	-0.641
22348	11.174	0.011	0.652	-0.641
22349	11.175	0.011	0.656	-0.645

Figure 10 Query in simulation results, displayed in an interactive table

3 CONCLUSION

The development realized in OpenCPS allowed to transform Papyrus into an integrated simulation environment, providing scripting capabilities to interact with Papyrus models for FMU composition, execution of simulations, and result analysis with interactive dashboards. Those developments have been successfully tested on both Saab and Sherpa demonstrators. The functionalities, including both SSP extensions developed by IncQuery Labs and scripting extensions developed by CEA will be released soon as free open source features available on Eclipse platform repository at the following address:

- https://download.eclipse.org/modeling/mdt/papyrus/components/

They provide a strong base to further developments and investigations, more particularly in connection with the flows described by Saab and Sherpa in deliverable *D2.5 "Integration of the FMI standard"*. This document is proposing to establish tight links between System Architecture SysML models and lower level simulation models, aiming at increasing consistency and automation capabilities between those two activities.