# What is DACCOSIM?

1.1 DACCOSIM project contributors

1.2 DACCOSIM licensing mode

# Overview of DACCOSIM 2017

2.1 Main DACCOSIM features

2.2 DACCOSIM control architecture

# Installation procedure

3.1 Required Eclipse and Java distribution

3.2 Eclipse plug-ins installation

3.3 Required Visual C++ Redistributable Packages

3.4 Source codes installation

# Graphical User Interface of DACCOSIM 2017

4.1 Activating the GUI

4.2 Handling DACCOSIM projects

4.3 Designing calculation graphs

4.4 Defining calculation resources

---

4.4.2 Defining calculation resources

4.4.2.1 Creating a new cluster

4.4.2.2 Deleting an existing cluster

4.4.2.3 Modifying an existing cluster

4.4.2.4 Adding nodes to a cluster

4.4.2.5 Editing a cluster node

---

4.3.3 Designing calculation graphs

4.3.3.1 DACCOSIM CalculationGraph metamodel

4.3.3.2 DACCOSIM CalculationGraph Editor

4.3.3.2.1 Quick description of the interface

4.3.3.2.2 Creating a new calculation graph

4.3.3.2.3 Deleting an existing calculation graph

4.3.3.2.4 Modifying an existing calculation graph

4.3.3.2.5 Adding elements from the palette

4.3.3.2.6 Adding factory blocks from the palette

4.3.3.2.7 Adding internal connectors between blocks

4.3.3.2.8 Adding external connectors from/to the external environment

4.3.3.2.9 Defining external variables

4.3.3.2.10 Drawing connections in a calculation graph

4.3.3.2.11 Defining/editing the variables exchanged on internal links

4.3.3.2.12 Defining/editing the variables exchanged on external links

4.3.3.2.13 Adding an annotation on a calculation graph

4.3.3.2.14 Associating a cluster to a calculation graph

4.3.3.2.15 Handling existing elements of the design sheet

4.3.3.2.16 Graphical options

4.3.3.3 Loading a FMU into the palette

4.3.3.4 Clearing the palette

---

4.1.1 DACCOSIM project contributors

4.2.1 Creating a new project

4.2.2 Deleting an existing project

4.2.3 Modifying an existing project

Copyright © 2017 EDF
6  *Post-mortem and real time visualization* .................................................................................................................. 58

6.1  *Post-mortem visualization outside DACCOSIM* ......................................................................................................... 58

6.1.1  Post-mortem visualization after a single node run .................................................................................................. 58

6.1.2  Post-mortem visualization after a multi-nodes run ................................................................................................ 59

6.2  *Real time and post-mortem visualization within the DACCOSIM GUI* ........................................................................ 59

6.2.1  Real time visualization .............................................................................................................................................. 59

6.2.2  Post-mortem visualization ....................................................................................................................................... 61

7  *Use cases* ....................................................................................................................................................................... 62

7.1  *Co-initialization examples* .............................................................................................................................................. 62

7.1.1  tripleEquation case ...................................................................................................................................................... 62

7.1.1.1  Graphs and simulation properties ..................................................................................................................... 62

7.1.1.2  Discussion on the results .................................................................................................................................. 62

7.1.2  equationPair case ....................................................................................................................................................... 63

7.1.2.1  Graphs and simulation properties ..................................................................................................................... 63

7.1.2.2  Discussion on the results .................................................................................................................................. 63

7.2  *Co-simulation examples* .................................................................................................................................................. 63

7.2.1  tankYPipe case ............................................................................................................................................................ 63

7.2.1.1  Graphs and simulation properties ..................................................................................................................... 64

7.2.1.2  Discussion on the results .................................................................................................................................. 64

7.2.2  events case ................................................................................................................................................................. 65

7.2.2.1  Graphs and simulation properties ..................................................................................................................... 65

7.2.2.2  Discussion on the results .................................................................................................................................. 65

7.2.3  tankBarrel case ............................................................................................................................................................ 66

7.2.3.1  Graphs and simulation properties ..................................................................................................................... 66

7.2.3.2  Discussion on the results .................................................................................................................................. 66

7.2.4  simpleHeatTransfer school case ............................................................................................................................. 68

7.2.4.1  Graphs and simulation properties ..................................................................................................................... 68

7.2.4.2  Discussion on the results .................................................................................................................................. 68

7.2.5  pWH case ................................................................................................................................................................. 69

7.2.5.1  Graphs and simulation properties ..................................................................................................................... 70

7.2.5.2  Discussion on the results .................................................................................................................................. 70

7.2.6  heavyHeatTransfer case .............................................................................................................................................. 71

7.2.6.1  Graphs and simulation properties ..................................................................................................................... 72

7.2.6.2  Discussion on the results .................................................................................................................................. 72

7.2.7  iDEAS case ................................................................................................................................................................. 72

7.2.7.1  The building model ............................................................................................................................................. 73

7.2.7.2  The grid model ..................................................................................................................................................... 74

7.2.7.3  Graphs and simulation properties ..................................................................................................................... 75

7.2.7.4  Discussion on the results .................................................................................................................................. 76

8  *Some known limits* ............................................................................................................................................................. 77

9  *Annex 1: about variable step methods implemented in DACCOSIM 2017* ......................................................... 78

9.1  Calculating two approximations with same method ....................................................................................................... 78

9.2  Calculating two approximations with embedded methods ............................................................................................. 80
1 What is DACCOSIM?

DACCOSIM (Distributed Architecture for Controlled CO-SIMulation) is an FMI\(^1\) compatible Master Algorithm generator developed by the RISEGrid institute\(^2\) jointly launched by Electricité de France\(^3\) (EDF) and CentraleSupélec\(^4\) in December 2012 as a component of the Paris Saclay campus.

RISEGrid main objectives are the study, modelling and simulation of smart distribution power grids and their interactions with the whole energetic system.

Research topics cover both theoretical aspects and more applicative and industrial ones. RISEGrid aims to be in perfect accordance with the strong and challenging evolutions of the electric power systems in Europe: opening of electricity markets, development of decentralized production, ubiquitous information and communication means, decentralized control and regulation algorithms...

Research studies carried out in the Institute combine the multidisciplinary aspects of smart grids (power systems, control, information systems, telecommunication networks...). In addition, new tools for modelling and simulation of complex systems are deeply investigated.

Smart grids are made of numerous interacting subsystems: power grid, automated meter management, centralized and decentralized production, demand side management (including smart charging for electric vehicle), storage, ICT resources...

All these subsystems need to be equally taken into account to obtain a fairly realistic representation of the behavior of the whole system. For this purpose, RISEGrid has made at the beginning the choice to develop their wide system tool betting on the co-simulation principle in order to associate heterogeneous active components exported from various modeling tools.

DACCOSIM is a major step towards a multi-domain co-simulation including both time-continuous dynamic systems (for multi-physics purpose) and discrete time simulators (for ICT subsystem).

DACCOSIM is based on the FMI-CS (FMI for Co-Simulation) in order to couple active components (simulators) within a global co-simulation environment. These simulators are executable codes exported from different tools as distribution files called FMUs (Functional Mock-up Unit) each including both a model and a numerical solver.

DACCOSIM is able to take advantage of the main features available in the version 2.0 of the FMI standard (FMI 2.0 July 25, 2014) that are variable communication step size, inputs extrapolation, FMU state and rollback, directional derivatives... as far as the coupled FMUs support them.

1.1 DACCOSIM project contributors

DACCOSIM is a RISEGrid project permanently improved, tested and maintained by the following core-team:

- Mathieu Caujolle (PhD, research engineer at EDF Lab Paris-Saclay, France),
- Jean-Philippe Tavella (simulation expert & research engineer at EDF Lab Paris-Saclay, France),
- Stéphane Vialle (professor at CentraleSupélec, University Paris-Saclay, Metz, France),
- Virginie Galtier (associate professor at CentraleSupélec, University Paris-Saclay, Metz, France),
- Michel Ianotto (associate professor at CentraleSupélec, University Paris-Saclay, Metz, France).

In addition, every year, one-time contributors participate to the development of DACCOSIM, especially students from CentraleSupélec and various high schools trainees at EDF Lab Paris-Saclay.

The DACCOSIM project team plans to deliver each year a new release of the DACCOSIM tool.

\(^1\) Functional Mock-up Interface (https://www.fmi-standard.org/).
\(^3\) https://www.edf.fr/en/the-edf-group/world-s-largest-power-company/activities/research-and-development
\(^4\) http://www.supelec.fr/374_p_14603/welcome.html
The first public version of the DACCOSIM application was released in October 2015 (DACCOSIM 2015). DACCOSIM 2017 is the most recent version of DACCOSIM now available for downloading (https://daccosim.findry.supelec.fr/).

1.2 DACCOSIM licensing mode

DACCOSIM is released under the AGPL license (http://www.gnu.org/licenses/agpl-3.0.en.html). The DACCOSIM application uses the “JavaFMI wrapper” and the “JavaFMI Builder” both developed and maintained by an academic partner team from SIANI, University of Las Palmas, Spain (https://bitbucket.org/siani/javafmi/wiki/Home).
2 Overview of DACCOSIM 2017

DACCOSIM 2017 emphasizes a complete and user-friendly interface for configuring and performing local or distributed co-simulations with potentially many heterogeneous FMUs compliant with the co-simulation part of the FMI 2.0 standard (FMI-CS 2.0).

2.1 Main DACCOSIM features

The DACCOSIM application consists in three separate but complementary parts:

- A Graphical User Interface (GUI) to achieve the preparation and implementation of co-simulations, especially:
  - calculation graphs design (i.e. the blocks to be considered and the data to be exchanged between them) using the “Calculation Graph editor”;
  - calculation resources definition using the “Cluster editor”;
  - parameterization of the co-simulations (i.e. duration, co-initialization method, time step control strategy...);
  - master codes generation and compilation;
  - co-simulations execution and results visualization;

All the GUI features are described in detail in chapter 4.

- A dedicated underlying computation package able to manage all the tasks relative to the co-simulation either on a single computer or on a cluster of calculation nodes;

- A Parallel Python deployment tool (DacRun) to automate the operating process on distributed calculation nodes, especially pre-processing (distribution and compilation of generated codes), co-simulation and post-processing (collect of results and log files).

⚠️ All the codes generated by DACCOSIM can run under 32-bit or 64-bit Windows or Linux environment. 64-bit FMUs should be used in a 64-bit JRE and 32-bit FMUs should be used in a 32-bit JRE: mixing 32-bit and 64-bit FMUs is not allowed with DACCOSIM 2017.

Figure 1: Illustration of the main steps involved in the processing of co-simulations with DACCOSIM
2.2 DACCOSIM control architecture

The approach used in DACCOSIM is to define a calculation graph consisting of blocks (mainly FMUs) that are connected by logical links and potentially distributed on different computation nodes. This graph is then translated into Java master codes in conformance with the features described in the FMI-CS 2.0 standard. More precisely, DACCOSIM offers:

- The construction of the global causal dependency graph, automatically built both from the FMUs (internal dependencies) and the calculation graph (external dependencies) in order to determine the causal chains and loops involving all the exposed variables;
- A global and distributed co-initialization method based on a generalized Newton-Raphson algorithm to solve initial values propagation whatever the nature of the dependency graph (purely cyclic, purely acyclic or hybrid);
- An ordered (pessimistic) or overlapped (optimistic) orchestration mode inside distributed masters that can operate with constant time steps or with adaptive time steps controlled by one-step methods (Euler or Richardson) or a multistep method (Adams-Bashforth);
- Two approximate event detection algorithms while waiting for a next FMI-CS version of the FMI standard able to correctly handle hybrid co-simulations;
- An input data extrapolation for a better precision in co-simulation results.

DACCOSIM generated master codes follow a **centralized hierarchical approach** where a **unique global master** located on one cluster node is in charge of handling the control data coming from several **local masters** distributed on some other cluster nodes and taking step by step decisions based on these information. Every master, whether global or local, aggregates these control data that are coming from each FMU wrapper present on its cluster node. This is done before communicating synthesized information to the global master.

The control data exchanged between masters and between FMUs and masters are called “vertical data”. Of course when the co-simulation is run on a single machine, only one master code is generated by DACCOSIM.

The originality of the DACCOSIM architecture lies in the fact that the data to be exchanged between FMUs at each communication step are directly transmitted from the senders to the receivers without passing by a master.

The masters, the wrapped blocks (mainly FMUs with wrappers) as well as the communication channels between them are automatically generated by DACCOSIM by translating the calculation graph defined by the user via a GUI (Graphical User Interface).

![Figure 2: Illustration of the distributed global / local master architecture](image-url)
3 Installation procedure

DACCOSIM 2017 is a Java application distributed as a plug-in for the Eclipse environment.

3.1 Required Eclipse and Java distribution

Here are the recommended Java and Eclipse environment to use:

- the Neon distribution of Eclipse (Eclipse IDE for Eclipse Committers) that can be downloaded from http://www.eclipse.org/downloads/eclipse-packages/;
- the 1.8 Java Runtime Environment (JDK) that can be downloaded from http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html.

The installation of Eclipse and JDK should be consistent with the host machine, i.e. a 64-bit Eclipse and JDK should be installed on a 64-bit computer.

Prior installation of Eclipse and JDK is not detailed in this documentation. The following only explains how Eclipse plug-ins and DACCOSIM 2017 codes can then be installed.

However, for a 64-bit Windows or 64-bit Linux environment, the DACCOSIM 2017 website (https://daccosim.foundry.supelec.fr/) proposes to download a zipped “Eclipse_Neon” folder including a suitable Neon distribution of Eclipse with all the required plug-ins. After unzipping of this file, only the DACCOSIM codes have to be added more (refer to § 3.3).

In all cases, pay attention to some system environment variables. For example on a Windows machine:

- C:\Program Files\Java\jdk1.8.0_73\ should be added to the JAVA_HOME variable;
- C:\Program Files\Java\jdk1.8.0_73\bin should be added to the PATH variable.

3.2 Eclipse plug-ins installation

DACCOSIM 2017 depends on the following Eclipse plug-ins:

- Acceleo 3.6;
- EMF- Eclipse Modeling Framework SDK 2.12;
- GEF (MVC) SDK 3.11.

This paragraph details their installation and is based on figures from Eclipse Neon. Some very light adaptations are required when a previous Eclipse version is used.

5 Do not forget the last \ in this path!
Plug-ins can be installed with the dedicated Eclipse helper that can be accessed through the “Install New Software...” command of the Eclipse “Help” menu.

![Diagram](image.png)

**Figure 3: Access to the Eclipse plug-ins installation helper**

In the Eclipse plug-ins installation helper, select the URL “Neon - http://download.eclipse.org/releases/neon” in the “Work with:” field. Then unfold the “Modeling” folder, select the plug-ins to be added, click the “Next” button and follow the instructions given by the helper.

It is advised to check the “Contact all update sites during install to find required software” checkbox so that Eclipse searches and automatically installs the plug-in dependencies.
To be sure a complete and correct plug-in installation has been done, please use the “Installation Details” command of the Eclipse “Help” menu.

3.3 Required Visual C++ Redistributable Packages

FMUs are basically archive ZIP files that are each composed of an XML model description and a compiled C code. On Windows, running an FMU requires to install the redistributable packages associated to the compiler that were used to generate the FMUs. The most common are the Visual C++ Redistributable Packages. It is recommended to install the versions 2010, 2012, 2013, or 2016 in both 32 and 64 bit.

3.4 Source codes installation

Download first the DACCOSIM_2017.zip file from the website (https://daccosim.foundry.supelec.fr/) and unzip it locally on the host machine.

Then, import the DACCOSIM projects into the Eclipse workspace thanks to the “Import...” command of the Eclipse “File” menu.

After selection of the “Existing Projects into Workspace” wizard in the “General” folder, click the “Next” button and give in the “Select root directory:” the folder where the DACCOSIM files have been locally installed.

Finally, select all the projects in the “Projects:” window and click the “Finish” button.
After building of the workspace, the Eclipse “Package Explorer” should look like the following screen:

Figure 5: How to import the DACCOSIM projects

Figure 6: Overview of the DACCOSIM 2017 projects after installation
The thirteen DACCOSIM 2017 folders are packaged as follows:

- Library used by the DACCOSIM global and local masters to control the co-simulation:
  
  fr.daccosim

- Resources for automatic generation of Java master codes with Acceleo:
  
  fr.daccosim.generator
  fr.daccosim.generator.ui

- Resources for the DACCOSIM Script Language management:
  
  fr.daccosim.manager

- Resources for the calculation graph editor:
  
  fr.daccosim.model.calculationgraph
  fr.daccosim.model.calculationgraph.edit
  fr.daccosim.model.calculationgraph.editor
  fr.daccosim.model.calculationgraph.gef

- Resources for the cluster editor:
  
  fr.daccosim.model.cluster
  fr.daccosim.model.cluster.edit
  fr.daccosim.model.cluster.editor

- Resources for the FMUs management:
  
  fr.daccosim.model.fmu
  fr.daccosim.model.fmu.edit
4 Graphical User Interface of DACCOSIM 2017

As previously described, the purpose of the DACCOSIM GUI is to facilitate the co-simulation studies from the pre-processing stage (configuration) to the post-processing (results visualization).

Each function that can be invoked from this graphical interface is described in detail in this chapter beginning with the activation of the GUI and the management of DACCOSIM projects under Eclipse.

4.1 Activating the GUI

Once the different project folders have been imported in the Eclipse environment, the DACCOSIM GUI is accessible by selecting one of the folders in the Eclipse “Package Explorer” view.

Then select the “Run As” command of the project context menu and click the “Eclipse application” option. A new specialized Eclipse environment based on the GEF framework and including all the features of DACCOSIM 2017 is launched and the DACCOSIM GUI appears.

A quicker way to get there is possible by simply clicking the “Run” button of the Eclipse toolbar.

Figure 7: Launching the DACCOSIM GUI

Two Eclipse applications are now launched on the computer. The first one is helpful for debug purpose as its “Console” tab logs all the traces of the child Eclipse application running DACCOSIM.
4.2 Handling DACCOSIM projects

A DACCOSIM project must be defined by the user before describing all the resources needed to run a co-simulation. Each project appears as a folder visible in the “Package Explorer” of the Eclipse environment and may contain structured information organized in calculation graphs and cluster descriptions.

Calculation graphs and clusters are the main objects that will be detailed at length further in this documentation.

Design of calculation graphs and definition of calculation resources are only possible within existing DACCOSIM projects.

4.2.1 Creating a new project

To create a new project, the user can activate the “New → Project...” command of the Eclipse “File” menu. The “Project” wizard is to be selected inside the “General” folder. Then, after a click on the “Next” button, a project name and a full location should be given into the “New Project” window. Do not forget to click the “Finish” button.

By default the new project is created in the sub-folder “runtime-EclipseApplication” located in the folder where the DACCOSIM application has been installed.

![Figure 8: Creating a new DACCOSIM project](image)

The new project is now visible in the Eclipse “Package Explorer”.

4.2.2 Deleting an existing project

Each DACCOSIM project visible in the Eclipse “Package Explorer” can be deleted with the “Delete” command available from its context menu. The files associated to this project can optionally be deleted on disk.

4.2.3 Modifying an existing project

Each DACCOSIM project visible in the Eclipse “Package Explorer” can contain one or more calculation graphs (refer to § 4.3) and optionally one or more calculation resources descriptions (refer to § 4.4).

More details can be found in the following paragraphs.

4.3 Designing calculation graphs

Several elements have to be defined in order to perform a co-simulation:

- The simulation blocks involved (mainly FMUs);
- The variables they exchange;
- A calculation resource (local computer or cluster) and how the graph blocks are mapped on it;
- Simulation properties (co-initialization method, simulation duration, time step adaption strategy...).
All these information are defined within a “DACCOSIM calculation graph” and stored in a file with a .dcg (or .daccosimcalculationgraph) file extension.

The attributes accessible or editable by the user and the edition window are presented in the following paragraphs.

### 4.3.1 DACCOSIM CalculationGraph metamodel

The DACCOSIM CalculationGraph metamodel defines the blocks to consider in the co-simulation, the data exchanges between them and the cluster nodes associated to each of them. This model can be constructed and edited as a graph representation within a graphical interface.

The FMU block objects present within the CalculationGraph metamodel are described by a large number of attributes. Most of them come from the modelDescription.xml file of the FMUs and are non-editable while some others can be modified by the user through the DACCOSIM GUI (in italic):

- **Can Be Instantiated Only Once Per Process**: this Boolean indicates whether a FMU can be instantiated more than once on a cluster node. If false, several copies of the same FMU can be used within the calculation graph.
- **Can Get And Set FMU State**: this Boolean indicates if the FMU can take and restore snapshots of its internal and external states. If false, no variable step simulation method can be used within the calculation graph.
- **Can Handle Variable Communication Step Size**: this Boolean indicates if the FMU can be run with variable simulation time steps. If false, no variable step simulation method can be used within the calculation graph.
- **Can Interpolate Inputs**: this Boolean indicates if a FMU can interpolate its inputs when derivatives are provided. If false, there will be no input extrapolation within the calculation graph.
- **Can Not Use Memory Management Functions**: this Boolean indicates if a FMU can use its own functions for memory allocation and freeing only.
- **Can Run Asynchronously**: this Boolean indicates if a FMU can carry out the fmi2DoStep(...) call asynchronously. Note that asynchronous mode is not possible with DACCOSIM 2017.
- **Can Serialize FMU State**: this Boolean indicates if a FMU can serialize the internal FMU state. Not used with DACCOSIM 2017.
- **FMI Version**: version number of the FMI standard used to export the FMU. DACCOSIM only supports the version 2.0 of the FMI standard.
- **FMU Generation Date And Time**: date and time when the FMU modelDescription.xml file was generated.
- **FMU Generation Tool**: optional name of the tool that generated the FMU.
- **Internal Solver Tolerance**: this parameter defines the internal FMU solver tolerance. If this parameter is not provided by a FMU, 10^{-6} is given as default value.
- **Max Output Derivative Order**: this Boolean indicates if a FMU can provide derivatives of outputs with a given maximum order.
- **Name**: name of the FMU block. Each block instantiated in the CalculationGraph should have a unique identifier. The default name is the file name used when importing the FMU.
- **Needs Execution Tool**: this Boolean indicates if a tool is needed to execute the model. The FMU just contains the communication path to this tool.
- **Node**: name of the cluster node on which the FMU will be simulated. Default value is localhost.
- **Path**: full path name to access the FMU file. The default path is the file location used when importing the FMU.
- **Port**: port identifier used for the considered cluster node when TCP communication between FMUs hosted on other nodes are considered. This parameter is essential when cluster co-simulations are performed. Default value is 0.
- **Provides Directional Derivative**: this Boolean indicates if a FMU can provide the directional derivative of the equations at communication points.

Copyright © 2017 EDF
4.3.2 DACCOSIM CalculationGraph Editor

The interface used to design calculation graphs consists of four different parts.

4.3.2.1 Quick description of the interface

These parts are:

a) A toolbar that allows performing one-click actions;

b) A design sheet where one or more calculation graphs can be graphically edited, each graph being attached to a different tab;

c) A palette that enables drag-and-drop actions within the design sheet and the drawing of links via connectors;

d) A property window dedicated to the edition of some block attributes.

Figure 9: DACCOSIM graphical user interface for designing calculation graphs

Different functionalities are enabled with this interface and assigned to these four areas.

All the functionalities are detailed in the following sections.

4.3.2.2 Creating a new calculation graph

A new instance of “CalculationGraph” is created by using the command “File -> New -> Other” (or Ctrl+N as shortcut) and by selecting the “DACCOSIMCalculationGraph Model” wizard in the “DACCOSIM Model Creation Wizards” folder.
After a click on the “Next” button, a new empty calculation graph is visible in the Eclipse “Package Explorer” under the project on which it depends.

### 4.3.2.3 Deleting an existing calculation graph

Each DACCOSIM calculation graph visible in the Eclipse “Package Explorer” can be deleted with the “Delete” command available from its context menu. The file associated to this calculation graph can optionally be deleted on the disk.

### 4.3.2.4 Modifying an existing calculation graph

By double-clicking on an already existing DACCOSIM calculation graph visible in the Eclipse “Package Explorer”, the information previously defined for this graph are loaded and can be graphically modified in the design sheet.

More details can be found in the following paragraphs.

### 4.3.2.5 Adding elements from the palette

All objects visible in the palette can be dragged and dropped into the current calculation graph of the design sheet.

A button is provided in the right corner of the interface to hide or reveal the palette.

Particularly when FMUs have already been loaded in the palette (refer to § 4.3.3), drag-and-drop actions perform a copy of the FMUs to involve in the calculation graph being edited.
4.3.2.6 Adding factory blocks from the palette

Some predefined factory blocks are available in the palette. They can be added in the current calculation graph and then manipulated as FMU blocks.

These blocks correspond to the following predefined functions:

- Addition or subtraction of several (at least two) inputs;
- Multiplication or division of several (at least two) inputs;
- Signal gain creation (only one input);
- Signal offset creation (only one input).
4.3.2.7 Adding internal connectors between blocks

Before connecting two blocks instantiated in the design sheet of the calculation graph, internal input and output connectors have first to be created within each block.

These connectors and the associated connections possess the following properties:

- several variables can be transmitted through the same link, enabling efficient communication between two blocks whether on the same node or on another one;
- an input connector must be connected to one and only one output connector;
- communication from a block A to a block B is limited to a single link between an output connector from block A and an input connector from block B. All the necessary variable exchanges are defined with this link. No additional link from A to B is authorized.

The internal connectors are added to blocks by selecting / dragging the desired connector type from the palette and dropping it within the target block.

![Figure 14: Internal connectors in the palette](image)

4.3.2.8 Adding external connectors from / to the external environment

A calculation graph can exchange data with its environment thanks to external input and output connectors added in the graph.

These connectors and the associated connections possess the following properties:

- several variables can be transmitted through the same link, enabling efficient communication between the external environment and the internal blocks;
- an external input connector must be connected to one and only one internal input connector;
- communication from the external environment is limited to a single link between an external input connector to a given block input connector. All the necessary variable exchanges are defined with this link. No additional link is authorized;
- for a given block, communication to the external environment is limited to a single link between an internal output connector from this block to an external output connector. All the necessary variable exchanges are defined with this link. No additional link is authorized.
The external connectors are added to the graph by selecting / dragging the desired connector type from the palette and dropping it within the design sheet.

![External connectors in the palette](image)

If no external variables were previously defined, the drag / drop of external connector triggers the apparition of a window in order to add external variables into the calculation graph (see 4.3.2.9).

**4.3.2.9 Defining external variables**

This action is normally performed by selecting the “Configure external input and output variables” button accessible from the toolbar.

![Configuring external input or output variables](image)

A window appears to define the external variables with attributes close to the FMI standard definitions:

- **Name**: name of the external variable;
- **Nature**: *Input* or *Output* variable. Default value is *Input*;
- **Type**: *Real*, *Integer*, *Boolean*, *String* or *Enumeration*. Default value is *Real*;
- **Variability**: *Constant*, *Discrete* or *Continuous*. Default value is *Continuous*;
- **Init mode**: *Undefined*, *Exact*, *Approximated* or *Calculated*. Default value is *Undefined* for inputs (because not applicable) and *Exact* for outputs;
- **Description**: free text field to describe the variable.

Three buttons allow the following actions:

- **Create external variable**: button to create the variable with the previously defined parameters;
- **Remove selected variables**: button to remove one or more variables selected among the “External inputs” or “External outputs” lists;
- **Close**: button to close the window.
In addition, for external inputs, a default initial value can be defined thanks to the button “Define external inputs properties” accessible from the toolbar.

The following interface is used to set the initial value of each external input:

4.3.2.10 Drawing connections in a calculation graph

When the required (external) input and output connectors have been positioned on the blocks, the object Link should be selected from the palette to draw the communication channel existing between them.
By doing this, the mouse selection mode changes. To draw a new inter-block connection, a free output connector of the source block should be first selected, and then, a free input connector of the target block. After this operation, a direct link should appear between the selected output and input connectors. Not-connected input and output connectors appear in gray instead of white and black respectively.

Returning to the normal mouse selection mode is done by pressing Escape or clicking on any other item of the GUI palette.

It is to be noted that the links can be broken with bend points. The points present in the middle of the link segment can be displaced allowing the creation of two new link segments with new bend points.

In a very similar way, the Link object is also able to draw external connections between the environment and blocks inside a calculation graph.

4.3.2.11 Defining / editing the variables exchanged on internal links

Once a link between two blocks has been established, the user has to select the output variables of the source component that will be associated to the input variables from the target component. A link without defined variable exchange appears in red instead of black.
This action is done by right-clicking on the considered link and selecting the “Define output input connections” item of its contextual menu.

![Figure 22: Contextual menu for links](image)

After this action, an interface consisting of three list boxes appears:

- The list box on the left allows to select one output of the source block;
- The list box in the middle allows to select one of the available inputs (i.e. not already associated to another output from this source block or from any other block) of the target block;
- The list box on the right (matching list box) displays the variables already associated in this link.

The input / output associations are performed with the following rule: **an output can be connected to several inputs, but an input can only be connected to a single output**.

- When an output and an input are both selected, clicking on the “Register the connection” button saves the connection: the associated two variables appear now linked in the matching list box and the selected input no longer appears in the middle list box;
- When an association of variables is selected in the matching list box, clicking on the “Delete the connection” button removes the association: it no longer appears in the matching list box and the input appears again in the middle list box of the interface.

![Figure 23: Configuration of internal links](image)

It is also to be noted that a tooltip appears as an information box when the user hovers a link between two blocks with his mouse pointer.

![Figure 24: Tooltip associated to an internal link](image)

4.3.2.12 Defining / editing the variables exchanged on external links

A similar mechanism is available to define and edit the variables to be exchanged with the external environment.

This action is done by right-clicking on the considered link and selecting the “Define output input connections” item of its context menu.

The interface is adapted to incoming or outgoing variables (e.g. next figure refers to links with outgoing variables).
Figure 25: Configuration of links with the environment (outgoing variable)

Again, a tooltip appears as an information box when the user hovers a link between a block and the external environment with his mouse pointer.

4.3.2.13 Adding an annotation on a calculation graph

The user can add comments to a calculation graph by adding an annotation on the design sheet.

A free text can then be entered in the opened box on the design sheet and will be seen as a comment.
4.3.2.14 Associating a cluster to a calculation graph

As previously pointed out, a calculation resource must be associated to a calculation graph.

A default resource is implicitly defined by DACCOSIM 2017. This resource corresponds to a standard PC environment so it contains one “cluster” node named “localhost” with a node port set to 3000 by default. An automatic association is done between this default cluster node and any block added to the calculation graph. A run with this setting implies that only one single global master file will be generated and executed on a local machine.

If a multi-nodes cluster is to be used, a cluster must be associated to the calculation graph. This association is possible under the condition that the desired cluster resource has already been defined (refer to § 4.4). By default, an automatic association is done between the cluster master node and any block added to the calculation graph. This association is edited afterwards as described below.

This cluster resource must first be loaded with the button “Define bloc <-> cluster association” from the toolbar.

When clicked, this button displays the following window where the single node “localhost” is by default associated:

Loading another resource is now possible via the button “Load Cluster” and a file explorer appears which enables to select the file containing the cluster characteristics to import.
After a cluster loading, the mapping between the blocks defined in the current calculation graph and the cluster nodes is possible as the configuration window changed appearance:

As it is not possible to define a standalone global master, at least one FMU block must be associated to the node tagged “global master”.

Other FMU blocks can be freely mapped to any cluster node available in the cluster. A color is automatically associated by DACCOSIM to each cluster node so that with only one glance on the design sheet the user can get an overview of the resource mapping. This set of colors can be manually edited by the user using the associated buttons.

Be careful all the blocks defined in the calculation graph are mapped on cluster nodes, each with a different communication port.

The button “All->global master” is useful to reallocate all the FMU blocks on the cluster node of the global master. This action allows to run the co-simulation on the hosting single node computer which is equivalent to the “localhost” resource.
4.3.2.15 Handling existing elements of the design sheet

The graphical interface of DACCOSIM enables to perform several generic actions to manage the calculation graphs or the links between the blocks:

- Saving the current calculation graph (Ctrl-S);
- Selecting one or more graph elements (Ctrl-A);
- Moving one or more blocks of the calculation graph;
- Suppressing calculation graph element(s) (Delete key);
- Undoing a command performed in the GUI (Ctrl-Z);
- Redoing a command previously undone (Ctrl-Y);
- Copying / pasting a part of the calculation graph (Ctrl-C followed by Ctrl-V). This command is only available for FMUs with their `canBeInstantiatedOnlyOncePerProcess` attribute set to `false`. Connections and variables exchanged inside the selection are preserved;
- Editing the name of a block instantiated in the calculation graph.

Some of these commands (Undo, Redo, Delete, Copy and Paste) are graphically accessible from the toolbar:

![Figure 32: Usual options available from the toolbar](image)

4.3.2.16 Graphical options

Graphical functions helping in the design of the calculation graphs are accessible from the toolbar:

- Some zoom buttons to enlarge or reduce the size of the edited diagram;
- A “Grid” button to display / hide the grid on which the components can be positioned;
- A “Snap to Geometry” button so that the geometry of the elements already positioned on the design sheets acts as a magnet in order to help positioning the new elements.

![Figure 33: Graphical options available from the toolbar](image)

An additional graphical option “Snap to grid” is accessible in the “Editor properties” tab of the “Simulation setup” window (refer to § 4.5.6.3). When activated with the grid displayed, the grid lines and points act as a magnet in order to help positioning the new elements.

“Grid”, “Snap to Geometry” and “Snap to grid” options can be configured in the “Simulation setup” window.

![Figure 34: Example illustrating the “Snap to grid” functionality](image)
4.3.3 Loading a FMU into the palette

FMUs should be loaded into the palette interface before a drag-and-drop action into the calculation graph design sheet.

This action is performed by selecting the “Load FMU into Palette” button accessible from the toolbar.

A file explorer that allows the selection of one or more FMUs appears. Only files with .fmu extensions can be chosen. All FMUs should be in conformance with the FMI-CS 2.0 standard.

When one or more files have been selected by the user, their content is read and, if valid, the FMUs are loaded into the palette for future drag-and-drop actions.
4.3.4 Clearing the palette

Already loaded FMUs visible in the palette can be all removed by selecting the “Clear imported FMUs from Palette” button accessible from the toolbar.

4.4 Defining calculation resources

Performing a co-simulation requires defining the available data processing resources (cluster and cluster nodes characteristics) where the codes are to be deployed and launched.

These resources information are defined within a “DACCOSIM cluster”. A file defining a cluster has a .dcl (or .daccosimcluster) file extension. The structure of cluster files and the way to create them are described in the following paragraphs.

Except in the particular case where the co-simulation is to be run on the host machine (cluster named “localhost” by default), it is expected to specify in an exhaustive way the cluster nodes where the global and local master codes are to be loaded.

4.4.1 DACCOSIM Cluster metamodel

The DACCOSIM Cluster metamodel is used to specify the resources used by the co-simulation.

The Cluster objet is used to store information regarding every cluster node defined as Node objects. Each Node object contains the following information, all editable:

- **Architecture**: cluster node architecture (32 / 64 bits);
- **Host Name**: DNS name or IP address of the cluster node (no space or special character allowed);
- **OS**: operating system of the cluster node (Windows / Linux);
- **Port**: port ID to be used on the cluster node for TCP-based communication;
- **RAM**: memory capacity available on the cluster node, in Go.

⚠️ Note that only “Host Name” and “Port” are exploited by DACCOSIM 2017.

4.4.2 DACCOSIM Cluster Editor

4.4.2.1 Creating a new cluster

A new instance of “ClusterModel” is created by using the command “File” -> “New” -> “Other” (or Ctrl+N as shortcut) and by selecting the “DACCOSIM Cluster Model” wizard in the “DACCOSIM Model Creation Wizards” folder.
A tree structured view appears in an empty “Resource Set” window in order to define the cluster nodes of the cluster. After some cluster nodes definition, the new cluster is visible in the Eclipse “Package Explorer” under the project on which it depends.

4.4.2.2 Deleting an existing cluster

Each DACCOSIM cluster visible in the Eclipse “Package Explorer” can be deleted with the “Delete” command available from its context menu. The file associated to this cluster can optionally be deleted on disk.

4.4.2.3 Modifying an existing cluster

By double-clicking on an already existing DACCOSIM cluster visible in the “Package Explorer”, the information previously defined for the cluster are loaded and can be modified in the “Resource Set” view.

More details can be found in the following paragraphs.

4.4.2.4 Adding nodes to a cluster

Within the “Resource Set” view, new Node objects can be associated to the cluster by right clicking on the cluster element (root of the tree structure) and selecting “New Child” -> “Node” in the context menu.
4.4.2.5   Editing a cluster node

After a cluster node selection, the attributes of this node can be edited through the “Properties” tab of the bottom part of the interface. If this field does not appear, selecting the “Show properties view” item in the context menu associated with the node makes the field visible.

![Figure 41: Tree structured information defining a cluster resource](image)

⚠️ All the nodes of a cluster must be defined with the same operating system (either Windows or Linux).

⚠️ Also note that the editable information defined for a cluster node (Accelerator, Network Information, Core) are not yet exploited by DACCOSIM 2017.

4.5   Preparing co-simulations

4.5.1   FMU settings

A contextual menu is available for FMU blocks drawn on the design sheet. The given commands mainly help the user to initialize variables, define the variables tolerance, indicate the variables to log as results, redefine the internal dependencies and define the parameters for the co-simulation.

![Figure 42: contextual menu for FMU blocks](image)

4.5.1.1   Defining the variables initial values

It is often necessary to define the initial values of the input variables of FMUs so that the co-simulation can start. To do so, select the “Define the variables initial values” item of the contextual menu of each concerned FMU.

An interface consisting of two tables appears. These tables display the FMU variables for which it is possible to define an initial value, their type as well as the value that may have been already attributed to them from the lifeDescription.xml file of the FMU.
In conformance with the FMI-CS 2.0 standard, some variables can be initialized before the initialization state of the FMU while some others cannot be initialized elsewhere than in this state. The variables are then ranked in two lists as follows:

- The “Before entering initialization mode” list gathers the variables that can be initialized before entering the initialization state (after a \texttt{fmi2Instantiate()} call and before a \texttt{fmi2EnterInitializationMode()} call) that is to say:
  - any variable with (variability="fixed" or variability="tunable") and causality="parameter";
  - any variable with (variability="fixed" or variability="tunable") and (causality="calculated parameter" or causality="local") and initial="approx";
  - any variable with (variability="discrete" or variability="continuous") and (causality="output" or causality="local") and (initial="exact" or initial="approx");

- The “After entering initialization mode” list gathers the variables that can be initialized only in the initialization state (after a \texttt{fmi2EnterInitializationMode()} call and before a \texttt{fmi2ExitInitializationMode()} call) that is to say:
  - any variable with (variability="fixed" or variability="tunable") and causality="parameter";
  - any variable with (variability="discrete" or variability="continuous") and (causality="output" or causality="local") and initial="exact";
  - any variable with causality="input".

A filter can be applied on the listed variables to select a subset of them whose name contains a given string (no uppercase lowercase distinction).

The four buttons of this initialization interface are used to:

- validate the initial values entered by the user (“Validate” button);
- discard any change performed since the opening of the window (“Cancel” button);
- restore the initial values to their state at the opening of the window (“Restore” button);
- restore the initial values in their original state read from the \texttt{modelDescription.xml} file from the FMU (“Restore defaults” button).

A tooltip appears and displays a variable description when hovering a variable of this initialization window with the mouse pointer. This functionality is only available if such an information is available within the \texttt{modelDescription.xml} file of the FMU.

4.5.1.2 Defining the variables tolerance

Tolerance settings defined on FMU internal variables are used to automatically trigger rollbacks in case an adaptive time step control strategy is considered: if the approximation made on one of these variables is considered higher than the set tolerance value, a rollback is performed and the time step is reduced by the co-simulation masters.
Tolerances can be set via the “Define the variables tolerance” item of the contextual menu of each concerned FMU. A window appears enabling to choose a value for each variable among the exposed variables within the FMU that are all variables with (variability="output" or variability="local") and (causality="continuous" or causality="discrete").

A filter can be applied on the listed variables to select a subset of them whose name contains a given string (no uppercase lowercase distinction).

A relative tolerance is by default associated to all FMU outputs. An editable field is proposed for this outputs relative tolerance with \(10^{-5}\) as default value. This tolerance can be automatically set to all outputs thanks to the “Apply to outputs” button.

The “Restore defaults” button reassign the value \(10^{-5}\) to all outputs and remove the tolerance for all internal variables.

The “Restore” button restores the tolerance of the variables as found at the opening of the window before any modification was carried out.

![Figure 44: Interface enabling the setting of FMU variables tolerance](image)

4.5.1.3 Selecting the variables to store

All the FMU exposed variables (inputs, outputs and internal variables) can be logged when they are selected via the “Select the variables to store” item of the contextual menu of each concerned FMU.

By default, all the FMU inputs and outputs are considered as variables to log. These variables are located at the top of the FMU exposed variables list. Some of them can be deselected if not considered as significant.

A filter can be applied on the listed variables to select a subset of them whose name contains a given string (no uppercase lowercase distinction).

The “Restore defaults” button reselects all the inputs and outputs as the only variables to be logged.

The “Restore” button restores the logging state of the variables as found at the opening of the window before any modification was carried out.
4.5.4 Editing the internal dependencies

By default, dependencies of the whole calculation graph are automatically computed according to the internal dependencies set inside the FMU (see 4.5.4).

Unhappily, depending on the tool that has exported the FMU, some internal dependencies can be missing or some of them can have incorrectly been added. For this reason, the default internal dependencies can be completed or modified via the “Edit internal dependencies” item of the contextual menu of each concerned FMU. A window consisting of three list boxes appears:

- The list box “FMU outputs” on the left allows to select one of the defined internal outputs;
- The list box “FMU inputs” in the middle allows to select one or more defined internal inputs;
- The list box “O/I Dependency” on the right (matching list box) is passive: it only displays the dependencies already defined.

By default, these three list boxes are prefilled according to dependencies deduced from the modelDescription.xml file of the FMU. However, if no dependency information is found in the FMU, each output is made dependent on all the FMU inputs.

The button “->” associates a selection of internal inputs with the current internal output.

The button “<-” disassociates a selection of internal inputs from the current internal output.

The button “Full Dependencies” sets dependencies with the assumption that each external output depends on all external inputs.

The button “Clear Dependencies” clears the list box “O/I Dependency”.

Copyright © 2017 EDF
4.5.1.5  **Reloading an FMU**

It is possible to update an FMU instance already involved in a calculation graph. This action is useful when the model inside the FMU has changed or when the inputs / outputs within the FMU have been modified. If needed, a new FMU location can be first set in the “Properties” view of the selected FMU block.

![Update of the “Path” attribute of a modified FMU](image)

Figure 47: Update of the “Path” attribute of a modified FMU

Then, a click on the “Reload FMU” item of the contextual menu of the concerned FMU updates the variables list of this FMU as well as their initial values by reading its `modelDescription.xml` file.

The user must then check the incoming and outgoing links for the updated FMU in the case where the names or types of the FMU input and output variables have been changed.

Note that if the “Properties” view is not visible, it is possible to get it either with the contextual command “Show properties view” or the “Show the “Properties” view” button accessible from the toolbar.

![Showing the “Properties” view](image)

Figure 48: Showing the “Properties” view

4.5.2  **Editing project properties**

The project properties can be configured by selecting the “Edit project properties” button accessible from the toolbar.

![The edition of the project properties](image)

Figure 49: The edition of the project properties

A window appears and the parameters accessible in the different tabs of the interface are described in the following paragraphs.
4.5.2.1 Simulation properties

The “Simulation properties” tab allows the definition of the co-simulation parameters.

![Simulation properties tab of the “Simulation setup” window](image)

**Figure 50: “Simulation properties” tab of the “Simulation setup” window**

The accessible fields are:

- **Co-initialization mode**: co-initialization method among:
  - *Co-initialization*, global method adapted to pure acyclic dependency graphs, pure cyclic dependency graphs and hybrid (cyclic / acyclic) dependency graphs,
  - *No co-initialization*, when no co-initialization should be performed.
  Default mode is *Co-initialization*;

- **Tolerance on SCC residuals**: tolerance for the Newton-Raphson algorithm used to calculate initial input values to propagate in each Strongly Connected Component (see SCC in 4.5.4) of the dependency graph when a co-initialization is required. Default value is $10^{-5}$. This field is not accessible when *No co-initialization* has been selected;

- **Max number of iterations**: maximum number of steps considered by the Newton-Raphson algorithm for each SCC during the co-initialization phase. Default value is 100. This field is not accessible when *No co-initialization* has been selected;

- **Start time (s)**: time instant indicating the beginning of the co-simulation (must be real and positive). Default value is 0.0;

- **Stop time (s)**: time instant indicating the end of the co-simulation (must be a real and positive number higher than or equal to the start time value). Default value is 0.0;

- **Max number of steps**: maximum number of simulation time steps. Default value 0 is to be interpreted as no limitation;

- **Stepsize control method**: method used to control the time step among:
  - *ConstantStepSize*: constant time step,
  - *EulerAdaptiveStepSizeControl*: variable time step controlled by the Euler method,
  - *RichardsonAdaptiveStepSizeControl*: variable time step controlled by the Richardson method,
More information on the variable step methods implemented in DACCOSIM 2017 can be found in Annex 1.

Default value is ConstantStepSize;

- **Use bisectional event detection method**: method to use for automatic event detection (on Booleans and integers variables only) when a variable time step method has been chosen. Default value is bisectional (the other method is the adoption of the smallest possible step size until the event time has been exceeded);

- **Adams-Bashforth order**: order (between 3 and 9) of the multi-steps Adams-Bashforth method when this method has been chosen. Default value is 3;

- **Data synchronization mode**: orchestration mode among:
  - Control_First_Send_Last, ordered orchestration where control and data transmission are serialized,
  - Send_First_Control_Last, overlapped orchestration where control and data transmission are parallelized.

This option is only settable when the step size control method is variable and in this case the default mode is Control_First_Send_Last. With constant step size method, the orchestration is forced to Send_First_Control_Last;

- **Extrapolate inputs**: input extrapolation for better accuracy in results (but the performance is slightly impacted) under the condition FMUs are compatible (CanInterpolateInputs attribute set to true);

- **Initial stepsize (s)**: step size (when the step size control method is constant) or step size for the first simulation step (when the step size control method is variable). Default value is 0.0;

- **Minimum stepsize**: value of the time step under which the selected control method cannot go. This field is not accessible with the method ConstantStepSize. Default value is 0.0;

- **Maximum stepsize**: value of the time step over which the selected control method cannot go. This field is not accessible with the method ConstantStepSize. Must be greater than or equal to “Minimum stepsize”. Default value is 0.0;

- **Safety factor**: safety factor used to limit the growth of the time step calculated by an adaptive method. A consensual value is often chosen in the range [0.8, 0.9]. This field is not accessible with the method ConstantStepSize. Default value is 0.9;

- **Intra-node communication protocol**: communication protocol to use inside each node:
  - **OMQ**: OMQ protocol,
  - **Shared Queue**: ad-hoc protocol.

Default value is OMQ.

Parameterization is validated by clicking the “Validate” button.

4.5.2.2 Log properties

The “Log properties” tab allows the definition of parameters for the log and results files.
The accessible fields are:

- **Generate process block logs**: generate a log at block level. Not selected by default;
- **Generate process virtual node logs**: generate a log at virtual node level. Not selected by default;
- **Generate results**: generate a results file at block level (one file for each block). Selected by default;
- **Fuse results at node level**: generate a results file at virtual node level by merging all blocks results files (one result file for each node). Selected by default;
- **Log buffer size**: define the number of result time steps to keep in a memory buffer before copying them on the disk in block result files. Default value is 10;
- **Cell separator**: select a cell separator for results files. Possible values are “;”, “,”, and “.”. Default value is “,”;
- **Decimal mark**: select a decimal mark for results files. Possible values are “,” and “.”. But “.” is the only possibility when the cell separator has been chosen to “,”. Default value is “.”;
- **Convert booleans to integers**: convert Boolean values “true” and “false” respectively into “1” and “0” in results files. Selected by default. It is to be noted that this conversion is only performed when writing the result files.

Parameterization is validated by clicking the “Validate” button.
4.5.2.3  Editor properties

The “Editor properties” tab allows the definition of some editor parameters.

![Figure 52: “Editor properties” tab of the “Simulation setup” window](image)

- **Display grid**: attribute displaying / masking the grid in the design sheet. Selected by default;
- **Snap to geometry**: attribute activating / deactivating geometry as magnet. Selected by default;
- **Snap to grid**: attribute activating / deactivating the grid as magnet. Selected by default;
- **Display node color**: attribute to display a color to distinguish the block mapping on virtual nodes. Selected by default;
- **Grid spacing**: integer defining the size of the grid mesh. Default value is 12;
- **Disable console output**: attribute disabling the display of some information during the run on the “Console” tab of both the mother and the child Eclipse applications. If the console display is disabled, huge performance gain is obtained. Not selected by default;
- **Progress steps size**: define a percentage of the co-simulation duration where a first ‘activity message’ is displayed on the “Console” tab. Next similar messages are logged at multiple dates of this value till the end of the run. Default value is 10.0 %.

Parameterization is validated by clicking the “Validate” button.

4.5.2.4  Java code

The “Java code” tab allows the selection of the DACCOSIM resources folder needed for the compilation of the master codes.

With DACCOSIM 2017, this folder must be given as an absolute path referencing the “DACCOSIM_deployment_setup” sub-folder depending on the DACCOSIM code installation on the machine (e.g. C:\DACCOSIM\DACCOSIM2017\DACCOSIM_deployment_setup).

With this information, DACCOSIM 2017 automatically selects in sub-folders the resources to use for the Java compiler depending on the operating system (Windows or Linux) and the JRE architecture considered (32-bit or 64-bit).
Parameterization is validated by clicking the “Validate” button.

4.5.3 Checking the graph consistency

A check of the graph consistency is accessible from the toolbar via the button “Check scheme consistency”.

This function controls that:

- A global master is assigned to the calculation graph;
- Every block present in the calculation graph has a cluster node and a port assigned. The port is used for inter-block communication when the blocks on each end of the link are assigned to a different node;
- Every block name is unique and each name only contains alphanumeric characters;
- Every port number associated to a cluster node is unique;
- Every input connector is connected to exactly one output connector;
- Every output connector is connected to one or several input connectors (if not, a warning is issued);
- Every external input connector is associated with at least one external input variable;
- Every external input connector is connected with at least one internal input variable;
- Every external output connector is associated with at least one external output variable;
- Every external output connector is connected with only one internal output variable;
- The FMU attributes and the simulation parameters are consistent (e.g. the start and stop time);
- Every FMU with its `canBeInstantiatedOnlyOncePerProcess` attribute set to `true` is only instantiated once on each node cluster;
- A DACCOSIM resources folder is defined (if not, a warning is issued).
4.5.4 Building the dependency graph

The construction of the causality graph is accessible from the toolbar via the button “Build scheme causality graph”.

![The causality graph button](image)

This function launches the construction of the dependency graph in order to visualize both the internal dependencies (set from the `modelDescription.xml` file of each FMU block but possibly updated by the user) and the external dependencies (deduced from the links between the FMU blocks in the calculation graph).

During this stage, chains and cycles existing between all the variables are also detected. The graph chains and cycles define the order in which the graph blocks should be initialized during the co-initialization stage.

Two different display modes are supplied according to different force-directed drawing algorithms where the edges are of more or less equal length and where there are as few crossing edges as possible. Display mode can be chosen thanks to buttons “FR” or “KK”:

- **“FR”** representation displays the graph built with the Fruchterman-Reingold algorithm using a combination of attractive forces on adjacent vertices and repulsive forces on all vertices;
- **“KK”** representation displays the graph built with the Kamada-Kawai algorithm based on the idea of using only spring forces between all pairs of vertices with ideal spring lengths equal to the vertices’ graph-theoretic distance.

“FR” is the default representation. For both representations, the graph is displayed in the tab “Complete graph” where some graphical information are pushed:

- The nodes correspond to the interconnected variables;
- The arcs correspond to the dependencies between variables;
- Nodes with the same color belong to the same block instance;
- Larger nodes are source nodes (or seed vertices), i.e. nodes having no predecessor;
- Full line arcs represent external dependencies between FMUs;
- Dotted line arcs represent internal dependencies inside FMUs;
- The number associated with arcs represents the order in which the variables should be initialized, starting from “0”. Arcs having the same number should be activated in a parallel way.

Some synthesis information are also available beneath the graph:

- The number of the seed vertices (larger nodes without any predecessor);
- The number of the cyclic components or cyclic SCCs (Strongly Connected Component).

When the dependency graph contains at least one SCC, a second tab “Acyclic graph” appears to complete the visualization with a simplified graph where SCCs are each replaced by a super-node.

Two buttons are enabled at the bottom for the graph representation:

- The “Transform mode” button (default mode of the interface) allows translation of the graph in the window using the mouse cursor and zooming in / out using the mouse scroll wheel;
- The “Pick-up mode” button allows the selection and the individual moving of the graph nodes to get a smarter representation.
All these information are useful for the co-initialization process that must start from the seed vertices, propagate values along the arcs and solve step by step all SCCs initialization.

Before building the dependency graph, a check of the graph consistency is always performed in order to ensure that the analyzed co-simulation graph is valid. Otherwise, an error message appears and the dependency graph building is not performed.

4.6 Java code generation and compilation

DACCOSIM 2017 only generates master codes in Java. C++ will be supported in next releases.

Before generating and compiling the master codes, a check of the graph consistency is always performed in order to ensure that the analyzed co-simulation graph is valid. Otherwise, an error message appears and the process is interrupted. If the “co-initialization” option is selected (see 4.5.2.1), the dependency graph of the calculation graph is automatically calculated before performing the generation and compilation.

4.6.1 Code generation

The generation of the Java master files associated to a calculation graph is directly achievable from the toolbar with the button “Generate Java master files”.

The codes are automatically generated with the Acceleo plug-in. All the required resources are taken from the configured DACCOSIM library folder (absolute path referencing the “DACCOSIM_deployment_setup” sub-folder, as seen in 4.5.2.4).

The folder that will contain the generated Java master files is named as the calculation graph filename excluding its extension.

At least one Java source file is generated as the global master file. This file is named DACCOSIMGlobalMaster.java. When the cluster associated to the calculation graph is a multi-nodes one, additional Java source files are also generated, one file per virtual node of the cluster exploited by the calculation graph besides the main node hosting the global master. These files are named DACCOSIMLocalMaster<i>.java, <i> being an index automatically supplied at the generation stage.
The code generation process also copies all the FMU blocks involved in the co-simulation into a sub-folder named resources and located below the code folder. This sub-folder also contains .csv files named <host>inputConnections.csv and <host>outputConnections.csv respectively gathering all the inputs and outputs for all FMUs on each node <host>. If only one node is involved, <host> is localhost.

⚠️ It is important to save the calculation graph under edition in the DACCOSIM GUI before launching the master code generation otherwise all the modifications carried out in between may not be considered in the generated files.

### 4.6.2 Code compilation

The compilation of all the generated Java master files associated to the current calculation graph is implicitly achieved within the code generation process launched by the button “Generate Java master files”.

The compiled master files are then located in the same folder than the Java source files.

The Java code generation and compilation automatically performs a checking of the graph consistency and the dependency graph calculation if not already done.

### 4.7 Managing a co-simulation on the localhost

With DACCOSIM 2017, managing a co-simulation via the GUI is restricted to cases where only one cluster node is considered, i.e. a single local machine.

#### 4.7.1 Running a co-simulation

Running a DACCOSIM application on the localhost is achievable from the toolbar with the button “Run Java master files”.

![Figure 58: How to run a “localhost” co-simulation](image)

The results of the co-simulation can be found into a sub-folder named results located below the code folder. Each FMU block generates results in a .csv file whose base name contains the FMU name prefixed with block_ (e.g. block_Genbui3.csv for results from the FMU block named Genbui3).

A block_ExternalWorld.csv file is also generated for the external environment block referencing all the external variables defined in the calculation graph.

The .csv files of all the blocks hosted on the same calculation node are then merged into a single .csv file named vnode_<host>.csv. Note that <host> is localhost when the default calculation resource is used.

Furthermore, a log file is generated for each block and node involved in the co-simulation execution. These .log files have the same base as the corresponding .csv files. As shown on the next figure, each log file contains some system information especially the number of rollbacks, successful steps & global iterations and the calculation & communication time.
For a better comfort of use, running a co-simulation automatically performs operations such as the consistency check of the calculation graph, the dependency graph calculation, the Java master files generation and compilation.

### 4.7.2 Aborting a co-simulation

When a run is too long to execute it may be helpful to stop it without closing the Eclipse application. This feature is accessible from the toolbar with the button “Stop generation/simulation processes”.

![Diagram of co-simulation resources and result folders organization](image)
After a click on this button during a co-simulation execution, all the Java processes of the DACCOSIM application are properly closed. If needed, this stopping action is also possible before a run, during the Java code generation / compilation or a Matryoshka FMU generation.

4.8 DACCOSIM scripting

The DACCOSIM GUI is a user-friendly tool to graphically elaborate calculation graphs. However, when the number of blocks or the number of connections between them is too large, a graphical approach may become very tedious.

A DSL (Domain Specific Language) is now available within DACCOSIM in order to manage large co-simulations schemes both in single node and multi-nodes mode. This DACCOSIM DSL is referred to as DACCOSIM Script Language (DSL), and the DSL files use accordingly the extension .dsl.

A DSL script is a textual set of instructions able to automatically build or update DACCOSIM calculation graphs (.dcg file extension) and DACCOSIM clusters definitions (.dcl file extension) in substitution to the usual and graphical way to edit these files through the DACCOSIM GUI.

A DSL script can also define the main running parameters and launch a co-simulation.

![DACCOSIM Script Language](image)

**Figure 62: Calculation graph automatic generation from a DSL script**

The DSL commands available within DACCOSIM 2017 are listed in different arrays in the following. It is possible to script the major part of the DACCOSIM commands including the setting of the main parameters from the “Simulation setup” window tabs.

4.8.1 Comment a DSL script

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| // This is a comment  
addNode: Node1; 4000;50; // This is another comment | Insert a comment line  
Or comment all the end of a line |

4.8.2 Cluster management

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
</table>
| createCluster: <cluster absolute path>;  
or createCL: <cluster absolute path>; | Create new cluster description. |
| loadCluster: <cluster absolute path>;  
or loadCL: <cluster absolute path>; | Load an existing cluster description. The file associated to this description should be edited. |
importCluster: <imported cluster absolute path>; or importCL: <imported cluster absolute path>; | Import an existing cluster description in the current cluster description.

addNode: <node name>; <node port>; | Add a cluster node to the current description (node name and port have to be provided).

removeNode: <node name>; | Remove an existing node name from the current cluster description.

editNode: <node name>; <node port>; | Edit the characteristics of an existing cluster node (node name and port should be provided).

A createCluster or loadCluster command should always precede any command associated to the design of a DACCOSIM cluster.

⚠️ It is not possible to automatically run a co-simulation on a multi-nodes cluster configuration from a DSL script with DACCOSIM 2017. Nevertheless, cluster management commands are available to prepare a distributed execution (refer to chapter 5).

### 4.8.3 DACCOSIM calculation graph management

#### 4.8.3.1 Blocks

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createGraph: &lt;calculationgraph absolute path&gt;; or createCG: &lt;calculationgraph absolute path&gt;;</td>
<td>Create new calculation graph description.</td>
</tr>
<tr>
<td>loadGraph: &lt;calculationgraph absolute path&gt;; or loadCG: &lt;calculationgraph absolute path&gt;;</td>
<td>Load an existing calculation graph description. The file associated to this description should be edited.</td>
</tr>
<tr>
<td>importGraph: &lt; imported graph absolute path&gt;; or importCG: &lt; imported graph absolute path&gt;;</td>
<td>Import an existing calculation graph description in the current calculation graph description.</td>
</tr>
<tr>
<td>addFMU: &lt;block name&gt;; &lt;FMU absolute path&gt;; addFMU: &lt;block name&gt;; &lt;FMU absolute path&gt;; &lt;x&gt;; &lt;y&gt;;</td>
<td>Add a FMU in the calculation graph. Block name and path to the .fmu file should be provided. Position (x, y) in the graph are optional arguments.</td>
</tr>
<tr>
<td>removeBlock: &lt;block name&gt;;</td>
<td>Remove a block (FMU or operator) with its connectors from the calculation graph.</td>
</tr>
</tbody>
</table>

A createGraph or loadGraph command should always precede any command associated to the design of a DACCOSIM calculation graph.

⚠️ Note that DACCOSIM 2017 does not provide commands to manipulate factory blocks (non-FMU blocks).

#### 4.8.3.2 Inter-blocs

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addLink: &lt;source block name&gt;; &lt;target block name&gt;; {&lt;output var list&gt;}; {&lt;input var list&gt;}; e.g.: addLink: block1; block2; {“a1”, “a2”, “a3”}; {“b1”, “b2”, “b3”};</td>
<td>Create a communication channel between two existing blocks of the graph.</td>
</tr>
<tr>
<td>updateLink: &lt;source block name&gt;; &lt;target block name&gt;; {&lt;output var list&gt;}; {&lt;input var list&gt;}; e.g.: updateLink: block1; block2; {“a1”, “a2”, “a3”}; {“b1”, “b2”, “b3”};</td>
<td>Update the variables exchanged on a communication channel between two existing blocks of the graph.</td>
</tr>
</tbody>
</table>
removeLink: <source block name>; <target block name>; Remove a communication channel existing between two blocks of the graph and the associated connectors.

### 4.8.3.3 External variables

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addEI: &lt;El name&gt;; &lt;type&gt;; &lt;variability&gt;;</td>
<td>Create an external input of the calculation graph:</td>
</tr>
<tr>
<td></td>
<td>• Type argument can be “Real”, “Integer”, “Boolean”, “String”, “Enumeration”;</td>
</tr>
<tr>
<td></td>
<td>• Variability argument can be “discrete” or “continuous”.</td>
</tr>
<tr>
<td>deleteEI: &lt;El name&gt;;</td>
<td>Delete an external input from the calculation graph.</td>
</tr>
<tr>
<td>addEO: &lt;EO name&gt;; &lt;type&gt;; &lt;variability&gt;; &lt;init_mode&gt;;</td>
<td>Create an external output of the calculation graph:</td>
</tr>
<tr>
<td></td>
<td>• Type argument can be “Real”, “Integer”, “Boolean”, “String” or “Enumeration”;</td>
</tr>
<tr>
<td></td>
<td>• Variability argument can be “constant”, “discrete” or “continuous”;</td>
</tr>
<tr>
<td></td>
<td>• Init mode argument can be “exact”, “approximated” or “calculated”.</td>
</tr>
<tr>
<td>deleteEO: &lt;EO name&gt;;</td>
<td>Delete an external output from the calculation graph.</td>
</tr>
</tbody>
</table>

### 4.8.3.4 External connectors

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addEIC: &lt;block name&gt;;</td>
<td>Add an external input connector to the graph.</td>
</tr>
<tr>
<td>addEIC: &lt;block name&gt;; &lt;x&gt;; &lt;y&gt;;</td>
<td>Position (x, y) in the graph are optional arguments.</td>
</tr>
<tr>
<td>removeEIC: &lt;block name&gt;;</td>
<td>Remove an external input connector from the graph.</td>
</tr>
<tr>
<td>addEOC: &lt;block name&gt;;</td>
<td>Add an external output connector to the graph.</td>
</tr>
<tr>
<td>addEOC: &lt;block name&gt;; &lt;x&gt;; &lt;y&gt;;</td>
<td>Position (x, y) in the graph are optional arguments.</td>
</tr>
<tr>
<td>removeEOC: &lt;block name&gt;;</td>
<td>Remove an external output connector from the graph.</td>
</tr>
</tbody>
</table>

### 4.8.3.5 External links

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addFromLink: &lt;EIC&gt;; &lt;block name&gt;; {&lt;external input var list&gt;}; {&lt;input var list&gt;};</td>
<td>Create a communication channel between an external input connector and an existing block of the graph.</td>
</tr>
<tr>
<td>updateFromLink: &lt;EIC&gt;; &lt;block name&gt;; {&lt;external input var list&gt;}; {&lt;input var list&gt;};</td>
<td>Update the variables exchanged on a communication channel between an EIC and an existing block.</td>
</tr>
<tr>
<td>removeFromLink: &lt;EIC&gt;; &lt;block name&gt;;</td>
<td>Remove a communication channel existing between an EIC and an existing block.</td>
</tr>
<tr>
<td>addToLink: &lt;block name&gt;; &lt;EOC&gt;; {&lt;output var list&gt;}; {&lt;external input var list&gt;};</td>
<td>Create a communication channel between an existing block of the graph and an external output connector.</td>
</tr>
<tr>
<td>updateToLink: &lt;block name&gt;; &lt;EOC&gt;; {&lt;output var list&gt;}; {&lt;external input var list&gt;};</td>
<td>Update the variables exchanged on a communication channel between an existing block and an EOC.</td>
</tr>
<tr>
<td>removeToLink: &lt;block name&gt;; &lt;EOC&gt;;</td>
<td>Remove a communication channel existing between an existing block and an EOC.</td>
</tr>
</tbody>
</table>
### 4.8.3.6 Associations and resources management

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>associateCluster: &lt;cluster absolute path&gt;;</td>
<td>Associate the cluster description defined in the selected file to the current calculation graph representation.</td>
</tr>
<tr>
<td>setGlobalMasterNode: &lt;node name&gt;;</td>
<td>Define the node where the global master will be hosted. By default, the first node of the associated cluster is used.</td>
</tr>
<tr>
<td>setBlock: &lt;block name&gt;; &lt;node name&gt;; &lt;port&gt;;</td>
<td>Define the node and the port associated to the wrapper of an existing block.</td>
</tr>
<tr>
<td>setBlockNode: &lt;block name&gt;; &lt;node name&gt;;</td>
<td>Define the node associated to the wrapper of a block.</td>
</tr>
<tr>
<td>setBlockPort: &lt;block name&gt;; &lt;port&gt;;</td>
<td>Define the port used by the wrapper of a block.</td>
</tr>
</tbody>
</table>

### 4.8.4 Variables initialization

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setBlockVariables: &lt;block name&gt;; {&lt;var list&gt;}; {&lt;value list&gt;};</td>
<td>Set initial value to a specified list of variables.</td>
</tr>
</tbody>
</table>

### 4.8.5 Variables logging

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logVariables :&lt;block name&gt;; true; {&lt;var list&gt;};</td>
<td>Add variables to the list of variables to store.</td>
</tr>
<tr>
<td>logVariables :&lt;block name&gt;; false; {&lt;var list&gt;};</td>
<td>Remove variables to the list of variables to store.</td>
</tr>
</tbody>
</table>

### 4.8.6 Tolerance setting

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setFMUSolverTolerance: &lt;block name&gt;; &lt;value&gt;;</td>
<td>Define an absolute tolerance for the FMU solver (only for FMU blocks).</td>
</tr>
<tr>
<td>setFMURealOutputsTolerance: &lt;block name&gt;; &lt;value&gt;;</td>
<td>Define the same absolute tolerance to all outputs for triggering rollbacks (only for FMU blocks).</td>
</tr>
<tr>
<td>setFMURealOutputTolerance: &lt;block name&gt;; {&lt;output var list&gt;}; {&lt;value&gt;};</td>
<td>Define absolute tolerance to specified outputs for triggering rollbacks (only for FMU blocks). If true is added, all previously configured tolerance lists are kept.</td>
</tr>
<tr>
<td>e.g.: setFMURealOutputTolerance: wall; (&quot;Q_a&quot;, &quot;Q_b&quot;); {0.001, 0.0001};</td>
<td></td>
</tr>
</tbody>
</table>

### 4.8.7 Internal dependencies setting

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addInternalDependencies :&lt;block name&gt;; &lt;output var&gt;; {&lt;input var list&gt;};</td>
<td>Define block internal dependencies.</td>
</tr>
<tr>
<td>removeInternalDependencies :&lt;block name&gt;; &lt;output var&gt;; {&lt;input var list&gt;};</td>
<td>Remove block internal dependencies.</td>
</tr>
</tbody>
</table>
### 4.8.8 Log properties

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>areSimulationLogged: &lt;true/false&gt;;</td>
<td>Generate a log file for each block and at virtual node level. Default is false (no generation).</td>
</tr>
<tr>
<td>areResultsStored: &lt;true/false&gt;;</td>
<td>Generate a results file for each block and merge the results at node level. Default is true (generation).</td>
</tr>
<tr>
<td>setCellSeparator: &lt;semicolon/comma/&gt;;</td>
<td>Define the cell separator for results files. Default is “comma”.</td>
</tr>
<tr>
<td>setDecimalSeparator: &lt;comma/point&gt;;;</td>
<td>Define the decimal mark for results files. Default is “point”.</td>
</tr>
<tr>
<td>convertBooleans: &lt;true/false&gt;;</td>
<td>Define a Boolean conversion from “true” and “false” respectively into “1” and “0” in results files. Default value is true (conversion).</td>
</tr>
</tbody>
</table>

### 4.8.9 Configuration of the co-initialization

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setCoInitMethod: &lt;no_coinit/coinit&gt;;;</td>
<td>Indicate if a co-initialization is required:</td>
</tr>
<tr>
<td></td>
<td>• coinit (co-initialization)</td>
</tr>
<tr>
<td></td>
<td>• no_coinit (no co-initialization)</td>
</tr>
<tr>
<td></td>
<td>Default value is coinit.</td>
</tr>
<tr>
<td>setCoInitMaxIterations: &lt;iterations&gt;;;</td>
<td>Define the maximum number of steps for the co-initialization phase when a co-initialization is required. Default value is 100.</td>
</tr>
<tr>
<td>setCoInitToleranceOnSCC: &lt;tolerance&gt;;;</td>
<td>Define the tolerance for the Newton-Raphson algorithm used to calculate initial input values to propagate in each SCC of the dependency graph when a co-initialization is required. Default value is $10^{-5}$.</td>
</tr>
</tbody>
</table>

### 4.8.10 Configuration of the co-simulation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setSimulationDuration: &lt;duration&gt;;;</td>
<td>Define the simulation duration (unit is s).</td>
</tr>
<tr>
<td>setSimulationTimeStep: &lt;duration&gt;;;</td>
<td>Define the reference time step of the simulation (unit is s). It is the time step for constant time step simulations, and the initial time step for adaptive time step simulations.</td>
</tr>
<tr>
<td>setSimulationMinTimeStep: &lt;duration&gt;;;</td>
<td>Define the minimum time step of the simulation (only useful for adaptive time step simulations).</td>
</tr>
<tr>
<td>setSimulationMaxTimeStep: &lt;duration&gt;;;</td>
<td>Define the maximum time step of the simulation (only useful for adaptive time step simulations).</td>
</tr>
<tr>
<td>setSimulationStepSizeControl:</td>
<td>Define the time step control method used in the simulation:</td>
</tr>
<tr>
<td>Constant/Euler/Richardson/Adams&gt;;;</td>
<td>• Constant: simulation using the reference time step</td>
</tr>
<tr>
<td></td>
<td>• Euler: time step adaptation using the Euler method</td>
</tr>
<tr>
<td></td>
<td>• Richardson: time step adaptation using the Richardson method</td>
</tr>
<tr>
<td></td>
<td>• Adams: time step adaptation using the Adams-Bashforth method</td>
</tr>
<tr>
<td></td>
<td>Default value is Constant.</td>
</tr>
</tbody>
</table>
setSimulationOrchestration: <Overlapped / Ordered>; Define the orchestration method used for the simulation:
- Overlapped/ovl: simultaneous execution of control and data transmission tasks
- Ordered/ord: sequential execution of the control task and the data transmission task
Overlapped is the default value with constant time step control method.
Ordered is the default value with variable time step control method.

setIntraNodeCommunication: <0mq,inproc>; Communication protocol to use inside each node among:
- 0mq: 0MQ protocol
- inproc: shared queue protocol
Default value is 0mq.

4.8.11 Configuration of the code generation

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setLibraryPath: &lt;absolute path&gt;;</td>
<td>Define the path of the DACCOSIM execution library.</td>
</tr>
<tr>
<td>setFMUPath: &lt;absolute path&gt;;</td>
<td>Define the path where the FMUs to import are located.</td>
</tr>
<tr>
<td>setOutputPath: &lt;absolute path&gt;;</td>
<td>Define the path for the DACCOSIM outputs (FMUs, Java files, simulation results).</td>
</tr>
<tr>
<td>checkGraphConsistency;</td>
<td>Checking of the graph consistency. No check when the command is absent.</td>
</tr>
<tr>
<td>buildGraph: &lt;true/false&gt;;</td>
<td>Build the causality graph and display it if argument is true. Default value is false.</td>
</tr>
<tr>
<td>copyFMUsfromGraph;</td>
<td>Gather and copy the FMUs present in the graph into a single folder. By default, the FMUs are copied into the folder specified by the setOutputPath command.</td>
</tr>
<tr>
<td>copyFMUsfromGraph: &lt;absolute path&gt;;</td>
<td></td>
</tr>
<tr>
<td>generateJavaMasters;</td>
<td>Build the causality graph and generate the associated Java master files into the specified folder. By default, the Java files are generated into the folder given by the path specified by the setOutputPath command.</td>
</tr>
<tr>
<td>generateJavaMasters: &lt;absolute path&gt;;</td>
<td></td>
</tr>
<tr>
<td>compileJavaMasters;</td>
<td>Compile the Java master files previously generated for the current calculation graph. By default, the Java files are compiled into the folder given by the path specified by the setOutputPath command.</td>
</tr>
<tr>
<td>compileJavaMasters: &lt;absolute path&gt;;</td>
<td></td>
</tr>
<tr>
<td>runJavaMasters;</td>
<td>Execute the Java master previously generated for the current calculation graph under the condition all the blocks are on the same cluster node or local machine. By default, the results are copied into the folder results in the path specified by the setOutputPath command.</td>
</tr>
<tr>
<td>runJavaMasters: &lt;absolute path&gt;;</td>
<td></td>
</tr>
</tbody>
</table>

4.8.12 Real time visualization

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>displayResults: true; {&lt;chart nb&gt;,&lt;block name&gt;,&lt;var name&gt;};</td>
<td>Define variables to draw on different graphs for real time visualization. This command has to be run before the runJavaMasters command to allow Real Time display.</td>
</tr>
</tbody>
</table>
Define variables to draw on different graphs for post-mortem visualization. This command has to be run after the runJavaMasters command.

In both case:
- each graph is identified by a number starting from 1
- block names must be given as string (with quotes)
- “dt” is a system variable denoting the simulation time. In this case, the block name is “” by convention.

⚠️ It is not possible to run a DSL script out of the GUI with DACCOSIM 2017.

4.8.13 Loading a DSL script

DSL scripts can be prepared with any textual editor. Accepted extensions for these files are .dsl and .txt.

Then, a DSL script can be selected and executed directly from the toolbar with the button “Load DACCOSIM script”.

⚠️ It is not possible to export a DSL script when the calculation graph contains factory blocks with DACCOSIM 2017.
4.8.15 Example of DSL script

Here is the DSL script illustrating the case tankBarrel (refer to chapter 7).

```plaintext
// This script automatically creates and runs a co-simulation on a use case named "tankBarrel"
// The calculation graph is composed with 4 FMUs exported from Dymola 2017
// The run is on a single node machine (localhost is defined as the default calculation resource)

// Create a DACCOSIM calculation graph
createGraph: C:\DACCOSIM\user-guide-use-cases\tankBarrel\Dymola-2017\tankBarrel\tankBarrel.dcg;
// Create blocks & external variables
addFMU: C:\DACCOSIM\user-guide-use-cases\tankBarrel\Dymola-2017\TankWin3264.fmu; Tank; 160; 180;
addFMU: C:\DACCOSIM\user-guide-use-cases\tankBarrel\Dymola-2017\BarrelWin3264.fmu; Barrel; 350; 180;
addFMU: C:\DACCOSIM\user-guide-use-cases\tankBarrel\Dymola-2017\C1Win3264.fmu; C1; 50; 50;
addFMU: C:\DACCOSIM\user-guide-use-cases\tankBarrel\Dymola-2017\C2Win3264.fmu; C2; 240; 50;
addEO: water; Real; continuous; calculated;
addEO: water; 520; 190;
// Create connections
addLink: C1; Tank; {"Valve"}; {"Valve"};
addLink: C2; Barrel; {"Abort","Size"}; {"Abort","Size"};
addLink: Tank; Barrel; {"outFlow"}; {"inFlow"};
addToLink: Barrel; water; {"Water"}; {"water"};
// Simulation properties
areSimulationLogged: true;
setCellSeparator: comma;
convertBooleans: true;
setSimulationDuration: 18;
setSimulationTimeStep: 0.01;
setSimulationOrchestration: Overlapped;
// Generation
setLibraryPath: C:\DACCOSIM\DACCOSIM2017\branches\Matryoshka\DACCOSIM_deployment_setup;
setOutputPath: C:\DACCOSIM\user-guide-use-cases\tankBarrel\Dymola-2017\tankBarrel;
checkGraphConsistency;
copyFMUsFromGraph;
generateJavaMasters;
compileJavaMasters;
// Visualization & run
displayResults: true;
runJavaMasters;
```

Figure 65: A DSL script example (use case tankBarrel supplied in the distribution)

4.9 DACCOSIM as a Matryoshka FMU

DACCOSIM is able to export a calculation graph as a parallel FMU (what is called here a Matryoshka).

The procedure to build a “superFMU” requires only few additional steps after having designed a valid co-simulation graph with at least one external input or one external output. During this initial stage, the co-simulation must be configured as for any test case. These settings determine the internal behavior of the Matryoshka, with in particular:

- The co-initialization mode;
- The step size control method including its step size characteristics;
- The event detection method.

Then the inputs and outputs of the Matryoshka FMU must be defined and linked to the internal FMUs variables which implies:

- To create external variables and set for each a name, a causality (input or output), a type, a variability (constant, discrete or continuous) and an initialization mode (exact, approximated or calculated);
- To define default initial values for each external input;
- To add external connectors, connect them to the FMUs connectors in the graph and associate their variables;
- To generate the Matryoshka FMU by a one click button from the toolbar.
4.9.1 Internal dependencies

When a valid calculation graph including both external inputs and outputs must be exported as a superFMU, the dependencies of external outputs regarding its external inputs must be first specified.

By default, dependencies are automatically computed according to the causality graph of the calculation graph. However, these default dependencies can be modified thanks to the button “Define external outputs’ dependencies to external inputs”.

![Figure 66: Defining internal dependencies for a “superFMU”](image)

After clicking this button, an interface consisting of three list boxes appears:

- The list box “External outputs” on the left allows to select one defined external output;
- The list box “External inputs” in the middle allows to select defined external inputs;
- The list box “O/I Dependency” on the right (matching list box) is passive and displays the dependencies already done.

By default, these three list boxes are prefilled according to dependencies deduced from the causality graph of the calculation graph.

The button “->” associates a selection of external inputs with the current external output.

The button “<” disassociates a selection of external inputs from the current external output.

The button “Full Dependencies” sets dependencies with the assumption that each external output depends on all external inputs.

The button “Compute Dependencies” recalculates dependencies according to the causality graph of the calculation graph.

The button “Clear Dependencies” clears the list box “O/I Dependency”.

![Figure 67: Tuning of internal dependencies within a “superFMU”](image)

4.9.2 Exporting an FMU

Once the internal dependencies have been set (either automatically or by a user-defined action), the button “Create DACCOSIM Matryoshka FMU” is used to export a single node calculation graph as a powerful multi-threaded FMU.

![Figure 68: Exporting a DACCOSIM application as a “superFMU”](image)
A user interface appears allowing to select the execution operating system to consider for the generated FMU. It can be either Windows 64-bit or Linux 64-bit. By default, the operating system of the hosting machine is proposed in this interface. Selecting “Cancel” means exiting the generation process.

The FMU is exported with a base name corresponding to the calculation graph name (e.g. dy_tankBarrel.fmu). The modelDescription.xml file of this FMU is built with the following main attributes:

- generationTool="SIANI FMU Builder v2.4.3";
- author="V. Galtier and M. Ianotto";
- description="DACCOSIM Matryoshka FMU";
- fmiVersion="2.0";
- modelName="dy_tankBarrel" (derived from the calculation graph name).

In addition here are the exposed attributes of the element “CoSimulation”:

- providesDirectionalDerivative="false";
- canSerializeFMUstate="false";
- canGetAndSetFMUstate="false";
- canBeInstantiatedOnlyOncePerProcess="true";
- canHandleVariableStepSize="true";
- needsExecutionTool="true".

The needed resources for a “superFMU” built as an export FMU from DACCOSIM are several files required for the co-simulation (with extensions .fmu, .dll, .jar, .csv).

⚠️ The Matryoshka is exported as a 64-bit Windows or 64-bit Linux FMU. No 32-bit architecture is supported. This also supposes that only 64-bit FMU can compose the internal calculation graph.

4.10 Online help

A button “Help” is defined on the toolbar to deliver some information about the version of the DACCOSIM distribution.
The displayed window offers a link to the DACCOSIM 2017 website (https://daccosim.foundry.supelec.fr/) where the “DACCOSIM 2017 user’s guide”, the “DacRun user’s guide and technical manual” and some publications can be downloaded. Some videos can also be seen in streaming mode.

Figure 71: Help information
5 Deploying and launching a co-simulation on a cluster

The distributed master architecture of DACCOSIM enables executions on distributed calculation nodes (PC clusters). This allows to take advantage of additional scalability and to improve the performance of heavy computation co-simulations.

Deploying a large scenario on a cluster becomes quickly complex if performed manually. Fortunately, a Python software called DacRun has been designed to easily compile, run and collect results for a co-simulation application prepared by DACCOSIM.

5.1 What is DacRun?

DacRun is a Parallel Python deployment tool provided with DACCOSIM 2017 in order to simplify and automate the deployment operations, and in particular:

- Mapping the resources, i.e. which virtual node corresponds to which physical node of the cluster;
- Compiling the DACCOSIM Java master files knowing that exactly one JVM (Java Virtual Machine) running one Java program is installed to support one virtual node;
- Launching the execution of distributed co-simulation cases;
- Collecting the simulation result and log files generated on each node and merging the results node by node into a synthesis file;
- Cleaning the cluster nodes at the end of the co-simulation.

The whole deployment and results collection process is illustrated above.

DacRun is freely distributed with DACCOSIM 2017 into the folder fr.daccosim.dacrun. It is mostly designed for PC cluster architecture but can also be run on a local machine.

DacRun is basically compliant with OAR and SLURM cluster managers. However when used on a single machine, no cluster manager is required.

⚠️ The resources of the cluster have first to be allocated before launching DacRun.

DacRun achieves many checks on operations run on the computing nodes, and can generate detailed logs and error reports, which is very helpful to debug a distributed execution and when porting DACCOSIM on a new computing plateform.
DacRun allows some control of the collection and aggregation of distributed execution logs and results. This functionality is still under development.

DacRun is developed under Python 2.7, and requires the installation of ppserver.py package on all used computing nodes.

For more information about DacRun, especially for syntax, options and examples, please refer to the “DacRun user’s guide and technical manual” included in the folder Documentation of the DACCOSIM 2017 distribution.

5.2 DacRun command line options

A wide set of options are automatically added by DACCOSIM 2017 to the generated Java master files. These options are available as command line options for DacRun. These command options are listed and detailed in the “DacRun user’s guide and technical manual”.

```
dacrun.py
    # Mandatory path options
    -p PROJECT
    -fp FMUFOLDERPATH -lp LIBRARYPATH
    -x BLOCKRESULTS -b BLOCKLOGS -v1 VIRTUALNODELOGS
    # Automatic creation of missing directories
    [-mkdir]
    # Co-simulation computation setting
    [-m METHOD]
    [-e EXTRAPOLATION]
    [-iss INITIALSTEP] [-iss MINIMALSTEP] [-maxs MAXIMALSTEP]
    [-sd SIMULATIONDURATION]
    # Co-simulation execution setting
    [-ic INTERNALCOMMUNICATION]
    [-cm ORCHESTRATIONMODE]
    [-noresult] [-nolog] [-noio]
    # Adaptation to the cluster manager used
    [-cm CLUSTERMANAGER]
    # Information retrieval
    [--daccosimversion]
    [--dacrunversion]
    [-V VERBOISE]
    [-h]
    # Work in progress (do not use)
    [--cleanbefore]
    [-gp GATHERPATH] [-parallelgather PARALLELGATHER]
    [-zip] [-merge] [-erasetlog]
```

Figure 73: Overview of the DacRun command line options (refer to the ad-hoc documentation)

It is to be noted that some of these options can override the parameters initially defined in the “Simulation setup” interface of the DACCOSIM GUI (refer to § 4.5.6).

5.3 Nodes mapping strategies

When designing a co-simulation with the DACCOSIM GUI or a DSL script, a fixed number of virtual nodes is set and each FMU block is associated to one of these virtual nodes. Each virtual node is a JVM process, and FMUs associated to this node are threads inside the JVM of the node.

⚠️ More sophisticated mapping strategies including load balancing will be next available.
6 Post-mortem and real time visualization

The result values computed by DACCOSIM during a co-simulation are registered in .csv files.

A .csv result file is generated for each FMU involved in a DACCOSIM co-simulation process. Furthermore, an additional .csv file is built for each cluster node by merging the results from the FMUs belonging to this calculation resource as if only one “super FMU” was running on the node.

The two first columns of each .csv file are always reserved to the system variables ‘time’ and ‘dt’ (Δt).

6.1 Post-mortem visualization outside DACCOSIM

These .csv files can be read after run by external tools like spreadsheet programs.

6.1.1 Post-mortem visualization after a single node run

Depending on the configured decimal mark and cell separator, result files can be read by external tools like Microsoft Excel knowing that conventions are depending on languages and countries. For example, in France, Excels is often configured with the “,” as cell separator and “.” as decimal mark.

A nice possibility is given by the “OpenModelica Connection Editor” where .csv files can be read using the “Open Result File(s)” command in the “File” menu of OMEdit, the graphical model editor / browser of OpenModelica. The only conditions for that are (refer to § 4.5.6.2):

- The decimal mark is configured as “.”;
- The cell separator is configured as “,”;
- The “convert Boolean into integers” checkbox is checked.

The plot windows can easily be parameterized thanks to the “Variables Browser” as variables are structured in the form of a tree, each variable having a checkbox. Ticking the checkbox will plot the variable values. There is a find box on the top for filtering the variable in the tree. The filtering can be done using regular expression, wildcard and fixed string. The complete “Variables Browser” can be collapsed and expanded using the “Collapse All” and “Expand All” buttons.

![Figure 74: DACCOSIM results files variables plotted with OpenModelica](image)

This plotting mechanism is also very helpful when variables in result files must be compared with some others files with or without the same time series coming from other tools like Dymola, often seen as a reference in modeling and simulation.
6.1.2 Post-mortem visualization after a multi-nodes run

The solution with OMEdit is smart also because it is possible to open and visualize several .csv files either in different plot windows or on the same graph. It is very interesting when several cluster nodes .csv files have to be simultaneously opened and variables from these different files have to be represented together.

6.2 Real time and post-mortem visualization within the DACCOSIM GUI

A dedicated GUI is now available in DACCOSIM 2017 for real time and post-mortem visualizations when the co-simulation is done on a single-node machine.

6.2.1 Real time visualization

A new graphical environment dedicated to simulation and visualization can be launched from the toolbar with the button ‘Display results’.

![Figure 75: Launching a simulation with visualization GUI](image)

This GUI is a complete interface to set and run co-simulations, and visualize state variables.

![Figure 76: Overview of the GUI dedicated to simulation with visualization](image)

The toolbar available at the top of the main window contains different buttons.
On the left part of this dedicated toolbar, three buttons allow the parameterization and the control of co-simulation:

- The “Edit use case settings” button has the same behavior than the “Edit project properties” button of the main toolbar;
- The “Run simulation use-case” button can launch a run like the “Run Java master file” button of the main toolbar;
- The “Stop simulation use-case” button can stop a run like the “Stop generation/simulation processes” button of the main toolbar.

On the right part of the toolbar, six buttons manage the visualization of graphs in the “Temporal display” tab:

- The “Fit content” button rescales the current graph so that all the curves are visible covering the simulation time;
- The “Zoom X Axis” allows to define a range of time [Min x limit, Max x limit] with values set by the user in accordance with the simulation time;
- The “Move X Axis fast backward” shift backward the current graph content by a large time step;
- The “Move X Axis progressively backward” shift backward the current graph content by a small time step;
- The “Move X Axis progressively forward” shift forward the current graph content by a small time step;
- The “Move X Axis fast forward” shift forward the current graph content by a large time step.

Real time display in the “Temporal display” tab is controlled by an interval of time between two consecutive points defined by the “RT graph update interval (ms)” parameter (possible values are 100, 250, 500 and 1000 ms).

Let’s now focus on the left part of the “DACCOSIM Results Display” window.

The “Variable list” contains the list of all the variables that have been declared as variables to store in the calculation graph.

After selection of one or several variables from this list, the “Add selected variables into a new graph” button builds a new graph dedicated to these variables.

Several graphs named Graph1, Graph2, etc. can be defined in this way.

Additional variables can then be added to one existing graph thanks to the “Add selected variables to an existing graph” button.

The “Displayed variables” recap the variables appearing in graphs with information such as:

- The graph where the variable is displayed (column Table);
- The FMU where the variable is defined (column Block);
- The variable name (column Output).
After selection of one or several variables from this list, the “Remove selected variables” button allows the removing of variables.

The “Clear all displayed graphs” button is useful to remove all the displayed graphs of the “Temporal display” tab.

The “Reset and wait for simulation” button clears all the graphs (without removing them) in order to launch again a run. The .csv files are also removed from the folder whose name is recalled in the “Folder under analysis” field.

The “Table display” tab is another view on the results where the information is displayed in a tabular form with a first column for the time series completed with all the selected variable series.

Real time display in the “Table display” tab is controlled by an interval of time between two consecutive points defined by the “RT file update interval (ms)” parameter (possible values are 1, 5, 10, 15, 20, 25, 50 and 100 ms).

6.2.2 Post-mortem visualization

Post-mortem visualization is possible via the button “Load simulation result” to reload the full content of the simulation result files available in the folder whose name is recalled in the “Folder under analysis” field.

It is recommended to use this button after a co-simulation end in order to get maximum accuracy of results display.
7 Use cases

Several cases are supplied with the DACCOSIM 2017 delivery in the folder Examples. Some are trivial examples while others are more business-oriented use cases.

All the referenced FMUs are license-free and have been built using Dymola, ControlBuild or Papyrus in order to be executed on Windows or Linux environment. Without exception, all the Modelica models are also supplied.

To try these use cases, adapt the given DSL scripts or use the DACCOSIM GUI according to the information given in the following.

All the DSL scripts correctly run when the folder user-guide-use-cases contained in the folder Examples is copied into a folder named C:\DACCOSIM\ on the host machine.

Note each FMU name contains information on the OS used to export it; e.g. tankwin3264.fmu is able to run on a 32 or 64-bit Windows machine while the same FMU exported from Linux is named tanklin3264.fmu.

7.1 Co-initialization examples

This paragraph details two co-initialization examples available with the DACCOSIM 2017 distribution.

7.1.1 tripleEquation case

This example illustrates the co-initialization of a system composed with three coupled FMUs (intersection between three planes):

- plane1 model calculates $z$ from $x$ and $y$ according to the equation: $6x - y + z = 30$;
- plane2 model calculates $y$ from $x$ and $z$ according to the equation: $2x + 2y - 3z = 27$;
- plane3 model calculates $x$ from $y$ and $z$ according to the equation: $x - 3y + 2z = -9$.

To run this example, you have to adapt the given script tripleEquation.dsl to your processing context.

7.1.1.1 Graphs and simulation properties

Here are the calculation graph and the dependency graph for the tripleEquation case:

![Figure 77: Calculation graph and dependency graph for the tripleEquation case](image)

7.1.1.2 Discussion on the results

The vnode_localhost.csv file shows the calculated coordinates of the three planes intersection point: $(x, y, z) \approx (6, 3, -3)$. 
7.1.2 equationPair case
This example illustrates the co-initialization of a system composed with two equations and two unknowns:

- equation1 model calculates $x_2$ from $x_1$ according to the equation: $2x_1^2 + 5x_2 = 42$;
- equation2 model calculates $x_1$ from $x_2$ according to the equation: $x_1 - 6x_2 = 4$.

To run this example, you have to adapt the given script equationPair.dsl to your processing context.

7.1.2.1 Graphs and simulation properties
Here are the calculation graph and the dependency graph for the equationPair case:

![Figure 78: Calculation graph and dependency graph for the equationPair case](image)

7.1.2.2 Discussion on the results
The vnode_localhost.csv file shows the calculated solution of the 2 equations and 2 unknowns system: $(x_1, x_2) \approx (4.56, 0.09)$.

7.2 Co-simulation examples
This paragraph details several co-simulation examples available with the DACCOSIM 2017 distribution.

7.2.1 tankYPipe case
This example illustrates an academic case with two FMUs exported from Dymola, one being duplicated three times. External input/output variables are also included to communicate with the environment.

Each tank (tank.mo) has a 2 $m^2$ floor area with a hole of 0.1 $m^2$ and an initial water level of 3 m. Tank1 and Tank2 each calculates their output flow rate with respectively an initial input flow rate of 0.5 $m^3$ and 0.1 $m^3$. They pour water in the Y pipe (ypipe.mo model) that simply adds the two input rates as an output flow rate entering into Tank3.

⚠ Tank1, Tank2 and Tank3 are three instance of the same Modelica model tank.mo with a low data processing weight. Several variants of this model are given (only for Windows) with dummy differential equations included to artificially increase the time calculation of the numerical integration. Of course all the results are exactly the same when the co-simulation is done replacing one or several instances of the small model by some of the variant models.

These model variants all have 40.000 dummy variables but the proportion of algebraic and differential equations depends on each model. For example tank15000.mo model contains $30.000 = 15.000 \times 2$ differential equations and $10.000 = 40.000 - 30.000$ algebraic ones.

To run this example, you have to adapt the given script tankYPipe.dsl to your processing context.
7.2.1.1 Graphs and simulation properties

Here are the calculation graph and the dependency graph for the tankYPipe case:

![Diagram]

Figure 79: Calculation graph and dependency graph for the tankYPipe case

The chosen co-simulation parameters are the following:

- \textit{start time} = 0 \text{ s}, \textit{stop time} = 100 \text{ s};
- \textit{constant step size} = 0.5 \text{ s}.

7.2.1.2 Discussion on the results

As shown below, a co-simulation with a constant step size equal to 0.5 s is sufficient for this case.

![Graph]

Figure 80: Results for the academic tankYPipe case
7.2.2 events case

This example illustrates the capability of DACCOSIM to simulate several independent models. Each model contains different type of events:

- M1.mo model is the famous bouncing ball model with eight model state events;
- M2.mo model contains two model state events;
- M3.mo model three model time events.

To run this example, you have to adapt the given script `events.dsl` to your processing context.

7.2.2.1 Graphs and simulation properties

Here is the calculation graph for the events case (the dependency graph is empty as there are no links between FMUs):

![Calculation graph for the events case](image)

The chosen co-simulation parameters are the following:

- \( start \text{ time} = 0 \, s, \, stop \text{ time} = 10 \, s; \)
- \( constant \text{ step size} = 0.01 \, s. \)

7.2.2.2 Discussion on the results

The visualization shows all the events are correctly detected with a small enough constant step size.

![Results for the events case](image)
7.2.3 tankBarrel case

This example illustrates a hybrid co-simulation mixing control and physics and it is very helpful to underline some limits of the current FMI-CS 2.0 release of the FMI standard.

In addition, this example shows the DACCOSIM capability to exhibit FMUs exported from heterogeneous tools (Dymola, ControlBuild or Papyrus) and whose models are built with different modelling languages (Modelica, IEC 61131-3 or UML/SysML) for the control part of the case.

To run this example, you have to adapt one of the given scripts to your processing context:

- `dy_tankBarrel.dsl` script uses 4 FMUs all exported from Dymola;
- `dy_cb_tankBarrel.dsl` script mixes 2 FMUs exported from Dymola and 2 FMUs exported from ControlBuild;
- `dy_pa_tankBarrel.dsl` script mixes 2 FMUs exported from Dymola and 2 FMUs exported from Papyrus.

⚠️ For more information on the tankBarrel case, please refer to the paper “Hybrid Co-Simulation of FMUs using DEV&DESS in MECSYCO”, in Proceedings of the 2016 Spring Simulation Multiconference (TMS/DEVS’16), 2016 from B. Camus, V. Galtier, and M. Caujolle.

7.2.3.1 Graphs and simulation properties

Here are the calculation graph and the dependency graph for the tankBarrel case:

![Calculation graph and dependency graph for the tankBarrel case](image)

The chosen co-simulation parameters are the following:

- **start time** = 0 s, **stop time** = 18 s;
- **constant step size** = 0.5 s (first trial) and then 0.01 s (second trial).

7.2.3.2 Discussion on the results

A co-simulation with a constant step size equal to 0.5 s shows that events are not all detected in time. As an example with the c1.valve variable:

- First rising edge (theoretically at 0.5 s) is correctly detected as this time exactly corresponds to the end of a calculation step;
- Falling edge (theoretically at 3.2 s) is only detected at 3.5 s as a multiple of the constant step size;
- Second rising edge (theoretically at 5.2 s) is only detected at 5.5 s as a multiple of the constant step size.

A second trial with a constant step size equal to 0.01 s mitigates the error but events are still of course not correctly detected. The results are in conformance with the expectations.
The impact of a step size reduction on the accuracy is easily visible with OpenModelica where several .csv file can be visualized on the same diagram.

Figure 84: Main awaited signals for the tankBarrel case with a constant step size of 0.01 s

Figure 85: Inaccuracy in event detection on signal c1.valve with constant step sizes (0.5 s in red, 0.01 s in blue)

⚠️ Exact events handling is under consideration in the FMI Project managed by the Modelica Association. In advance, some improvements are unofficially being tested by a working group composed with Dassault Systèmes, EDF, CentraleSupélec and SIANI.
7.2.4 simpleHeatTransfer school case

The purpose of this school case is to illustrate a variable step size with rollbacks on a 4-FMU calculation graph composed with two thermal zone subject to external conditions and separated by a wall.

Each thermal zone is an assembly room in a building. The modeling takes into account the connection to boundary conditions (external temperature, internal gains ...), heat transfer through the outer walls and a thermal capacity representing evenly air volume. The outer walls are represented by a single equivalent wall.

Modeling a wall takes into account the convection of both sides as well as conduction and thermal inertia therein.

The functional goal is to check the temperature inside two rooms separated by a wall and subject to some external temperature fluctuations simply modeled as a sinus function.

To run this example, you have to adapt the given script simpleHeatTransfer.dsl to your processing context.

7.2.4.1 Graphs and simulation properties

Here are the calculation graph and the dependency graph for the simpleHeatTransfer case:

![Calculation graph and dependency graph for the simpleHeatTransfer case](image)

The chosen co-simulation parameters are the following:

- **start time** = 0 s, **stop time** = 86400 s;
- **stepsize control method** = Euler;
- **simulation orchestration** = Overlapped;
- **minimum step size** = 1 s and **maximum step size** = 100 s;
- **initial step size** = 10 s.

7.2.4.2 Discussion on the results

Among other information, the different log files (e.g. block_zoneA.log) indicate the number of rollbacks, the number of iterations, and the number of successful steps.

The obtained results seem realistic and in line with the expectations.
7.2.5 pWH case

The purpose of this case is to illustrate the real time visualization on a real business use case.

A zero sequence wattmetric protection has been developed to protect French MV neutral compensated distribution power grids against mean and medium resistive single phase faults. This protection - denoted PWH in French - is based on the analysis of both transient and steady state values of residual current and residual voltage available in the compensated MV substation.

PWH selective action is based on the known fact that the residual current in the faulty feeder is of the opposite direction and much greater than on any of the unfaulty feeders. Therefore, on the faulty feeder, the zero-sequence wattmetric relays detects a negative zero sequence active power. On contrary, the protection placed on an unfaulty feeder detects a positive zero sequence active power.

The PWH protection is efficient against two types of phase-to-ground faults:

- A permanent fault which involves residual current with a preponderant component at 50 Hz, after the transient phenomena due to the occurrence of the fault,
- A series of self-extinguishing faults, which is considered as the same fault.

In order to detect a permanent fault, the RMS values of the residual voltage & current and the residual active & reactive power are calculated for a cycle of 20 ms. For the two types of faults, residual power mean value is also calculated for a cycle of 60 ms. Depending on the results of these calculations, according to a condition test, a positive or negative residual power value is sent to the logical part of the protection which elaborates a trip protection message toward the circuit breaker of the faulty feeder.

The pWH case models the behavior of two PWH protections each located at the head of two feeders supplied by a HV/MV substation in a MV power grid. Each PWH device monitors the residual voltage and the residual current of its corresponding feeder using this information to control the opening of the feeder breaker in case of a downstream fault.

The system is composed with three supplied FMUs:

- Projet_0PWH_Pre_0FMUs_Reseau.fmu (the power grid block named Grid),
- Projet_0PWH_Pre_0FMUs_PWH_0F.fmu (the faulty feeder protection block named PWH_F),
- Projet_0PWH_Pre_0FMUs_PWH_0S.fmu (the healthy feeder protection block named PWH_S).

All the Modelica models have been developed with free components available in the Modelica standard library (version 3.2 rev2 from 30th July 2013), especially in the package Modelica.Electrical.Multiphase.
DACCOSIM 2017 User’s Guide

The power grid model simulates a single-phase fault at $t = 0.3 \, \text{s}$ on one of the two feeders. This default is detected by the two PWHs but only the protection with the downstream fault must open the circuit breaker at $t = 1.1 \, \text{s}$.

To run this example, you have to adapt the given script `pWH.dsl` to your processing context.

⚠️ The Modelica source models are not supplied as the PWH is the subject of a patent in France.

7.2.5.1 Graphs and simulation properties

Here are the calculation graph and the dependency graph for the pWH case:

The chosen co-simulation parameters are the following:

- **start time** = 0 s, **stop time** = 2 s;
- **constant step size** = 0.001 s.

7.2.5.2 Discussion on the results

Results are given within three graphs drawn by the new DACCOSIM 2017 real-time visualization tool. They are in conformance with the business expectations and a pure Modelica Dymola simulation.

Figure 88: Calculation graph and dependency graph for the pWH case

Figure 89: Residual current detected by each protection and only one breaker opening to remove the default
7.2.6 heavyHeatTransfer case

The purpose of this more sophisticated heat transfer use case is to illustrate the capability of DACCOSIM to handle a large number of FMUs. It is based on more sophisticated models built with the BuildSysPro library.

The BuildSysPro library is a free open-source Modelica library edited by EDF R&D under the Modelica License 2 (https://www.modelica.org/licenses/ModelicaLicense2).

This library is dedicated to building and energy systems modeling and is designed to be used in several contexts including building physics research, global performance evaluation, technology development and impact assessment. It is also a basis for urban and building stock simulation. BuildSysPro is intended for a relatively large audience ranging from R&D scientists to building services engineers.

BuildSysPro contains classes to describe the whole building and its energy systems including envelope components, HVAC systems and other energy conversion devices (DHW, thermal and photovoltaic panels…) and boundary conditions models. These classes are compliant with the Modelica.Thermal.HeatTransfer and Modelica.Media packages of the Modelica Standard Library to ensure a good level of interoperability with other Modelica libraries.

![Figure 90: Building envelope model and whole building / systems model created with BuildSysPro](image)

The heavyHeatTransfer case uses real temperature readings from a French city in the suburb of Paris (Trappes). Thermal zones are differentiated according to their geographical position (North or South) and their floor (intermediate, top, and bottom).

To run this example, you have to adapt the given scripts to your processing context:

- *heavyHeatTransfert11.dsl* with 11 FMUs;
- *heavyHeatTransfert101.dsl* with 101 FMUs;
7.2.6.1 Graphs and simulation properties

The chosen co-simulation parameters are the following:

- start time = 0 s, stop time = 86400 s;
- constant step size = 10 s.

7.2.6.2 Discussion on the results

Results are given within two graphs drawn by the new DACCOSIM 2017 real-time visualization tool. They are in conformance with the expectations according to business experts.

Exactly the same results are obtained with the script `heavyHeatTransfert101.dsl` including 101 FMUs.

7.2.7 iDEAS case

This use case is based on the “Open Integrated District Energy Assessment by Simulation” (OpenIDEAS) library offering transient simulation of thermal and electrical processes at neighborhood level. The IDEAS tool differs from existing building physics and electrical energy system based models by integrating the dynamics of the hydronic, thermal as well as electrical energy networks at both the building and aggregated level within a single model and solver.
The building model

The iDEAS case instantiates twelve times the same building with different parameters. Each building model consists of different zones which can be seen as rooms. Each zone is conditioned by a thermal transient simulation taking into account the climate, the heating and ventilation systems, the occupancy as well as an electrical consumption. A building consists of 3 different zones. For each zone the heat transfer for the outer surfaces (walls, roof, windows and ground) is calculated as the sum of the short wave radiation, long wave radiation, conductive and convective heat flows. The same type of heat flows are calculated for the heat transfer between the inner surfaces. To give an insight into the complexity of the model, the form of the equations used are listed below.

Short wave radiation

\[ q_{sw} = \varepsilon_{sw}(E_{Sd} + E_{SD}) \]

With \( E_{Sd} \) and \( E_{SD} \) being the diffuse and direct solar irradiation on the surface and \( \varepsilon_{sw} \) the shortwave absorption on the surface.

Long wave radiation - outer surfaces

\[ q_{lw} = \sigma \varepsilon_{lw}(T_s^4 - F_{ce} T_{ce}^4) - (1 - F_{ce})T_{db,e}^4 \]

Where \( T_s \) is the surface temperature, \( T_{ce} \) is the celestial dome temperature, \( T_{db,e} \) is the outdoor dry bulb temperature, \( \sigma \) is the Stefan-Boltzmann constant and \( \varepsilon_{lw} \) is the longwave emissivity of the surface. \( F_{ce} \) is a constant factor.

Long wave radiation - between the i and j inner surfaces

\[ q_{lw,i,j} = k (T_{si}^4 - T_{sj}^4) \]

Where \( T_{si} \) is the temperature of the i inner surface and \( k \) depends on the surfaces.

Conduction

The conduction into the walls is modelled as a one dimensional thermal circuit with thermal resistances and capacitances. The underlying equations are ordinary differential equations.

Convection

\[ q_{cv} = h_{cv} (T_{db} - T_s) \]

Where \( h_{cv} \) is the natural convective heat transfer coefficient that can depend of \( T_s, T_{db} \) is the dry bulb temperature and \( T_s \) is the surface temperature.

To solve those equations the model uses external data. Mainly climate conditions which are read from a text file.

Occupancy (stROBe)

The occupancy of the building is also modifying its energy consumption. The energy demand and comfort temperature needed by the occupants are input data which are loaded from a text file. The user of the model can choose the number of occupants in the building (ie the occupancy profile to load). The occupancy sets the setpoint temperature for the heating system and the electrical load due to the power consumption by the occupants’ activities (idOcc).
Heating system

The heating system modelled is an ideal heating system using radiators (no district hot water). The heating power is equal to the minimum between the 5000 \( W \) nominal power and \( C(T_{\text{Set}} - T_{\text{Sensor}})/t \) with \( C \) being the heat capacity of the heated zone, \( T_{\text{Set}} \) the setpoint temperature given by the occupancy, \( T_{\text{Sensor}} \) the measured temperature in the zone and \( t \) the time to reach the temperature setpoint. The heating power is split between radiative and convective power, 30% of the heating power is transmitted as radiative power while the rest is transmitted to the zone as convective power. The heating system has a COP (Coefficient Of Performance) of 3 used to calculate its electrical power consumption: \( P = \text{Total Heating Power} / \text{COP} \) and \( Q = 0 \).

Ventilation system

The modelled ventilation system is a constant air flow recuperation system. The efficiency of the pump for electricity consumption is modelled with a fan efficiency of 0.85 and a motor efficiency of 0.80. The heat recuperation efficiency can be set by the user in the parameters of the FMU. The default value for the heat recuperation efficiency is 0.7 (recupEff).

Building FMU parameters and complexity

The user of the building FMU can use different parameters to adapt the model to the situation he wants to simulate. Those accessible parameters are listed in the next table.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>idOcc</td>
<td>Integer</td>
<td>Select the occupancy profile</td>
</tr>
<tr>
<td>recupEff</td>
<td>Real</td>
<td>Set the heat recuperation efficiency of the ventilation system.</td>
</tr>
<tr>
<td>Distance</td>
<td>Real</td>
<td>Distance from the house to the feeder.</td>
</tr>
</tbody>
</table>

A building FMU holds 7 systems of non-linear equations that range from 1 to 18 equations per system and 2 systems of linear equations with 2 equations. There are also 3870 non-trivial equations in each FMU.

7.2.7.2 The grid model

The modelled grid is a radial grid as it is the main one used for low voltage distribution and it is easy to analyze. In this case it is a three phase system. The load flow is calculated with an incidence matrix that represent the grid. Each segment of the grid has its own complex impedance taking into account the resistive, capacitive and inductive effects in this part of the line with the form \( Z = R + jX \). The cable used is a 70 mm² aluminum cable with \( R = 0.461 \times 10^{-3} \Omega/m \) and \( X = 0.072 \times 10^{-3} \Omega/m \). The laws of Kirchhoff are used to determine the current in the line and the nodes and the voltage at each node with the load profile at each node.

The transformer

The transformer is modelled with the following equivalent scheme:

\[
\begin{align*}
Z_{\text{HV}} \quad & \quad Z_{\text{LV}} \\
\quad & \quad Z_{\text{IRON}} \\
\end{align*}
\]

Where \( Z_{\text{HV}} \) / \( Z_{\text{LV}} \) model the resistive / inductive effect of respectively the medium voltage and low voltage part of the transformer and \( Z_{\text{IRON}} \) models the iron loss of the transformer. The impedances are set by the choice of a rated apparent power of the 400 kVA transformer. The voltage on the secondary side is 230 V.
Feeder FMU parameter

The available parameters for the feeder FMU are listed in the next table:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLength[i]</td>
<td>Real</td>
<td>Length of the cable from the transformer to the i\textsuperscript{th} building</td>
</tr>
</tbody>
</table>

The Feeder FMU holds one non-linear equation and 779 other non-trivial equations.

⚠️ The number of buildings connected to the grid has to be chosen prior to exporting the FMU. As the number of neighborhoods has been chosen (to 12), exactly twelve building FMUs have to be connected in the calculation graph.

To run this example, you have to adapt the given \textit{iDEAS.dsl} to your processing context.

The script file assigns a different profile to each building (parameter idOcc). Each profile is composed of .txt files especially including active and reactive power consumptions for 1 year with a sampling of 600 s.

⚠️ Note the required resources (folder \textit{Data}, available in the distribution) must be located in the root folder \texttt{C:\} of the computer.

7.2.7.3 Graphs and simulation properties

![Figure 93: Calculation graph and dependency graph for the iDEAS case](image)

The chosen co-simulation parameters are the following:

- \textit{start time} = 0 s, \textit{stop time} = 86400 s;
- \textit{constant step size} = 600 s.
7.2.7.4 Discussion on the results

Results are given within two graphs drawn by the new DACCOSIM 2017 real-time visualization tool. They are in conformance with the expectations according to business experts.

Figure 94: Real part of the current and voltage for 4 buildings (#1, 4, 7 & 10)
8 Some known limits

Representation for Reals

DACCOSIM calculations are based on floating-point numbers to be in conformance with the fmi2DoStep() primitive of the FMI-CS 2.0 standard where both currentCommunicationPoint and communicationStepSize are represented as Real (double).

Floating-point numbers are represented by a fixed number of bits. In double-precision floating-point, for example, 53 bits are used, so the otherwise infinite representation is rounded to 53 significant bits.

0.1 is a typical example of inaccuracy in representation for floating points in double-precision. Let’s write it in binary truncated to 57 significant bits:

0.0001100110011001100110011001100110011001100110011011001...

In decimal, this is equivalent to:

0.1000000000000000055511151231257827021181583404541015625

which is slightly greater than 0.1

This default is well known in data computing but may induce confusion in user mind as it is especially visible in DACCOSIM results when a constant step size method is required for the co-simulation.

Very large co-simulations

The limit for the number of FMU blocks involved in a co-simulation has not been pinned down and is very dependent on the memory available on the machine. Note that when hundreds and hundreds of FMUs are to be used, a cluster environment should be preferable.

Another limit is given by the number of links, that is to say the external dependencies. When hundreds and hundreds of links are defined in the calculation graph, the generation of the dependencies is too long and should be cancelled by the user. A possible roundabout is to choose the NoCoInitialization option in order to avoid this too long process.

Temporary files not removed

FMUs are unzipped in a temporary folder and all temporary files are normally then removed on disk. Nevertheless, especially when a co-simulation run fails, this removal may be defeated.

The user can keep watch this point and manually remove rebellious folders fmu_.* generally findable in the folder C:\Users\<user>\AppData\Local\Temp\ on Windows environment.

Richardson method not correctly implemented

Some users have recently reported that the Richardson method is not correctly implemented in DACCOSIM 2017. After investigations of our developers, we advise not to use this method till deeper correction in next release.
DACCOSIM 2017 User’s Guide

9 Annex 1: about variable step methods implemented in DACCOSIM 2017

Assuming an FMU supplies a numerical method $\Phi$ with order $m$, DACCOSIM 2017 implements two categories of variable step methods allowing a local error calculation and step size tuning thanks to rollback capability from the FMI standard:

1) Calculating two approximations with same method but different step sizes accelerating by this way the convergence and offering a higher order extrapolation (e.g. Richardson extrapolation).

2) Calculating two approximations with same step size but another embedded method of different consistency order (e.g. single-step Euler, or multi-step Adams-Bashforth).

9.1 Calculating two approximations with same method

Denoting $Y(t_{i+1})$ as the exact solution of the integration at communication point $t_{i+1}$ and $\Phi$ the m-order numerical method used (by the FMU) to calculate an approximation at the same point, the equation becomes:

$$ Y(t_{i+1}) = \Phi(h_i) + K h_i^m + O(h_i^{m+1}) $$

With $K$ being a constant unknown value.

**How to estimate the local error with Richardson extrapolation?**

The Richardson extrapolation is commonly used to find a higher order approximation of $Y(t_{i+1})$ while using the same numerical method $\Phi$ (from the numerical solver inside the FMU).

For this, $Y(t_{i+1})$ can be calculated at $t_{i+1}$ using $\Phi$ with a step size of size $h_i$, and with a step size of size $h_i + \frac{h_i}{2}$ so that we get the following system with two equations and two unknowns:

$$ Y(t_{i+1}) = \Phi(h_i) + K h_i^m + O(h_i^{m+1}) $$

$$ Y(t_{i+1}) = \Phi\left(\frac{h_i}{2}\right) + \frac{K h_i^m}{2^m} + O(h_i^{m+1}) $$

Dropping tiny terms $O(h_i^{m+1})$, the two equations can be solved yielding approximate values for unknowns $Y(t_{i+1})$ and $K h_i^m$.

Taking $2^m$ times the second equation and subtracting the first equation gives:

$$ (2^m - 1)Y(t_{i+1}) \approx 2^m \Phi\left(\frac{h_i}{2}\right) - \Phi(h_i) $$

And then:

$$ Y(t_{i+1}) \approx \frac{2^m \Phi\left(\frac{h_i}{2}\right) - \Phi(h_i)}{2^m - 1} $$

We have hence generated an approximation of $Y(t_{i+1})$ whose error is of order $m + 1$, better than the starting approximation that is to say we have gotten an acceleration of convergence of 1 simply with a double approximation done at points $h_i$ and $\frac{h_i}{2}$.

Similarly, by subtracting the second equation from the first one in the system, we can find an evaluation of the local error using the Richardson extrapolation:

$$ Kh_i^m \approx \Phi\left(\frac{h_i}{2}\right) - \Phi(h_i) \frac{(1 - \frac{1}{2^m})}{(2^m - 1)} $$

This quantity is also an estimation of the truncation error done with the numerical method $\Phi$: $EST \approx |Kh_i^m|$.

Knowing $TOL$ as the acceptable tolerance, the optimum step size $h_{opt}$ can be calculated via the same kind of formula:

$$ TOL \approx |K h_{opt}^m| $$

By dividing the two quantities, we can set:

$$ h_{opt} \approx h_i \left(\frac{TOL}{EST}\right)^\frac{1}{m}.$$
Lots of articles then propose to introduce a safety factor $s_f \in [0, 1]$ to secure the calculation so that the next step size is given by: $h_{\text{opt}} \approx s_f h_i \left( \frac{\text{TOL}_j}{\text{EST}_j} \right)^{\frac{1}{m}}$.

Applied to FMUs, the advantage of this method is that starting from a communication point $t_i$, $\Phi(h_i)$ and $\Phi \left( \frac{h_i}{2} \right)$ can be calculated concurrently for a total of three calculation steps, simply modifying the step size for the FMU solver. However, two independent FMU instances need to be allocated in memory and the FMU state of both instances must be consistent at the beginning of each time communication point $t_i$.

**How to calculate the next step size?**

In the DACCOSIM 2017 tool, the maximum theoretical step size is calculated for each output and each exposed continuous variable $j$ with the following formula:

$$h_{\text{max},j} = \max \left( \min_{\text{step}}, \min \left( s_f \cdot h_i \left( \frac{\text{TOL}_j}{\text{EST}_j} \right)^{\frac{1}{m}}, \max_{\text{step}} \right) \right)$$

$$h_i \text{ IF } s_f \cdot h_i \left( \frac{\text{TOL}_j}{\text{EST}_j} \right)^{\frac{1}{m}} < h_i \text{ AND } (\text{EST}_j < \text{TOL}_j)$$

With:
- $\max_{\text{step}}$ being an upper value of the step size, $\min_{\text{step}}$ a lower value of the step size and $s_f$ a safety factor, all given as parameters in the tool;
- $h_i$ being the current step size;
- $\text{TOL}_j$ being the given relative tolerance and $\text{EST}_j$ the calculated truncation of the relative error, for each $j$ indexed variable.

DACCOSIM 2017 then select the next step size for the FMU as: $h_{\text{max}} = \min_j(h_{\text{max},j})$.

**When to rollback?**

When $h_{\text{max}} \geq h_i$, no rollback is required and $h_{\text{max}}$ becomes the step size for the next iteration: $h_{i+1} = h_{\text{max}}$. When $h_{i+1} < h_i$, a rollback is to be done, the iteration is erased and performed again with a smaller step size: $h_i = h_{\text{max}}$. Note the IF condition in the formula giving $h_{\text{max},j}$ is empirically set to avoid too frequent rollbacks.
9.2 Calculating two approximations with embedded methods

Denoting \( Y(t_{i+1}) \) as the exact solution of the integration at the communication point \( t_{i+1} \) and \( \Phi \) (resp. \( \Phi' \)) the m-order (resp. p-order) numerical methods embedded to calculate an approximation at the same communication point, the equations are:

\[
\begin{align*}
Y(t_{i+1}) &= \Phi(h_i) + Kh_i^m + O(h_i^{m+1}) \\
Y(t_{i+1}) &= \Phi'(h_i) + K'h_i^p + O(h_i^{p+1})
\end{align*}
\]

In this system, \( \Phi \) stands for the numerical method implemented in the FMU and \( \Phi' \) for another embedded method.

DACCOSIM 2017 tool adopted two different additional methods for \( \Phi' \):
- The simple explicit Euler method;
- And a more sophisticated and multi-step order method.

**Application to the Euler method**

Assuming that \( p < m \):

\[
\begin{align*}
Y(t_{i+1}) &= \Phi(h_i) + O(h_i^{p+1}) \\
Y(t_{i+1}) &= \Phi'(h_i) + K'h_i^p + O(h_i^{p+1})
\end{align*}
\]

Subtracting the second equation from the first one in the system, we can find an evaluation of the local error:

\[
K'h_i^p \approx \Phi(h_i) - \Phi'(h_i)
\]

This quantity is also an estimation of the truncation error done with the numerical embedded methods \( \Phi \) and \( \Phi' \):

\[
EST = |K'h_i^p|.
\]

Knowing \( TOL \) as the acceptable tolerance, the optimum step size \( h_{opt} \) can be calculated via the same kind of formula:

\[
TOL \approx |K'h_{opt}^p|.
\]

By dividing the two quantities, we can set: \( h_{opt} = h_i \left( \frac{TOL}{EST} \right)^{\frac{1}{p}} \).

As the Euler method is 1-order, \( p = 1 \) and \( h_{opt} = h_i \left( \frac{TOL}{EST} \right) \).

In the DACCOSIM 2017 tool, the maximum theoretical step size is calculated for each output and each exposed continuous variable \( j \) with a formula very similar to the one used for the Richardson extrapolation, that is to say, with the same notations:

\[
h_{max,j} = \begin{cases} 
\max \left( \min_{\text{step}}, \min \left( \frac{h_i \left( TOL_j \right)}{EST_j}, \max_{\text{step}} \right) \right) \\
h_i \left( TOL_j \right) & \text{IF} \left( \frac{h_i \left( TOL_j \right)}{EST_j} < h_i \ AND \ (EST_j < TOL_j) \right)
\end{cases}
\]

DACCOSIM 2017 then select the next step size for the FMU as: \( h_{max} = \min \left( h_{max,j} \right) \).

And the rollback conditions are exactly the same as for the Richardson extrapolation.

**Application to the Adams-Bashforth-Moulton method**

k-step Adams-Bashforth formula are known to be explicit k-order methods while k-step Adams-Moulton ones are implicit k+1-order. To make this formula implicit, we can use a predictor-corrector pair expressed as a variable step explicit k-step Adams-Bashforth predictor and a k-step Adams-Moulton corrector. As given by Lopez et al., 2010 the predictor and the corrector are:

\[
\begin{align*}
p(t_{i+1}) &= \Phi(h_i) + h_i \sum_{j=0}^{k-1} g_j(i) \beta_j(i) \varphi_j(i) \\
\Phi'(h_i) &= p(t_{i+1}) + h_i g_k(i) \varphi_k(i + 1)
\end{align*}
\]
Where the different coefficients are given by the following recurrences:

\[
\begin{align*}
\beta_0(i) &= 1 \\
\beta_j(i) &= \beta_j(i-1) \frac{t_i + \tau - j - 1}{t_i - t_j - 1} \\
\phi_0(i) &= \Phi(h_i) \\
\phi_j(i) &= \phi_j(i-1) - \beta_j(i-1) \phi_j(i-1) \\
c_{0,q}(t_{i+1}) &= \frac{1}{q} \\
c_{j,q}(t_{i+1}) &= c_{j-1,q}(t_{i+1}) - c_{j-1,q+1}(t_{i+1}) \frac{h_i}{t_{i+1} - t_i - j - 1} g_j(i) = c_{j+1}(t_{i+1})
\end{align*}
\]

It is easy to check that:
- the coefficients \(\beta_j(i)\) and \(g_j(i)\) are scalar and independent from any function values \(\Phi(h_i)\);
- the \(\phi_j(i)\) coefficients depend on the \(k-1\) previous function values \(\Phi(h_i)\).

Assuming the second embedded method \(\Phi'\) is a \(p\)-order one with \(p > m\), the system becomes:

\[
\begin{align*}
Y(t_{i+1}) &= \Phi(h_i) + K h_i^m + O(h_i^{m+1}) \\
Y(t_{i+1}) &= \Phi'(h_i) + O(h_i^{m+1})
\end{align*}
\]

And the optimum step size \(h_{opt}\) is calculated as: \(h_{opt} = s_f h_i \left( \frac{TOL_j}{EST_j} \right)^\frac{1}{m} \).

To limit the number of steps to keep in memory, we can choose \(p = m + 1\) and then adopt a \(m\)-step Adams-Moulton formula.

In the DACCOSIM 2017 tool, the maximum theoretical step size is calculated for each output and each exposed continuous variable \(j\) with a formula exactly identical to the one used for the Richardson extrapolation, that is to say, with the same notations:

\[
h_{max,j} = \begin{cases} 
\max \left( \min_{step}, \min \left( s_f \ast h_i \left( \frac{TOL_j}{EST_j} \right)^\frac{1}{m}, \max_{step} \right) \right) \\
h_i IF \left( s_f \ast h_i \left( \frac{TOL_j}{EST_j} \right)^\frac{1}{m} < h_i AND (EST_j < TOL_j) \right)
\end{cases}
\]

DACCOSIM 2017 then selects the next step size for the FMU as: \(h_{max,j} = \min_j (h_{max,j})\).

And the rollback conditions are exactly the same as for the Richardson extrapolation.