MeCoSim

Membrane Computing Simulator - User Manual

23/01/2012

Research Group on Natural Computing
Luis Valencia Cabrera
lvalencia@us.es

Contents

1. Getting Started

Follow these simple steps to start using MeCoSim (Membrane Computing Simulator):

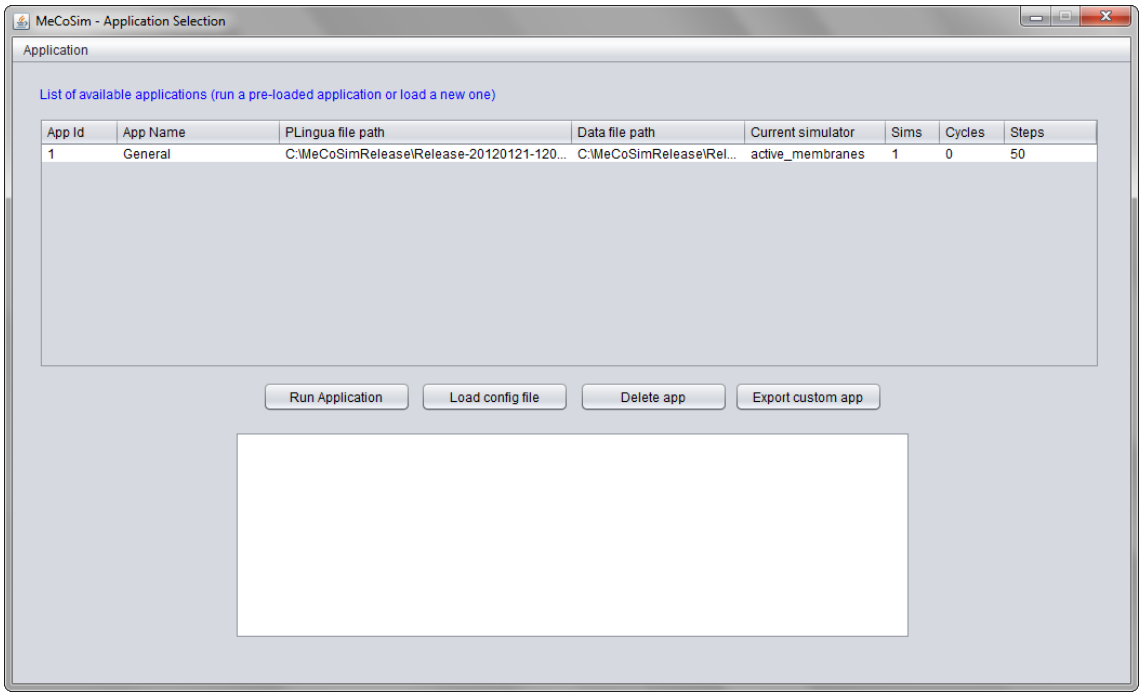
- Visit MeCoSim page: http://www.p-lingua.org/mecosim/docsite/
- Download the latest version of MeCoSim.
- Unzip the file.
    - Now you have a MeCoSim folder, including all the needed files.
    - You are now ready to run MeCoSim by launching the file ***Run_MeCoSim.bat***.

Additional info (present in MeCoSim downloads web page):

- MeCoSim zip file includes, among others, the following files inside MeCoSim folder:
    - **Run_MeCoSim.bat**: MeCoSim execution file.
    - lib/MeCoSim2.0Core.jar: is the core of MeCoSim application.
    - lib/pLinguaCore.jar: includes a recent version of pLinguaCore (including different variants of cell like, multienvironment, tissue like and SN P systems). Can be replaced for your own newer or custom version.
    - prop/config-properties: properties file to change some execution parameters for running MeCoSim (max dedicated memory in MB, main params).
    - prop/ecosim-properties: some setup parameters used by MeCoSim.
    - plugins: folder where we can include our own plugin jar files to be loaded for MeCoSim.
    - prop/plugins-properties: properties file to indicate to MeCoSim which method of which class it must call to run a specific plugin.
    - userfiles: folder where the user can include his/her own files to use in MeCoSim (.xls/.ods spreadsheet config file, .pli P-Lingua model files, .ec2 scenario data files, etc).

2. General MeCoSim Window

When you enter MeCoSim, a general graphical interface is showed:



This interface includes a list with the custom applications that we have previously loaded.

The list includes the following information:

- App id: an ID that the users have to set in their applications to identify it uniquely.
- App name: the desired name of the application (it is advisable to provide unique names, but not mandatory).
- P-Lingua file path: path to the .pli file containing the current P-Lingua model.
- Data file path: path to the .ec2 file containing the input data of the current scenario.
- Sims: number of simulations (it makes sense when we want to perform several simulations to get averages, deviations, etc. over the data across the different simulations).
- Cycles: number of cycles to simulate (it makes sense when we have models following some cyclic process repeating a sequence of steps each cycle, without a halting condition; otherwise we state 0 as the number of cycles, running the simulation until we reach a halting configuration).
- Steps: is the number of steps by cycle, making sense when we have a non-zero number of cycles.

By default, we provide a general application with id 1, called “General”, pre-configured for a very simple example of a custom MeCoSim app. We explain this later.

Below the list, there are some buttons whose functionality is explained in specific sections of the manual, but here you have a brief description:

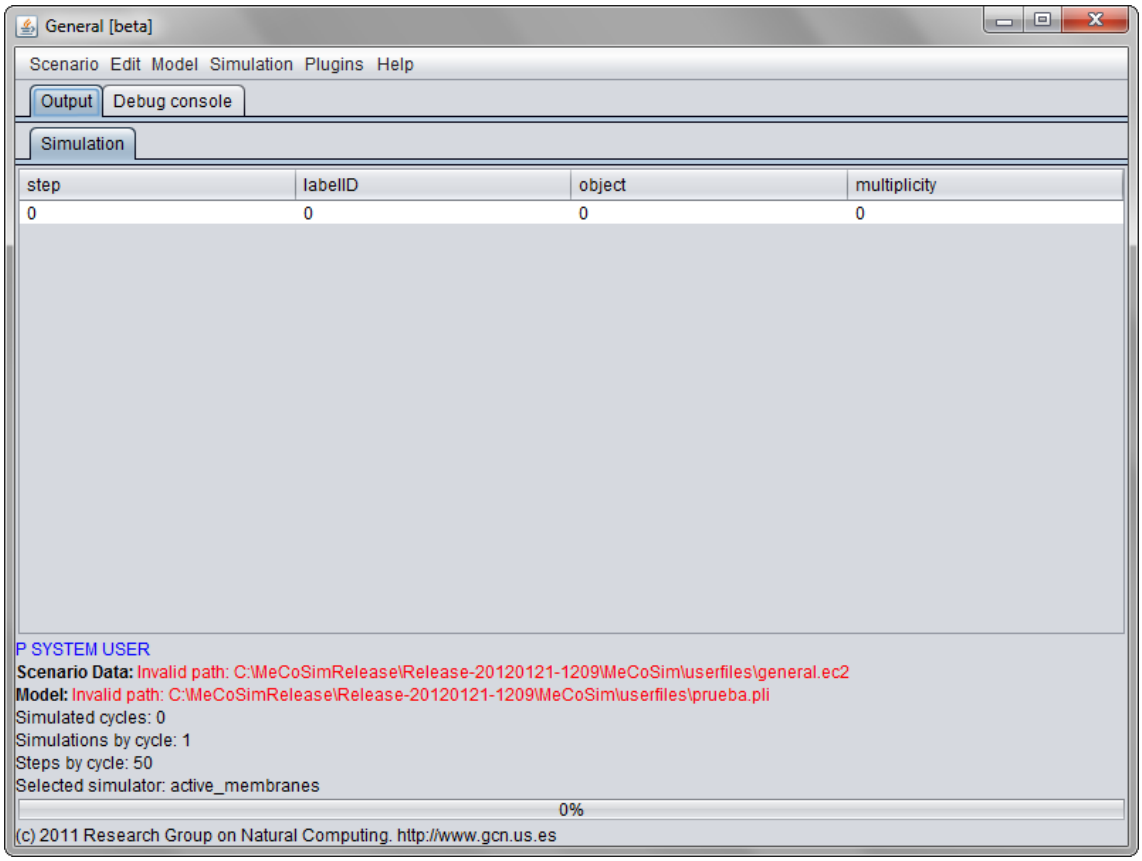
- “Run application”: runs the selected custom application, showing the main window of the custom application as it was configured.
- “Load config file”: creates a new custom application by loading a configuration file, or modify an existing one by reloading (total or partially) the configuration file.
- “Delete app”: deletes the selected custom application, deleting all its related information for MeCoSim.
- “Export custom app”: generates a self-contained custom app from the info contained in MeCoSim, providing a zip file with all the specific information and files, ready to use by running the application for a P systems designer or an end-user.

All the functionality provided by these buttons is also available from the “Application” menu.

Below the buttons we see a text area where the standard output is redirected, showing the user information about the actions performed and possible running errors (if you detect some errors, or you have suggestions, questions or comments, please notify to lvalencia@us.es).

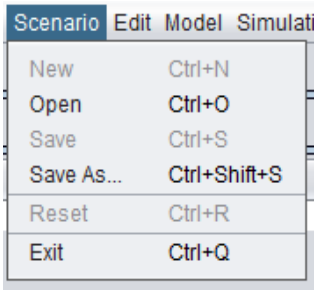
3. Run application

Whenever we run a custom MeCoSim application (via button, menu item or a previously exported app), a “MeCoSim – Custom app” window is showed, similar to this:

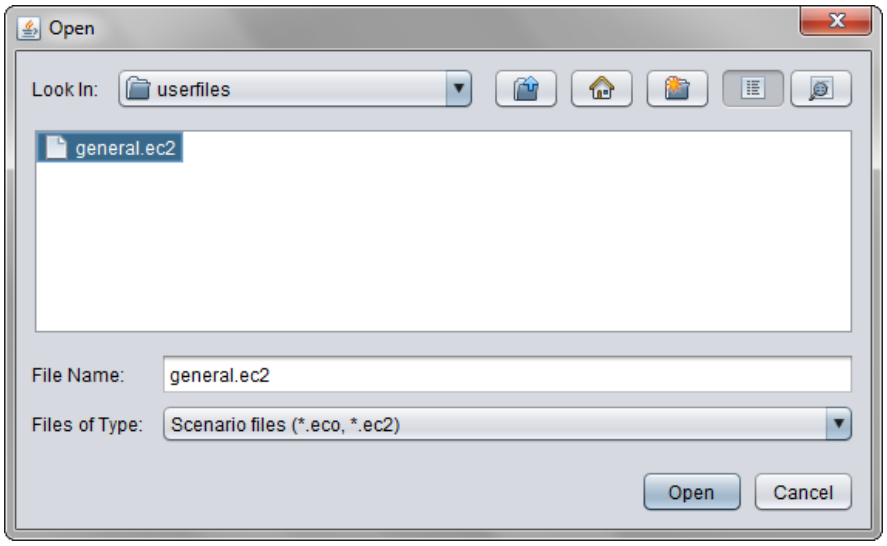


The title of the window corresponds to the name set in the configuration file that we loaded in MeCoSim. In the menu bar, we see some menus:

- Scenario:
    - New: enable the addition of a new scenario.
    - Open: open a file with the data of a previously saved scenario.
    - Save: save the data of the current scenario.
    - Save As: save the current data with a different name.
    - Reset: reset the behavior of the app to the default state.
    - Exit: exit the MeCoSim custom app (without exiting from the main window of MeCoSim).



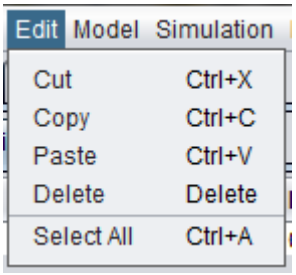
By default, when we load a config file for a new application or open the “General” example, the path to the file is not set or is invalid. We must select our scenario file clicking at “Open”. Then we select the desired .ec2 scenario data file.



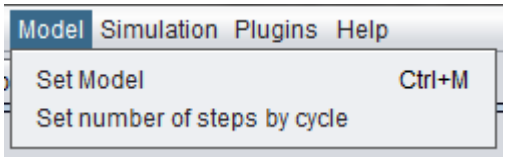
Now we can see how the path to the scenario data file (.ec2) is properly set:

- Edit:

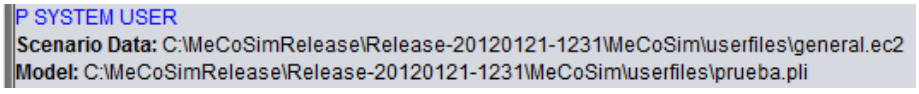
  As we can see, this menu provides the basic functionality to Cut, Copy, Paste, Delete and Select All.



- Model (available in designer mode):
  - Set Model: permit the selection of a P-Lingua model file (.pli).
  - Set number of steps by cycle: in cyclic models, set the number of steps that form a cycle.

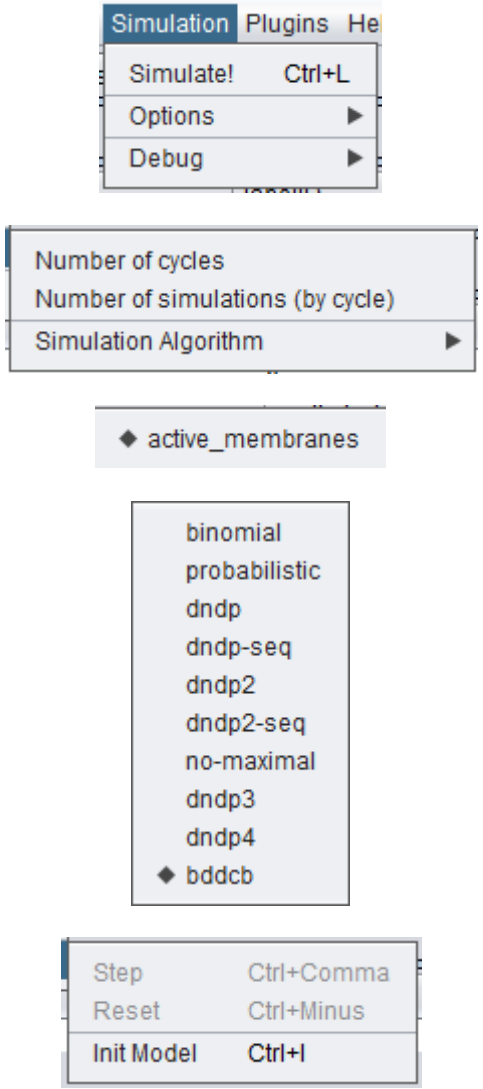


In the present example, if we choose the P-Lingua model file (.pli), we can see that the path is properly set:

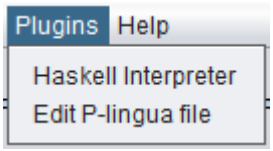


- Simulation:

- Simulate!: runs the simulation of the model set in the P-Lingua file. If the model needs some parameters not assigned in the .pli file, it has to be possible to generate them from the input data provided by the user as part of the current scenario.
- Options: permit selecting the number of cycles to simulate (for cyclic models; zero otherwise, running each simulation until a halting configuration is reached). We can also select a specific algorithm from a list of available algorithms, depending on the loaded model (e. g., active membranes for the present case, or dndp4 or bddcb chosen for a probabilistic model, from the showed list).
- Debug (available in designer mode): some options to init model, run step by step (Step), or reset, as we will see when describing the Debug console.



- Plugins: permit running plugins that included in MeCoSim. We provide a way to include some new features in MeCoSim by means of external plugins, integrated with the core of MeCoSim by filling some lines in the plugin-properties file, as we further describe in section 7: plugins development.



4. Load config file

The previous sections are based on a custom application, previously configured and loaded in MeCoSim. If we want to set up a new application, we must fill a spreadsheet file with the desired inputs and outputs. In what follows, we describe the various tabs of the spreadsheet to populate.

- General information:

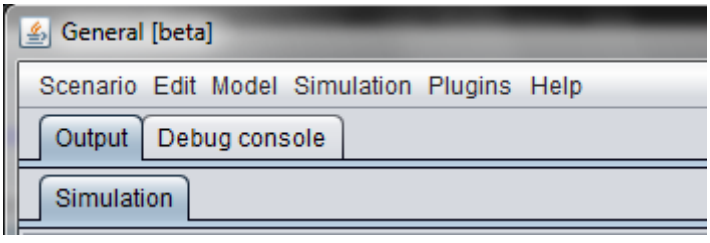
App Id	App Name	Version	Version Date	Ec2 file path	P-Lingua File Path	Simulation cycles	Simulations by cycle	Steps by cycle	Mode
1	General			C:\	C:\	1	1	50	Designer

We set a unique app Id number, the application name, the default paths to scenario and model files, simulation cycles, number of simulations, steps by cycle and mode (Designer or End-user). This information has been described yet.

- Tabs hierarchy:

Tab Id	Tab Name	Tab Parent Id	Order	Rol (optional)
1	SAT Spiking	0		MainTab
2	Output	1		
3	Simulation	2		

This feature enables the user to set the tabs hierarchy to be showed in the main window of the custom app. The first row with **Tab parent id** = 0 plays the role of the application, so the remaining rows should refer to 1 if we want to have these tabs at the first level (that is to say, its parent **tab** is the app), or any other number greater than zero (if zero, they are not showed), implying that they are children tabs of the referred tab parent id. In this example, we have only one tab (Output) at the first level, containing another tab (Simulation). In addition, if we are in "Designer" mode, a "Debug console" tab is also included at the first level. Thus, the custom application will present this hierarchy:



- Tables config:

Table Id	Table Name	Tab Id	Columns	Init Rows	Save To File	Input / Output	Output Graphic
1	Simulation	3	4	1	TRUE	Output	

Here we list the tables to include in the **leaf tabs** of the previous hierarchy (each table in the specified tab id), indicating the type of the table (Input or Output), the number of columns, the

initial number of rows (it usually changes for output tables after simulating). We also have to tell if the table should be saved as part of the scenario data file (.ec2).

Along with this information, more specific information about each table is needed. This is done by means of a specific tab in the spreadsheet files for each table. For example, for “Simulation” table we must have a tab called “TC_Simulation” (TC means table config).

Column Id	Column Name	Default Value	Editable	Tooltip
1	step		TRUE	step
2	labelID		TRUE	labelID
3	object		TRUE	object
4	multiplicity		TRUE	multiplicity

As you can see, this information provides an id, name, tooltip text, a default value and an indicator to enable/disable the edition of each particular column.

- Simulation params:

Param Name	Param Value	Index 1	Index 2	Index 3	Index 4
g	1				

When a P systems designer models a problem, usually tries to solve an abstract problem, a family of problems, not a single scenario. For that reason, P-Lingua permits defining a model with some variable information, in the form of parameters. These parameters could be written in the model file, but is more interesting to provide a way to permit the users introducing the relevant information about different scenarios without changing the model file.

Then we need a way to transform the input data provided by the user in parameters for P-Lingua to populate the initial configuration and rules of the instantiated P system.

The designer can introduce in the spreadsheet file the name of a parameter, optionally with a number of indexes between 1 and 4, joint with the value of the parameter. This value is expressed in a language that permits the use of usual numbers, mathematical operators, input tables’ data access, previously set parameters and some predefined functions. As an example, we include the parameter **g**, with value 1. Examples of the use of other important ingredients are:

- Param **x** with value **<1,2,4>**: access to the data of the table with id 1, specifically to row 2, column 4.
- Param **p** with value **1-<@ncdf,T{1,2},0.5,IT{1}>**, with index1 equals to **[1..20]** and index2 equals to **[1..18]**: returns parameters from p{1,1} to p{20,18}, each of them calculated like 1 minus the value of the normal distribution cumulative distribution function with parameters mu = T{1,2}, sigma = 0,5 and x = IT{1}. T{1,2} refers to a previously calculated parameter T, replacing for each parameter N for the current value of the iterator with index N.

We are working in a detailed manual with the description of the language accepted for the mechanism for generate these parameters for P-Lingua.

In addition, an extensible mechanism to provide specific functions is provided. These functions should be coded in your own jar files (added to the plugins folder), implementing the IAlgorithm interface and adding the function to ecosim-properties file. For example, we have by the default in the file, among other functions:

func-floor = algorithms.Floor
func-ceil = algorithms.Ceil
func-min = algorithms.Min
func-eq = algorithms.Equal
func-if = algorithms.If
func-abs = algorithms.Abs
func-g = algorithms.Greater
func-log = algorithms.Log
func-exp = algorithms.Exp

We are also working in incorporating a visual way to add functions written in Haskell inside MeCoSim, since it is a purely-functional language and we have defining mathematical functions.

- Simulation results:

Result Table Id	Result Table Name	Table Id	Referred Table Id
1	Simulation	1	0

Once the simulation have run and finished, some result should be showed to the user, but the type of result may be different in function of the problem, so we must indicate which result we want. MeCoSim permits defining output tables to show the desired processed results, possibly visualizing the output in a graphical way (if indicated in the column Graphic output in the tablesConfig tab of the spreadsheet). To define each result, we have to provide an Id, a name, the id of the output table to show this result (0 if we are defining an auxiliary table), and a referred table id (if the row corresponds to a result that depends on a previously defined auxiliary table). In addition to this tab, we have to fill a specific tab for each result. For example, for the Simulation result we have the “SD_Simulation” tab:

Criteria Id	Select/Where/Group	Criteria	Formula	ReferredCrite	Argument	Qualified Name	Where/Selec	Where condition
1	Select	step						
2	Select	labelID						
3	Select	object						
4	Select	multiplicity						
5	Where	simulation					Integer	1

Here we set the fields to show in the output table (Select rows), some criteria to filter in function of different conditions (Where rows), and possibly some grouping criteria (Group rows). These types of criteria derive from the theory of databases, as it provides a very general and flexible way to filter and processing data. The configured results are translated internally to SQL language to launch a query on the embedded database (HSQLDB on memory DB) that contains the result of the computation. One advantage of this approach is that we can use almost all the formula provided by the HSQLDB database(all of them including two or less input arguments).

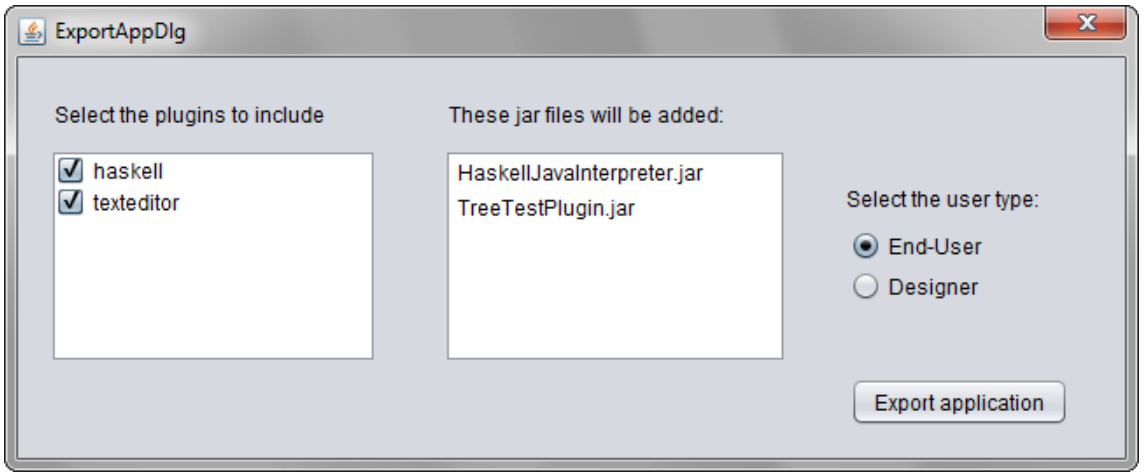
We are working in a detailed manual of the functionality of the language to define rich results, but we include some example here to illustrate some possible ingredients.

Criteria Id	Select/Where/Group	Criteria	Formula	Referred Criteria Id	Argument	Qualified Name	Where/Select condition type	Where condition
1	Select	simulation						
2	Select	year						
3	Select	labelID						
4	Auxiliary	multiplicity						
5	Select	formula	SUM	4		mult		
6	WhereAux	object					String	X{1%
7	Auxiliary	step						
8	WhereAux	formula	MOD	7	17		Integer	0
9	WhereAux	labelID					String	1
10	WhereAux	formula	AND	6	8			
11	Where	formula	AND	9	10			
12	Group	simulation						
13	Group	year						
14	Group	labelID						

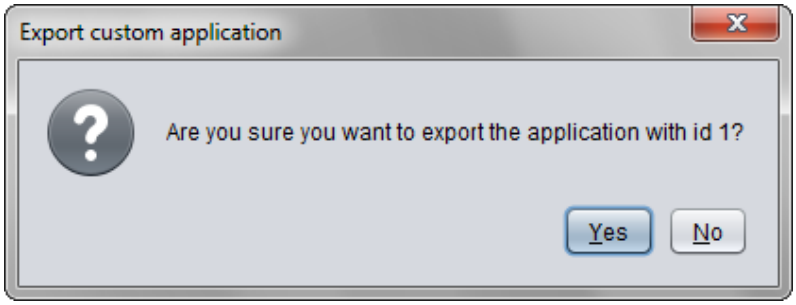
For any doubts, questions or comments, please feel free to contact with us in the e-mail lvalencia@us.es.

5. Export app

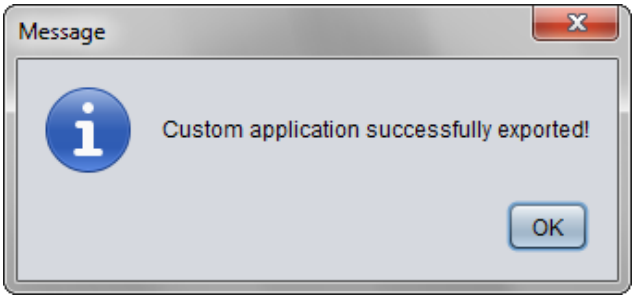
MeCoSim includes a way to export a single custom application for P systems designers or end users, deleting information about other applications, and abstracting them the non-relevant information depending on the type of user (Designer or End-User).



The export dialog enables the choice of the user type, along with the selection of the desired plugins to add to each custom application, from the available set of plugins present in the database. If we click the button, a confirmation message is presented:

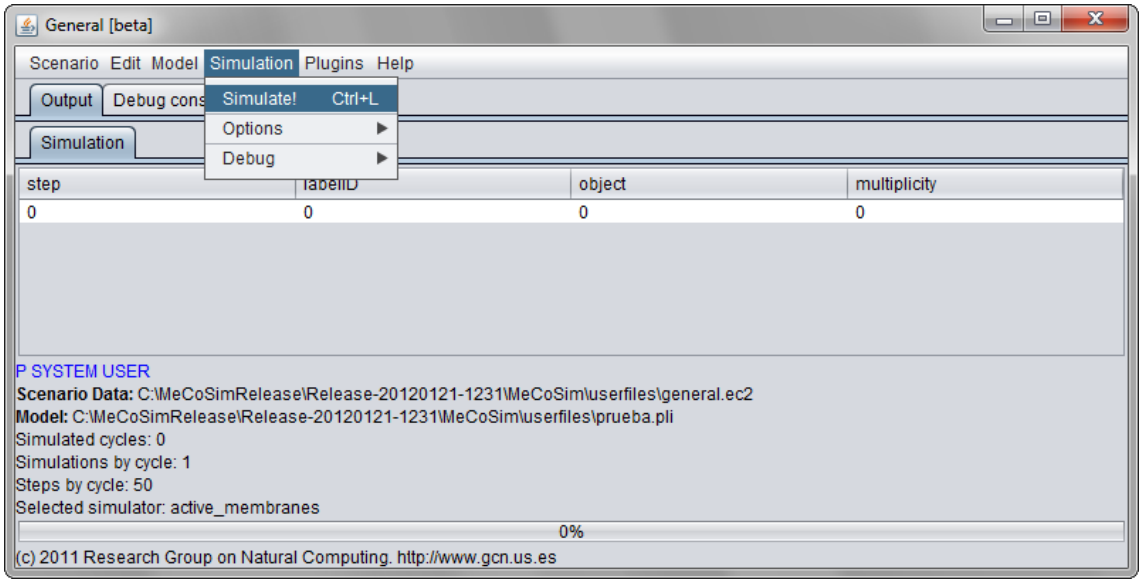


If confirmed, a zip file is generated with the self-contained custom application, including all the needed files and folders. After that, an information message is showed:



6. Simulation

Once we have set the model and the input data needed for our specific scenario, we run the simulation by pressing the menu item “Simulate!” (or pressing Ctrl+L):



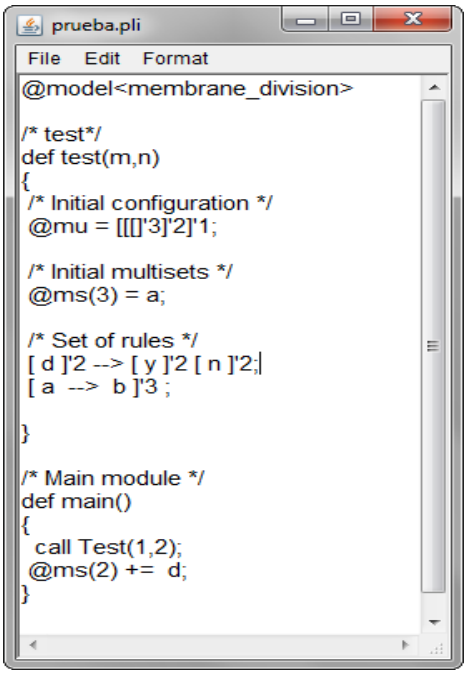
Then the simulation is performed, until a halting configuration is reached (if the number of cycles is set to 0) or a number of cycles (if they are set to more than zero, being each of them composed of the set number of steps by cycle). If a number of simulations greater than 1 is set, then the process is repeated the fixed number of simulations.

After the set of simulations has finished, MeCoSim shows the user the required output tables and graphics, as configured in the spreadsheet file (as described in section 4 – Load config file).

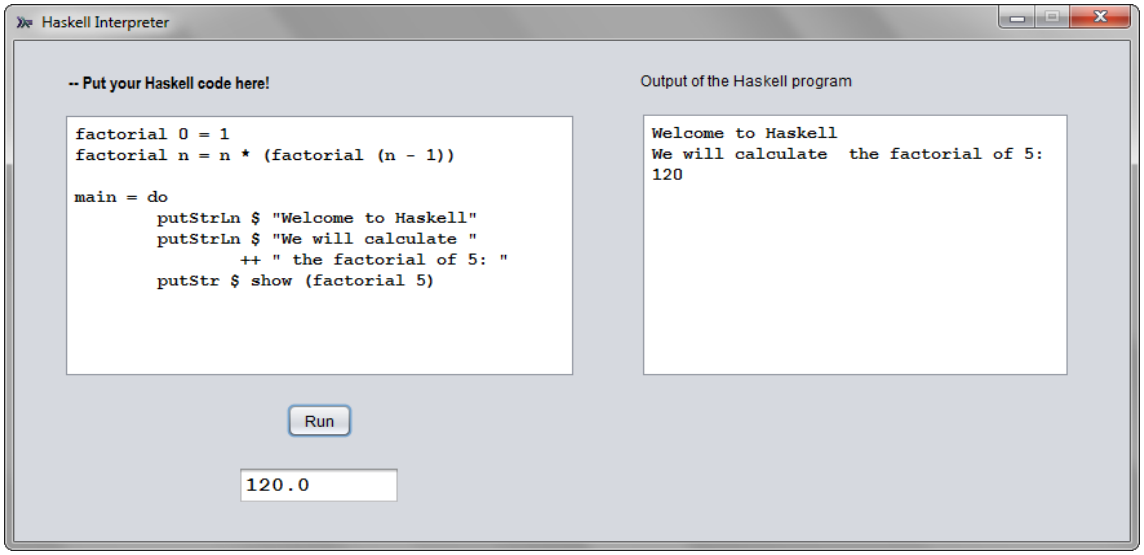
7. Plugins Development

In order to easily extend the functionality provided for MeCoSim and integrate it with other several applications, existent or to be developed, in a decoupled way in the sense of independent programs and developments, we have provided a simple mechanism to add external independent programs in the form of jar files to MeCoSim. By default, we provide two plugins to serve as examples:

- Edit P-Lingua file: opens the selected P-Lingua file in a very basic editor:



- Haskell Interpreter: an experimental Java interface to "compile and run" Haskell code from MeCoSim, showing the standard Haskell output in a textarea (**you need to have ghc interpreter installed**).



These plugins are under development and will be improved as soon as possible.

If you want to add your own Java plugins to MeCoSim, supposed that you have the application developed independently as a jar file (or a folder containing a main jar file), the process is as follows:

1. **Include the jar file** or complete folder **in the plugins folder** of MeCoSim.
2. **Write** some few lines **in plugins-properties** file (we illustrate with an example):
    a. ***plugin-***texteditor = alvarotreeapp.trail
    b. ***pluginname-***texteditor = Edit P-lingua file
    c. ***pluginmethod-***texteditor = openPLinguaFile
    d. ***pluginjar-***texteditor-1 = TreeTestPlugin.jar

    In this example, we name a plugin "texteditor", implemented in the static method "openPlinguaFile" of the qualified class "alvarotreeapp.trail" (inside the jar file "TreeTestPlugin.jar"), visible in MeCoSim through the menu option "Edit P-Lingua file". If you need some input parameters to your program, you can set them with an additional line for each needed input parameter (e. g. ***pluginparam-***texteditor-1 = va).