

CD++

A tool for DEVS and Cell-DEVS Modeling and Simulation

User's Guide

DRAFT – 8/22/2005

Gabriel A. Wainer

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Dr. Ottawa, ON. Canada

<http://www.sce.carleton.ca/faculty/wainer>
gwainer@sce.carleton.ca

Table of Contents

CD++.....	10
CD++BUILDER – QUICK REFERENCE GUIDE.....	10
DEVS MODEL DEFINITION: ATOMIC MODELS.....	21
CREATING NEW ATOMIC MODELS USING CD++BUILDER.....	21
<i>Opening an existing project</i>	21
<i>Creating a new project</i>	22
<i>Creating and editing files</i>	24
<i>Adding Files to projects</i>	25
<i>Using the Navigator</i>	28
<i>Using the task list</i>	29
<i>Using the Outliner</i>	29
DEFINING ATOMIC MODELS IN CD++.....	29
<i>Adding new models</i>	30
<i>Interacting with the Simulator</i>	30
<i>An example: a model of a queue</i>	31
<i>Creating new atomic models for parallel simulation</i>	34
COUPLED MODELS.....	40
CELL-DEVS MODELS.....	42
SUPPORTING FILES.....	45
DEFINING INITIAL CELL VALUES USING A .VAL FILE.....	45
DEFINING INITIAL CELL VALUES USING A .MAP FILE.....	46
EXTERNAL EVENTS FILE.....	46
PARTITION FILE.....	46
OUTPUT FILES.....	47
OUTPUT EVENTS.....	47
FORMAT OF THE LOG FILE.....	48
PARTITION DEBUG INFO.....	50
OUTPUT GENERATED BY THE PARSER DEBUG MODE.....	52
RULE EVALUATION DEBUGGING.....	53
MODEL SIMULATION.....	57
SIMULATION THROUGH ECLIPSE	57
<i>Compiling a new model</i>	57
<i>Simulating a model</i>	58
<i>Creating drawlog file for models</i>	59
<i>CD++ Modeler</i>	60
<i>Simulation Server</i>	62
PARALLEL SIMULATOR.....	69
UTILITY PROGRAMS.....	72
COUPLED ANIMATE, ATOMIC ANIMATE, CELL-DEV ANIMATE.....	73
DRAWLOG.....	75
<i>Bidimensional cellular models</i>	77
DEVS VIEW.....	79
<i>DEVS View Tutorial</i>	79
LTRANS (LATTICE TRANSLATOR).....	82
SETTING RANDOM INITIAL STATES – MAKERAND.....	87

CONVERTING .VAL FILES TO MAP OF VALUES – ToMAP.....	90
BITMAP TRANSLATOR.....	91
CONVERT TO CD++.....	93
MESSAGE COUNTER.....	93
ERROR CALCULATION.....	95
EXTRACT FROM DRAWLOG FILES.....	98
GRAFCELL.....	100
PARLOG.....	102
LOGBUFFER.....	103
SIMUMODELICA.....	103
CD++MODELER	107
<i>To install and run the standalone GUI:</i>	108
Using CD++Modeler.....	108
<i>Creation of DEVS Models (using the design space)</i>	109
Create Atomic Model.....	109
Setting the Model Title.....	110
Adding Units.....	110
Setting Properties of Units.....	111
Deleting Units.....	112
Adding Ports.....	112
Deleting Ports.....	112
Adding Links.....	113
Adding Transition Events to Internal and External Links.....	115
Deleting Links.....	117
Saving and exporting the model.....	117
8.16.2.1Create Coupled Model.....	119
Adding Atomic Units.....	120
Adding Coupled Models.....	122
Adding Ports.....	124
Adding Links.....	126
8.16.2.2.1Importing Models.....	128
8.16.2.2.2Exploding Models.....	133
Adding Images to Models.....	138
Adding Background.....	138
Adding Image to Units on Coupled Canvas.....	140
8.16.3 <i>Visualization of DEVS models (using the Animate menu commands)</i>	142
Model type selection.....	142
AtomicAnimate using: Atomic-DEVS, Coupled-DEVS, Atomic Cell-DEVS.....	143
AtomicAnimate Example of Atomic-DEVS: FunctionEval.....	145
Select the atomic-DEVS model to be visualized.....	147
Select the output signals to be plotted.....	147
8.16.3.2.1Select the time format for the horizontal axis.....	148
8.16.3.2.2Show the values of the output signal on the plot.....	150
Zoom in or out of the horizontal axis of the plot.....	152
Zoom in or out of the vertical axis of the plot.....	153
8.16.3.2.3Set the lower and upper bounds of the vertical scale.....	154
8.16.3.2.4Jump to the specified time interval.....	155
Go to the previous/next time interval.....	156
Jump to a specific time.....	158
AtomicAnimate Example of Coupled-DEVS: 4BitCounter.....	160
Select the model/component to be visualized.....	163
Availability of signals for the selected model/component.....	168
8.16.4 <i>CoupledAnimate using: Coupled-DEVS</i>	169
CoupledAnimate Example of Coupled-DEVS: 4BitCounter.....	171
Start the visualization.....	172
Stop the visualization.....	173

8.16.4.1	Pause the visualization.....	174
8.16.4.2	Go to the next display of the visualization.....	175
	Set the delay between displays	175
8.16.5	<i>Cell-DEVS animation using: Atomic Cell-DEVS, Coupled-DEVS.....</i>	177
	<i>Cell-DEVS animation Example of Atomic Cell-DEVS: Fire</i>	181
	Loading models to be visualized.....	181
	Adding models to the Available list.....	181
8.16.5.1	Selecting available models to be loaded.....	184
8.16.5.2	Loading models to be visualized.....	162
8.16.5.3	Modifying the appearance of the visualization.....	164
	Modify the palette of the loaded model(s).....	164
8.16.5.4	Select the lattice type.....	172
8.16.5.5	Show two-dimensional or multi-dimensional display.....	173
	Show cell values of the loaded model(s).....	173
8.16.5.6	Show names of the loaded model(s).....	174
	Remove/add the grid of the lattice.....	174
8.16.5.7	Running/Navigating the visualization.....	175
	Set the delay between steps.....	175
	Play/start the visualization.....	175
	Stop the visualization.....	176
8.16.5.8	Pause the visualization.....	179
8.16.5.9	Go directly to a particular time.....	180
	Go directly to a particular step.....	180
	Go to the previous/next step.....	181
8.16.6	<i>Cell-DEVS animation Example of multiple Atomic Cell-DEVS: Fire, SatelliteClouds.....</i>	184
8.16.6.1	Loading multiple (unrelated) models to be visualized.....	185
	Adding models to the Available list.....	185
	Selecting multiple available models to be loaded.....	185
8.16.6.2	Loading selected models to be visualized.....	187
8.16.6.3	Running/Navigating the visualization.....	190
	Play/start the visualization.....	190
	<i>Cell-DEVS animation Example of 3D Atomic Cell-DEVS: SatelliteClouds.....</i>	195
	Show multi-dimensional display of loaded models.....	196
8.16.7	<i>Cell-DEVS animation Example of Coupled-DEVS: RiceField.....</i>	197
8.16.7.1	Loading multiple models to be visualized.....	198
	Adding a coupled model to the Available list.....	198
8.16.7.2	Loading selected models to be visualized.....	201
8.16.7.3	Modifying the appearance of the visualization.....	204
	Modify the palette of the loaded models.....	204
	Running/Navigating the visualization.....	206
	<i>ther tools (using the Execute menu commands).....</i>	210
	Local CDD.....	210
	Remote CDD.....	213
	Text Editor.....	213
8.16.7.4	Drawlog.....	215
8.16.8	<i>CD++ Modeler: Suggestions & Bugs.....</i>	218
	Creating a Graphical Coupled Model (.gcm).....	218
	<i>AtomicAnimate: Suggestions & Bugs.....</i>	220
	Example: FunctionEval.....	220
	Example: 4BitCounter.....	220
	<i>Cell-DEVS animation: Suggestions & Bugs.....</i>	226
8.13.14	<i>Visualizing Cell-DEVS models.....</i>	227
	<i>Display simulation results for multiple models.....</i>	227
8.13.15	<i>Execute menu commands: Suggestions & Bugs.....</i>	229
	Local CDD.....	229
	Text Editor.....	229

Drawlog.....	229
7CGGAD GRAPHIC TOOL – USER MANUAL.....	231
INTRODUCTION.....	231
DEFINITION.....	231
9.1 EDITION (EDITING) OF COUPLED GGAD (CGGAD) MODELS.....	232
<i>Initial (Beginning)</i>	232
<i>Add atomic models</i>	234
<i>Add coupled models</i>	235
<i>Add ports</i>	236
<i>9.1.1 Add connections (links) between ports</i>	237
<i>Delete objects</i>	239
<i>Explode a model</i>	239
<i>Save and export the coupled model</i>	242
EDITION (EDITING) OF GGAD ATOMIC MODELS.....	243
<i>Initial (Beginning)</i>	243
<i>Add states</i>	244
<i>Add ports</i>	245
<i>Add state variables</i>	247
<i>9.1.2 Add internal transitions</i>	248
<i>Add external transitions</i>	251
<i>Add actions to the transitions</i>	254
<i>Delete objects</i>	257
<i>Menus</i>	258
<i>Save the atomic model</i>	261
9.2 COUPLED GGAD (CGGAD) MODELS.....	262
<i>Add ports</i>	262
<i>Delete objects</i>	264
9.3 GGAD ATOMIC MODELS.....	266
<i>Delete objects</i>	266
10APPENDIX A - CD++BUILDER – INSTALLATION GUIDE FOR WINDOWS	271
INSTALLATION.....	271
<i>STEP 1: Installing Java JRE</i>	271
<i>STEP 2: Installing Eclipse SDK</i>	271
<i>STEP 3: Installing Cygwin</i>	272
<i>STEP 4: Installing the CD++ Builder Plugin</i>	281
10.1.5 Troubleshooting.....	284
10.1.5.1 Internal File Corrupted.....	284
10.1.5.2 ‘Simu.exe: No such file or directory; Simu was was not created!’	286
10.1.5.3 ‘Deprecated JAVA methods’	288
10.2 COMMAND LINE INSTALLATION	289
10.3 INSTALLATION FOR PARALLEL SIMULATION.....	289
APPENDIX B - LOCAL TRANSITION FUNCTIONS FOR CELL-DEVS MODELS.....	293
A GRAMMAR FOR WRITING THE RULES.....	293
PRECEDENCE ORDER AND ASSOCIATIVITY OF OPERATORS.....	294
FUNCTIONS AND CONSTANTS ALLOWED BY THE LANGUAGE	295
<i>Boolean Values</i>	295
Boolean Operators.....	295
<i>Functions and Operations on Real Numbers</i>	296
Relational Operators.....	296
Arithmetic Operators.....	298
Functions on Real Numbers.....	299
Function Even.....	299
Function Odd.....	299

Function isInt.....	299
Function isPrime.....	299
Function isUndefined.....	299
Trigonometric Functions.....	300
Function tan.....	300
Function sin.....	300
Function cos.....	300
Function sec.....	300
Function cotan.....	300
Function cosec.....	300
Function atan.....	300
Function asin.....	301
Function acos.....	301
Function asec.....	301
Function acotan.....	301
Function sinh.....	301
Function cosh.....	301
Function tanh.....	301
Function sech.....	301
Function cosech.....	301
Function atanh.....	301
Function asinh.....	302
Function acosh.....	302
Function asech.....	302
Function acosech.....	302
Function acotanh.....	302
Function hip.....	302
Function sqrt.....	302
Function exp.....	302
Function ln.....	303
Function log.....	303
Function logn.....	303
Function power.....	303
Function root.....	303
Functions to calculate GCD, LCM and the Rest of the Numeric Division.....	304
Function LCM.....	304
Function GCD.....	304
Function remainder.....	304
Function round.....	304
Function trunc.....	304
Function truncUpper.....	305
Function fractional.....	305
Function abs.....	305
Function sign.....	305
Function randomSign.....	305
Function isPrime.....	305
Function nextPrime.....	305
Function nth_Prime.....	305
Function min.....	306
Function max.....	306
Function if.....	306
Function ifu.....	306
Function randomSign.....	306
Function random.....	307
Function chi.....	307
Function beta.....	307

Function exponential.....	307
Function f.....	307
Function gamma.....	307
Function normal.....	307
Function uniform.....	307
Function binomial.....	307
Function poisson.....	308
Function randInt.....	308
Function fact.....	308
Function comb.....	308
Functions for the Cells and his Neighborhood.....	308
Function stateCount.....	308
Function trueCount.....	308
Function falseCount.....	308
Function undefCount.....	309
Function cellPos.....	309
Function Time.....	309
Function radToDeg.....	309
Function degToRad.....	309
Function rectToPolar_r.....	309
Function rectToPolar_angle.....	309
Function polarToRect_y.....	310
Function CtoF.....	310
Function CtoK.....	310
Function KtoC.....	310
Function KtoF.....	310
Function FtoC.....	310
Function FtoK.....	310
Function portValue.....	311
Function send.....	311
Predefined Constants.....	312
Constant Pi.....	312
Constant e.....	312
Constant INF.....	312
Constant electron_mass.....	312
Constant proton_mass.....	312
Constant neutron_mass.....	312
Constant Catalan.....	312
Constant Rydberg.....	312
Constant grav.....	312
Constant bohr_radius.....	312
Constant bohr_magneton.....	312
Constant Boltzmann.....	313
Constant accel.....	313
Constant light.....	313
Constant electron_charge.....	313
Constant Planck.....	313
Constant Avogadro.....	313
Constant amu.....	313
Constant pem.....	313
Constant ideal_gas.....	313
Constant Faraday.....	313
Constant Stefan_boltzmann.....	313
Constant golden.....	313
Constant euler_gamma.....	313
TECHNIQUES TO AVOID THE REPETITION OF RULES.....	313

<i>Clause Else</i>	313
<i>11.4.2 Preprocessor – Using Macros</i>	314
11APPENDIX C – EXAMPLES.....	316
THE “ LIFE GAME”.....	316
A BOUNCING OBJECT	316
CLASSIFICATION OF RAW MATERIALS.....	318
LIFE GAME – 3D.....	320
USE OF MACROS.....	321
APPENDIX D– THE PREPROCESSOR AND TEMPORARY FILES.....	323
MANUAL OF THE VRML GUI FOR THE CD++ TOOL.....	323
<i>Install CosmoPlayer</i>	323
INTRODUCTION TO COSMOPLAYER VRML BROWSER [1].....	324
VRML GUI.....	325
<i>Info Panel</i>	325
<i>11.1.1Color Selection Dialog</i>	327
<i>Navigation Panel</i>	329
<i>Result Panel</i>	331
<i>Entity Panel</i>	334
<i>Examples</i>	334
Figure 16.....	340
Figure 17.....	340
Figure 18.....	341
Figure 19.....	341

Technical note:

The contents of this user manual includes source code and documentation written by:

Gabriel Wainer, Wenhong Chen, Juan Ignacio Cidre, Ezequiel Glinsky, Steve Leon, Pete MacSween, Ali Monadi, Alejandro Troccoli, Sankua Chao, Sameen Rehman, Jing Cao, K. Omair Muhi.

CD++

CD++ is a tool for Discrete-Event modeling and simulation, based on the DEVS formalism. It runs either in standalone (single CPU), server mode, real time mode, or parallel mode (over a linux cluster). This document is a user's guide to CD++, and we will only focus on tool-related aspects. If needed, the reader can refer to the following references for better understanding of DEVS and Cell-DEVS related topics (available at <http://www.sce.carleton.ca/faculty/wainer>):

"CD++: a toolkit to define discrete-event models". G. Wainer. In Software, Practice and Experience. Wiley. Vol. 32, No.3. November 2002. pp. 1261-1306

"A framework for remote execution and visualization of Cell-DEVS models". G. Wainer, W. Chen. In Simulation: Transactions of the Society for Modeling and Simulation International. November 2003. pp. 626-647.

"N-Dimensional Cell-DEVS". G. Wainer, N. Giambiasi. In Discrete Events Systems: Theory and Applications, Kluwer. Vol. 12, No. 1. January 2002. pp. 135-157.

"Timed Cell-DEVS: modeling and simulation of cell spaces". G. Wainer, N. Giambiasi. In Discrete Event Modeling & Simulation: Enabling Future Technologies. Springer-Verlag. 2001.

DEVS is a discrete event paradigm that allows a hierarchical and modular description of the models. Each DEVS model can be behavioral (atomic) or structural (coupled), consisting of inputs, outputs, state variables, and functions to compute the next states and outputs. Cell-DEVS allows modeling systems that can be represented as executable cell spaces. For more information about DEVS and Cell-DEVS models please refer to: <http://www.sce.carleton.ca/faculty/wainer/celldevs/introduction.html>. From now on, a complete understanding of DEVS and Cell-DEVS models is assumed. Details about the DEVS formalism can be found in:

"Theory of Modeling and Simulation". B. Zeigler, H. Praehofer, T. G. Kim. 2nd Edition. Academic Press. 2000.

To report errors in this user manual please contact: gwainer@sce.carleton.ca .

CD++Builder – Quick Reference Guide

This section is a quick reference for CD++Builder. It will explain how to run a number of simple examples without providing detailed information about the models themselves. The goal is to allow users to develop a basic familiarity with the tool and its functionality. The details of the tool are presented in the following sections.

Most of the functions of CD++ can be accessed through line commands but in this section we assume you are using CD++Builder, an Eclipse-based GUI for CD++. This tool can be downloaded at <http://www.sce.carleton.ca/faculty/wainer/wbgraf/distrib>. For username and password, please contact gwainer@sce.carleton.ca.

Eclipse (www.eclipse.org) is a workbench that can be used for any form of software development. It is an open-source integrated development environment (IDE) that can be extended through the definition of plug-ins. The desired functionality can be programmed as plug-ins by combining them with existing features of the platform. Task menus, notes, error logs, multiple project operations and even drag and drop tools/functionalities can be included. For a more detailed description of Eclipse, visit <http://www.eclipse.org/eclipse/faq/eclipse-faq.html>.

CD++Builder is a plug-in for Eclipse that provides the users with a CD++ development environment to create, edit or view CD++ simulation projects. The tool, implemented with the Eclipse workbench, provides an IDE where one can create, open and save projects. It enables editing CD++ related files and support for developing multiple projects.

CD++Builder is essentially a front-end to CD++, a toolkit for DEVS and Cell DEVS modeling and simulation (<http://www.sce.carleton.ca/faculty/wainer/celldevs/>). CD++Builder uses Eclipse and its platform plug-in development to provide an easy and simple environment to use CD++. Features such as a coupling syntax editor, C++ editing support, importing and exporting data, and a graphical user interface for the CD++ tools are featured in the plug-in. However, Eclipse has many of its own user-friendly tools that make it possible to develop anything on its workbench.

C++ support is also featured to allow users to edit C++ related files involved with a CD++ project. Creating, viewing and editing any sort of C++ file is done within a syntax editor with a coloring scheme. (C++ editing support is featured from the CDT project, <http://www.eclipse.org/cdt/>)

The CD++Builder packages all the potential tools that can be used in a CD++ project from the existing CD++ toolbox and the Eclipse workbench and its own tool set. A perspective in eclipse defines a set of editors and views arranged in an initial layout for a particular role or task. A default perspective for CD++Builder is given to support the CD++ environment. The CD++Builder plug-in must be installed to use its features.

Once CD++Builder is installed, if you start Eclipse, you will have access to the tools provided by CD++Builder. To start CD++ Builder you must first run Eclipse. To run CD++Builder follow the following instructions. Your window should look like this:

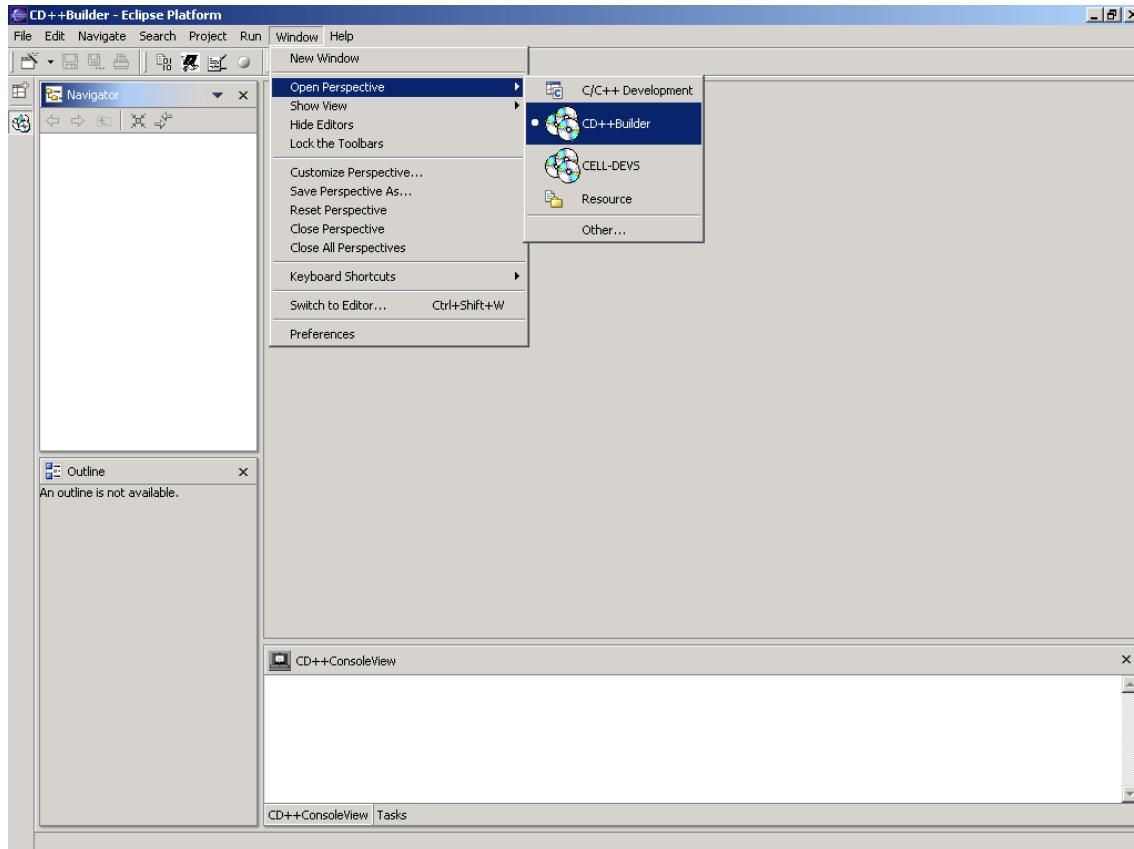
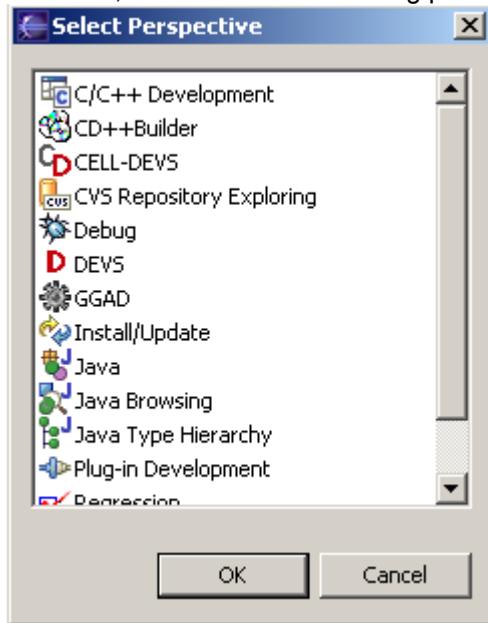


Figure 1: CD++Builder main window.

To open CD++Builder perspective, select *windows->Open perspective->CD++Builder*. If CD++Builder is not on the drop down menu, click *other*. The following panel will appear.

Figure 2: Perspective panel

Figure 3 displays all perspectives available within Eclipse. To activate the plug-in select *CD++Builder Perspective* and click *ok*. When this option is selected, CD++Builder is activated. Your window should look similar to Figure 3.



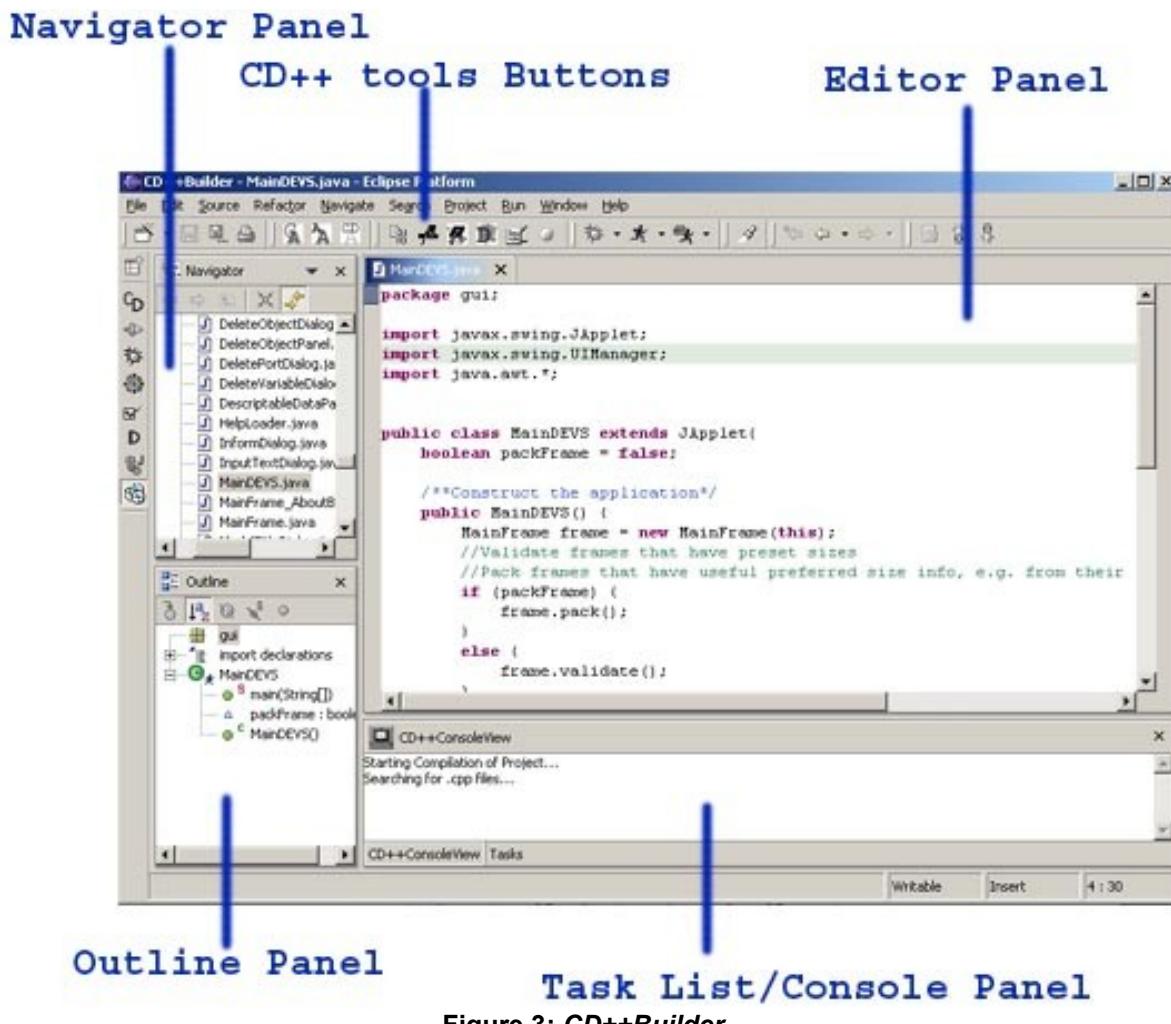


Figure 3: **CD++Builder**.

- **Navigator Panel**: allows user to view the current projects and their contents.
- **Editor panel**: allows user to view the contents of a selected file. It is programmed to open the default editor for that particular file.
- **Task list/Console panel**: a section to write down planned tasks for particular project. The task view also shows the errors encountered when compiling the project. The console will display any errors encountered while running a CD++ function and output from CD++ tools.
- **Outline panel**: outlines the functions and objects in a selected class file. This portion is only implemented for C++ files.

The CD++ tool set has a variety of components to execute DEVS models and to analyze simulation results. Our main perspective, called CD++Builder, integrates the main components as buttons located in the top toolbar. There are currently four buttons available:

- **Build**: This button automatically creates a makefile for a specific project and runs the *make* command to compile the source code for the models. The result is an executable to run a simulation.
- **Simulation**: This button activates the CD++ simulator. This executable represents a project-specific simulation program that will simulate what you are modeling.
- **Drawlog**: This button generates a (.drw file for easier visualization of the execution of a Cell-DEVS model in a text file.

-  **CD++Modeler:** This button loads the CD++ Modeler program, a graphical tool for designing and executing DEVS and Cell-DEVS models. In this application you can design atomic and coupled models as well as animate the simulation results.

Model Examples:

In this section, we will illustrate how to use the toolkit by executing two previously existing models. The first one, called *life*, is a Cell-DEVS version of the “Life” Game. The second example is called *ATM*, which is a DEVS model representing an ATM machine. These example (and other existing models) can be downloaded from <http://www.sce.carleton.ca/faculty/wainer/wbgraf>. Follow these instructions to download models.

- Click on *Model Samples* (Located on the left navigation pane). A Page of Samples should appear on the main page.
- Scroll through the list to find *ATM* and *LIFE*.
- Click on *Download Model and Sample*.
- Save the example files on hard drive.

Life Model:

The first step to start a simulation is to create a new project. Open Eclipse and make sure that the CD++Builder perspective is open (if not open, refer to the instructions earlier in this section). Click on the “File” tab, select “New”, and click on “Project...”. This will bring up a new project wizard panel as shown in Figure 4(a). Select “Other”, and on the right side of the project wizard panel, select “CD++Builder Project Wizard”. Click “Next” to start the creation of a CD++Builder project.

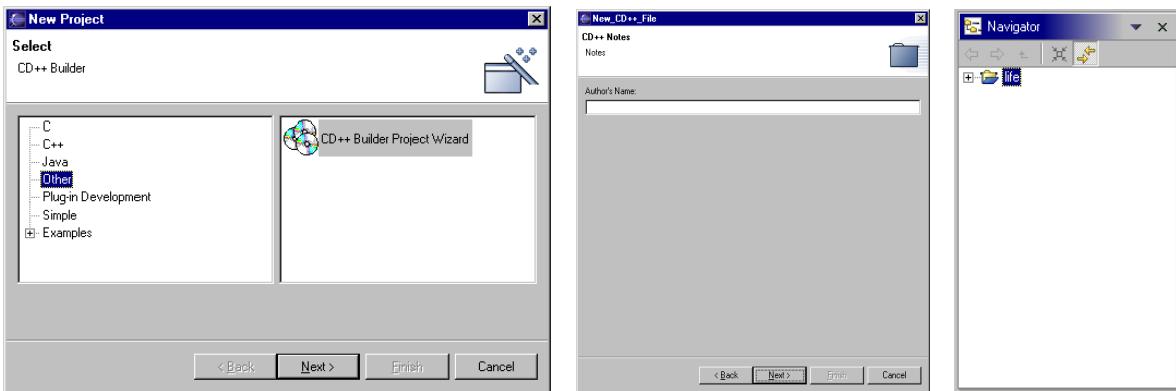


Figure 4 (a) New project wizard panel; (b) CD++Builder Project Wizard; (c) Navigator.

At this stage, the CD++Builder Project Wizard will be opened and the author name will be asked, as showed in Figure 4(b). Enter the project author's name and click *Next*. The wizard will ask for the project name. Enter *life* and click on *Finish*. The newly created project can be seen in navigator view, as illustrated in Figure 4(c).

After having created a new project, the next step is to add the *life* example model to the project. Select the project on the navigator view and right click on it. On the menu [Figure 5(a)], select “Import...”. This will open the panel shown in Figure 5(b).

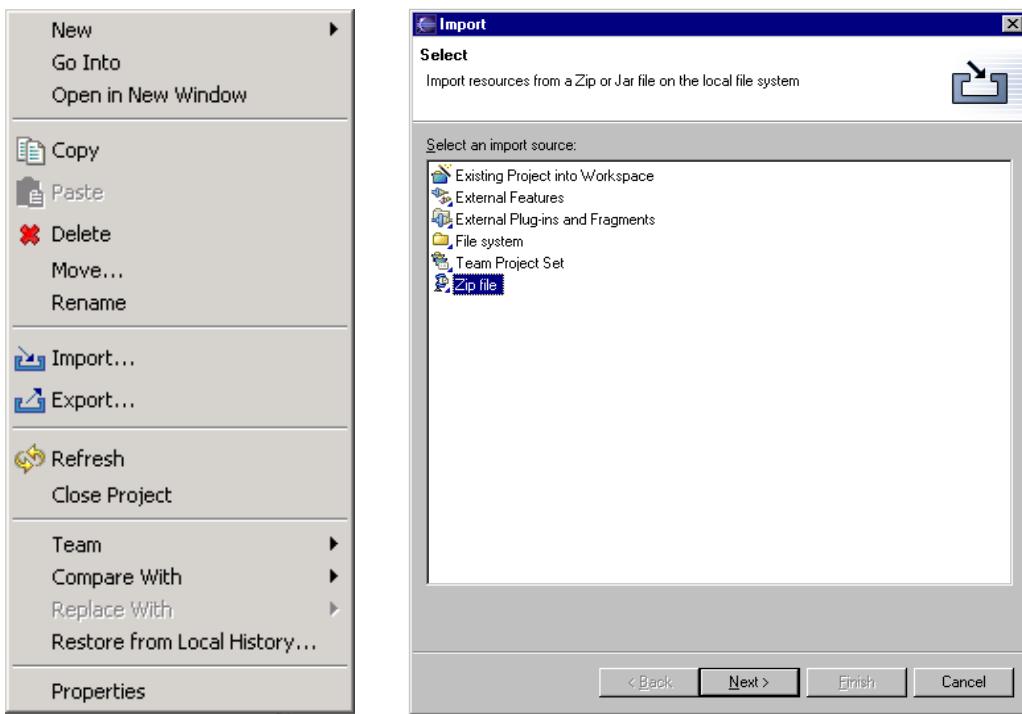


Figure 5: (a).Project Options Menu.

(b). Import panel windows.

Since the Life example model is compressed in a Zip file, select *Zip file* and click *Next*. Another panel [Figure 6] will open asking to enter the name of the zip file to be imported. To locate the zip file click on the *Browse...* button located on the top right of the panel, open *life2d.zip* and click *Finish*.

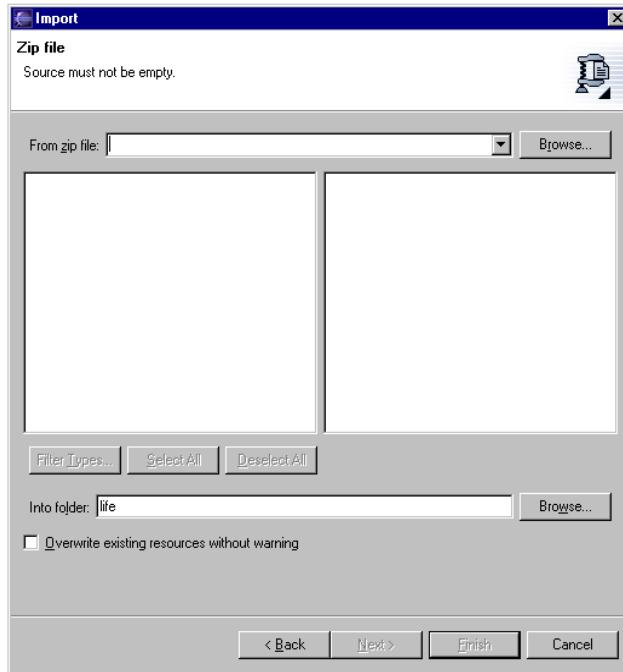


Figure 6: Import panel windows.

Figure 7(a) shows the navigator view, where you will see the new life example model that was added to the project. We will now simulate this model. The first step is to select any file from the folder by clicking on it. (THIS STEP IS REQUIRED; as there are different folders for different projects, and this step determines which file is to be run). The folder containing the file must be open, and the required file must be selected. Then click on the *simulation* button  and the panel in Figure 7(b) will open.

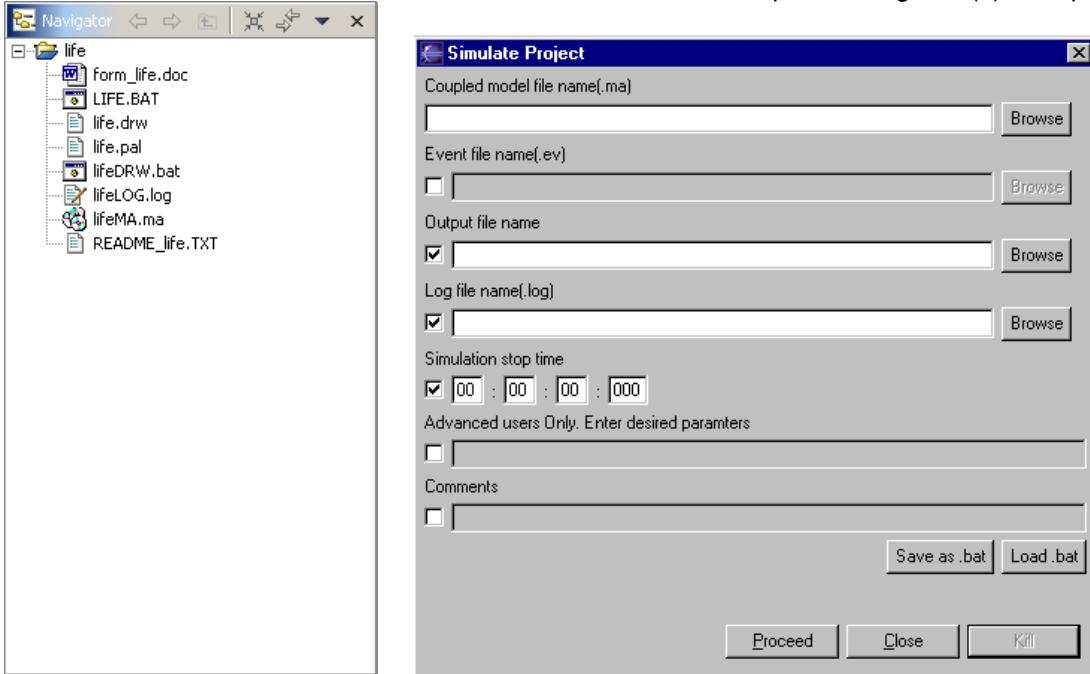
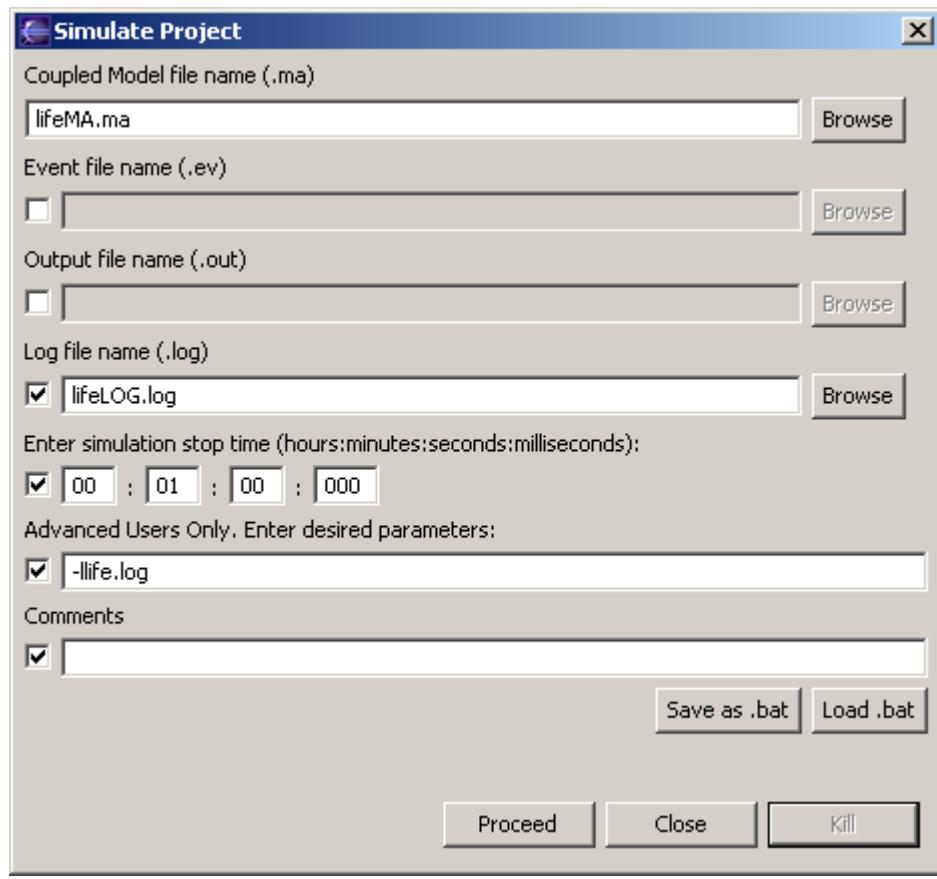


Figure 7: (a) Navigator view;

(b) Simulation panel.

There are different ways to run a model, which will be discussed later. Here, we will use a previously defined script. To do so, we must click on the “Load .bat” button, and open the file *life.bat* (included in the *life2d.zip* you originally downloaded). This will fill the panel with the necessary parameters to run the simulation [Figure 8(a)]. Click on *Proceed*. At this stage, the console view will show all the details of the simulation [Figure 8(b)].



(a)

```

CD++ConsoleView
C:\eclipse\workspace\{a}\LifeOld>simu.exe -m'LIFE.ma' -o'LIFE.out' -l'LIFE.log'
-t'00:00:00:000'
N-CD++: A Tool to Implement n-Dimensional Cell-DEVS models
-----
Version 2.0-R.45 December-1999
Daniel Rodriguez, Gabriel Wainer, Amir Barylko, Jorge Beyoglonian
Departamento de Computacion, Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires, Argentina.

Loading models from LIFE.ma
Loading events from
Message log: LIFE.log
Output to: LIFE.out
Tolerance set to: 1e-08
Configuration to show real numbers: Width = 12 - Precision = 5

```

(b)

Figure 8 (a). Parameters from life.bat (b). CD++ Console View

To see the log file created from this simulation, double click on life.log from the navigator view [seen in Figure 7(a)]. This will open an editor view showing the content of the log file [Figure 9].

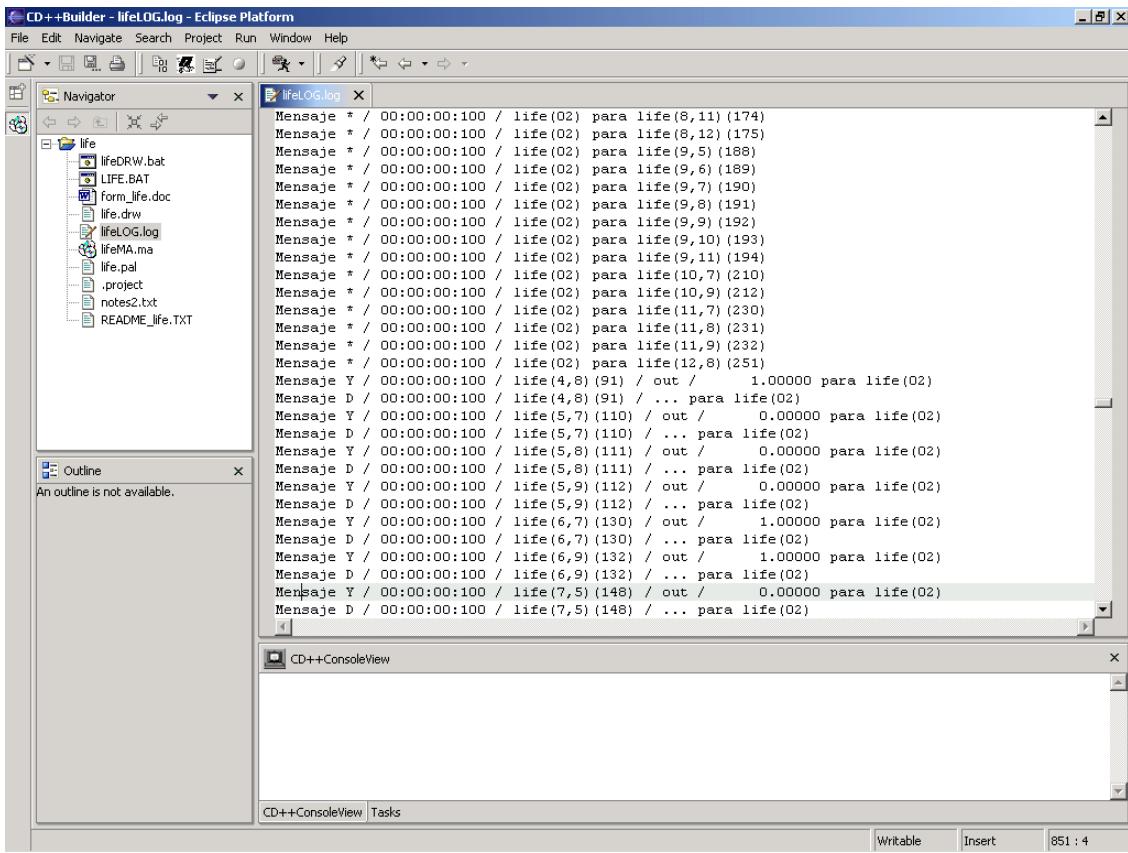


Figure 9: CD++ log outputs.

To view the state of a Cell-DEVS model, we can use the *Drawlog* tool which permits viewing the outputs in a simpler way. To use this feature, click on the *Drawlog* button . The following panel will open [Figure 10]:

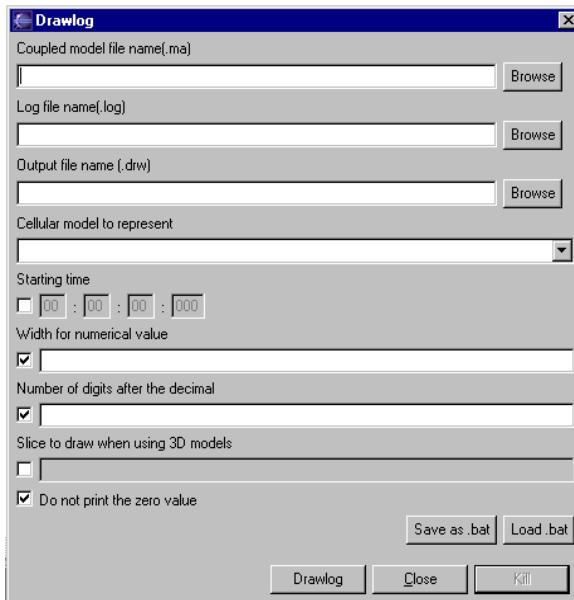


Figure 10. Drawlog panel

Here, we will also use a previously defined script. Click on the “Load .bat” button, and open the file *lifedraw.bat* (included in the *life.zip* you originally downloaded). This will fill the panel with the necessary parameters to run the drawlog. Click on *drawlog*. The CD++ console view will once again show the state of the drawlog creation. Once the conversion finishes, a text file will be created and opened illustrating the state of the simulation, as in Figure 11.

```

Line : 840 - Time: 00:00:00:000
01234567890123456789
+-----+
0|   |
1|   |
2|   |
3|   |
4|   |
5|      111
6|   |
7|      1 1 1
8|      1 111 1
9|      1 1 1
10|
11|      111
12|
13|
14|
15|
16|
17|
18|
19|
+-----+
Line : 1086 - Time: 00:00:00:100
01234567890123456789
+-----+
0|   |
1|   |
2|   |
3|   |
4|      1
5|
6|      1 1
7|      11 11
8|      1 1 1
9|      11 11
10|      1 1
11|
+-----+

```

Figure 11: *.drw file

ATM Model:

Step 1. Open Eclipse and make sure that the CD++Builder perspective is open (if not open, refer to the instructions earlier in this section).

Step 2. Click on **File → New → Project**. This will bring up a new project wizard panel as shown in Figure 4(a). On the left side of the project wizard panel select “Other”, and on the right side select “CD++Builder Project Wizard”.

Step 3. Enter the project author's name and click “Next” then click “Finish”.

Step 4. Enter *atm* as the project name and click on *Finish*. The newly created project can be seen in navigator window, as illustrated in Figure 4(c).

Step 5. In the navigator window right-click on the project titled “atm” and select import. The import panel should pop up shown in Figure 5(b). Select “zip”, click next and a new window will popup shown in Figure 6.

Step 6. Click on the *Browse* button located on the top right of the window [shown in Figure 6] browse to and open *atm.zip* and click *Finish*.

Step 7. Since the ATM model is a DEVS model, it needs to be compiled before simulating it. To compile the project, **click on one of the files in the project within the navigator panel** (in order to select the project we want to compile) and press the *Build* button . A new panel, shown in Figure 12(a), will appear asking the user if they would like to run in verbose mode (a mode that provides detailed info of the compiling project during compilation). Click "Yes" to run in verbose mode.

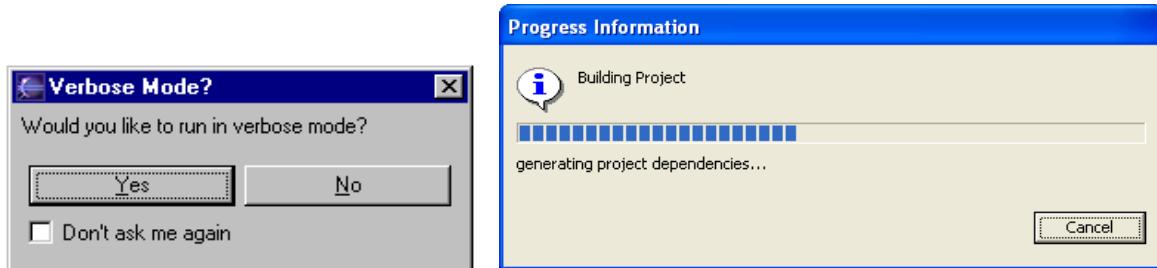


Figure 12: (a) Verbose mode panel (b) Progress dialog

Step 8. A progress dialog will appear showing the progress Figure 12(b). Also, since the verbose mode was chosen, the console view will show the details of the compilation and will inform you if it was successful. If there are errors, you can see them on the console, as showed in the following figure [Figure 13]. Once the compilation is successful, we can start simulating the project.

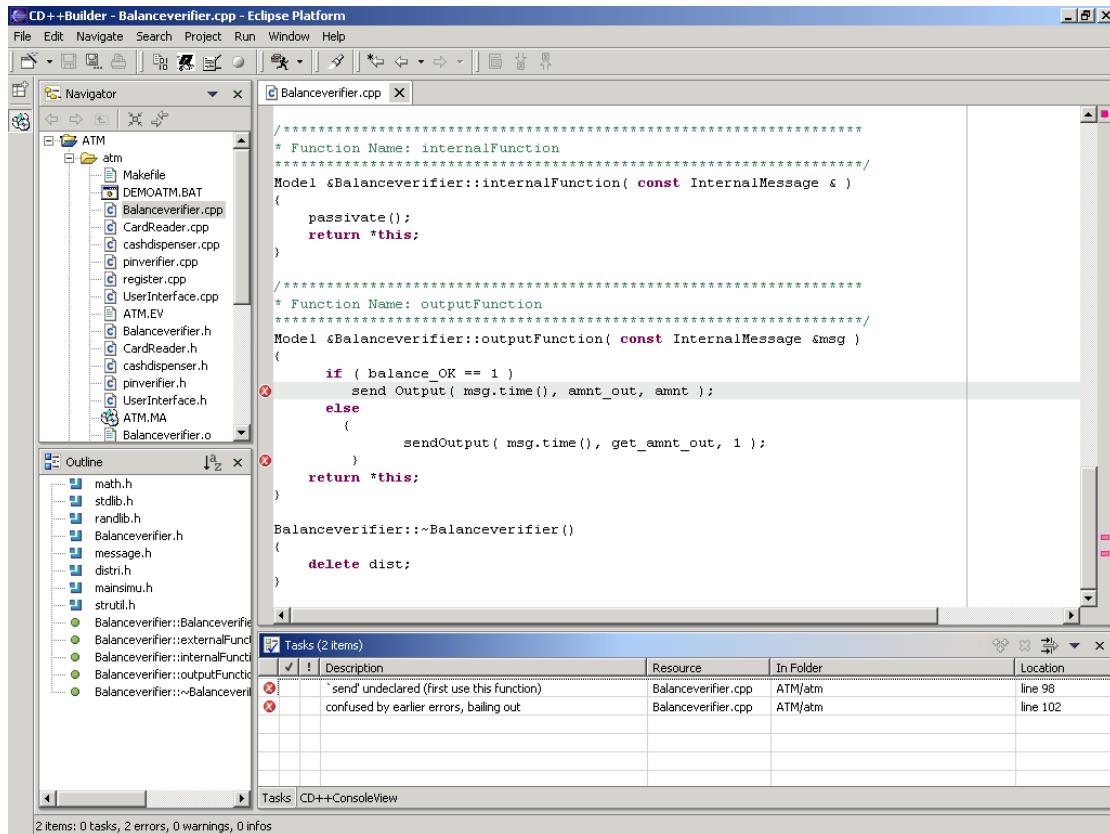


Figure 13: Error display.

Step 9. To simulate the model, the first step is to select any file from the folder by clicking on it. **(THIS STEP IS REQUIRED)**; as there are different folders for different projects, and this step determines which file is to be run). Then click on the *simulation* button  and the panel in Figure 7(b) will open.

DEVS Model definition: Atomic models

This section describes the mechanism to define and incorporate new atomic models into CD++. These models can be used to interact directly with other models or to be part of a DEVS coupled model. Atomic models are added to the tool at compile time, and if a new atomic model needs to be defined; it must be coded in C++ and incorporated into the CD++ model hierarchy.

Creating new atomic models using CD++Builder

Eclipse provides a set of tools and plug-ins, which facilitate the creation of a model. This section will provide useful features that Eclipse delivers to create a new atomic model.

Opening an existing project

To open an existing project into your workspace, you will have to import it by using the import wizard. Follow these steps:

Step 1: Select *File* on the main menu of Eclipse. Click on *Import*. The panel in Figure 14(a) will appear.

Step 2: Select *Existing Project into Workspace*. Click *Next*. The panel in Figure 14(b) should appear.
 Step 3: Browse to find the folder containing the project you want to import. Click *Finish*.

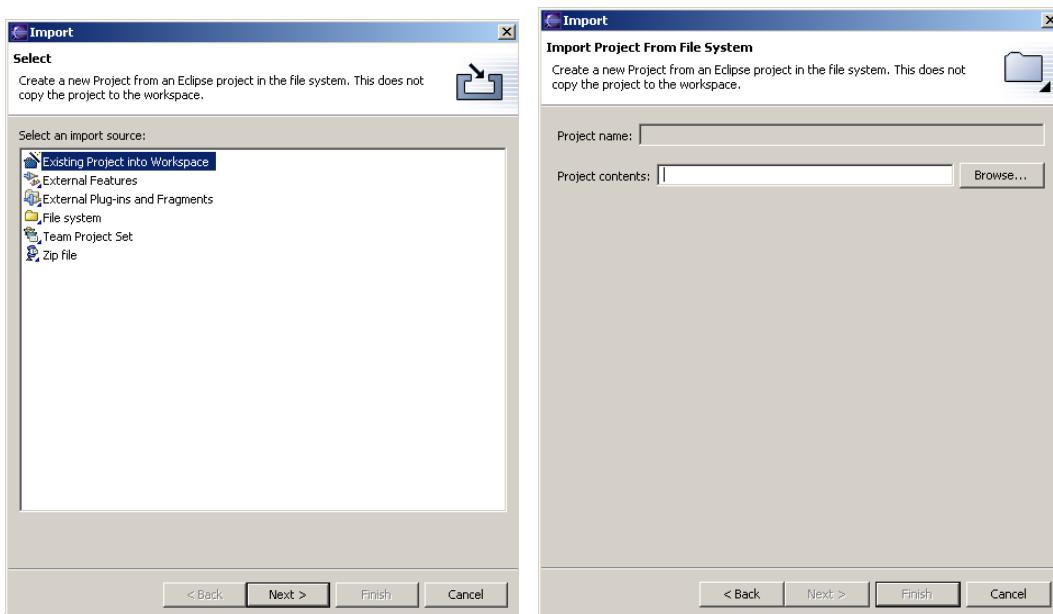


Figure 14 (a). Import wizard panel (b) Browse for existing project

The project will now be in your workspace/project navigator for you to edit or view.

Creating a new project

To create a new project you have to use the new project wizard. Follow these steps:

Step 1: Select *File* on the main menu of Eclipse. Select *new*, and then click on *project*. The panel in Figure 15(a) will appear.

Step 2: Select *Other* on the left pane. Select *CD++ Builder Wizard* on the right pane. Click *Next*. The panel in Figure 15(b) should appear.

Step 3: Enter *Author's Name* (typically your name). Click *Next*. The panel in Figure 17(a) will appear.

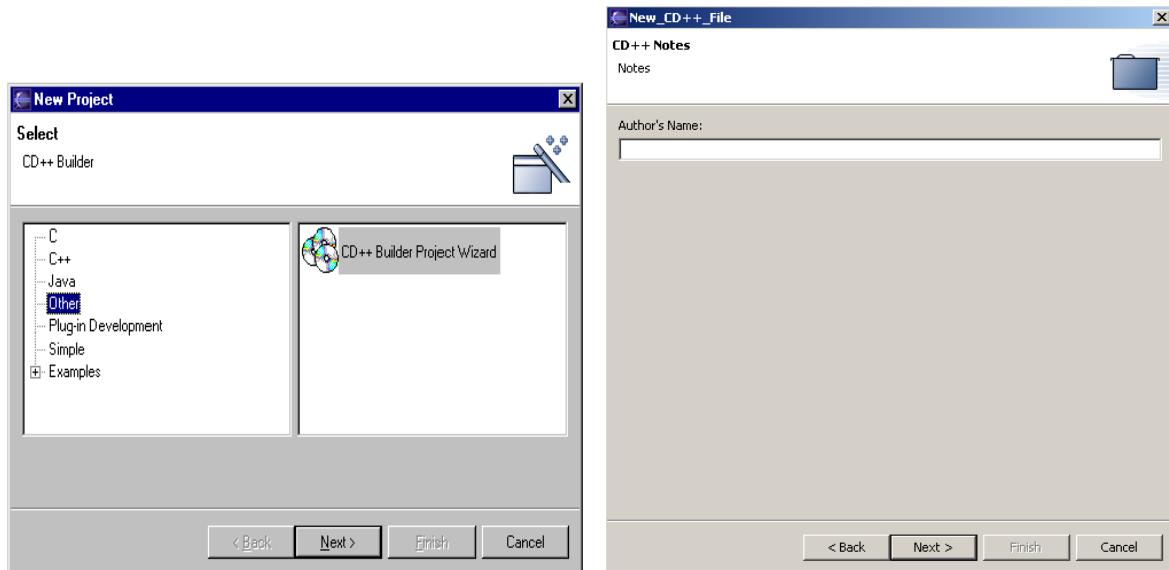


Figure 15 (a). New project wizard panel (b) Author's Name panel

Step 4: Enter the name of your project. At this point, *Use Default* should be checked. If you are ready to finalize your project, click *Finish*. If you want to make references to existing projects in your workspace, click *Next*. This will allow you to include models already defined for other projects in the newly created one. The panel similar to Figure 17(b) will appear.

NOTE: To save your workspace on the network drive [or at a different location], uncheck *Use Default* now click the *Browse* button (see Figure 16) to locate the folder in which you will save your project.

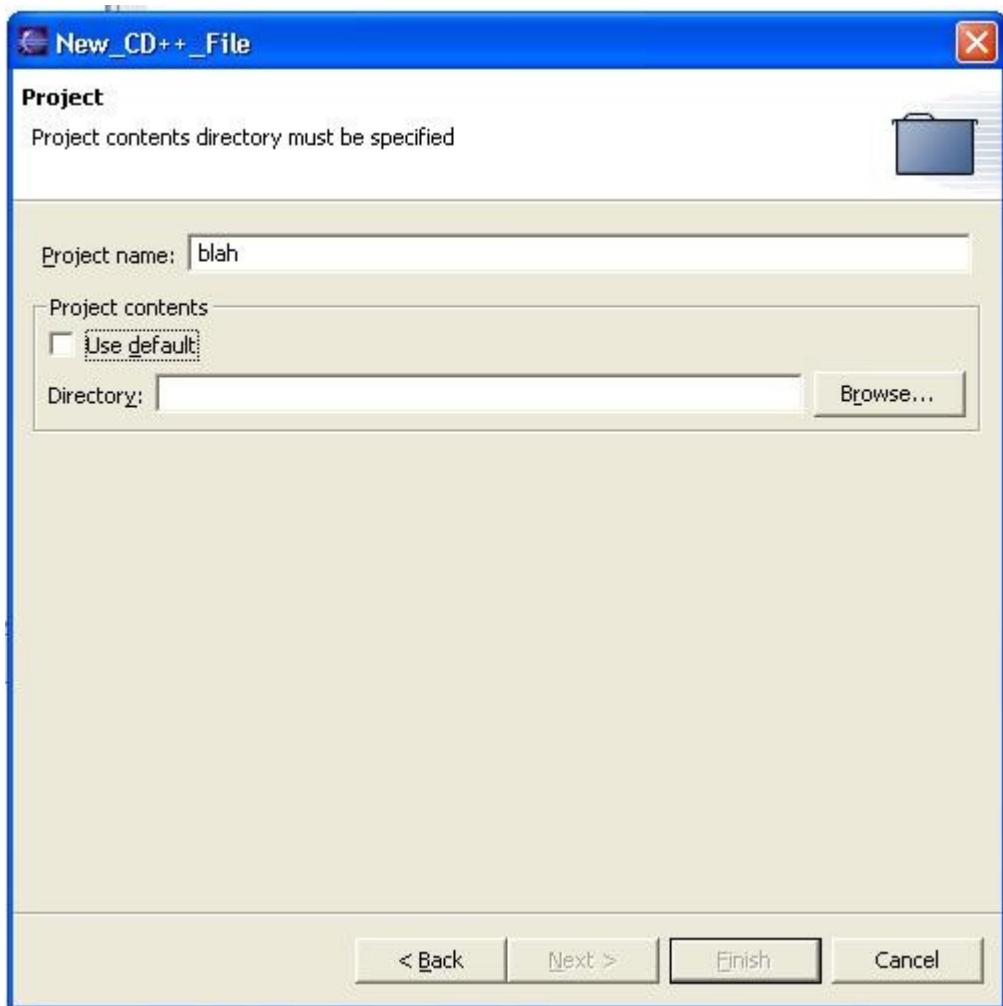


Figure 16: Choosing the directory to save the project.

Step 5: Select the checkboxes next to the projects you want to make references to. Click *Finish* to finalize.

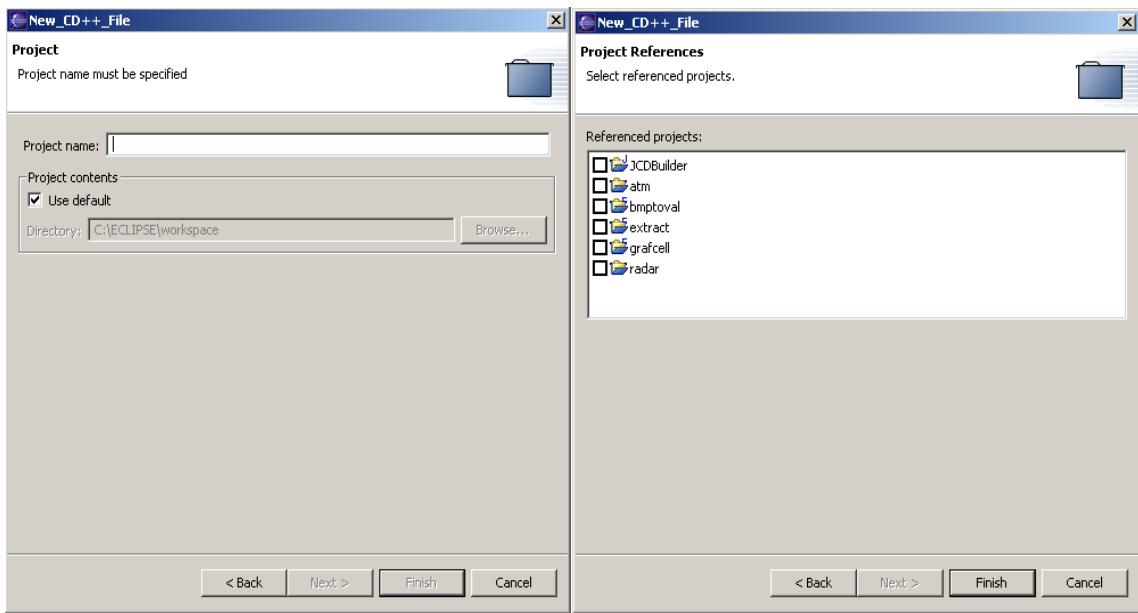


Figure 17 (a): Enter Project Name (b): Referencing existing projects

Your project is now created and can be viewed in the navigator window. You are able to access a *notes* file and a *project* file. The *notes* file is a simple text file that you can store short notes about your project. The *project* file is a file that acts as an identification file for the project. Figure 18 shows a newly created empty project *Clock*.

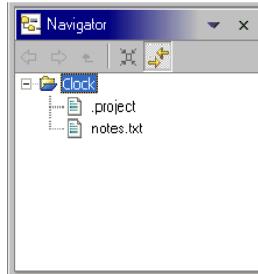


Figure 18: New project shown in the navigator view

Creating and editing files

To create a new atomic or coupled model, we need to create, and then edit a new file. To create a file, highlight the folder (in the navigator view) you want the file to belong to. From the main Eclipse menu, select *File* -> *New* -> *File*. This will bring you to a new file wizard that will request information about the type and location of the file as illustrated in the figure below:



Figure 19: New file wizard panel

Highlight the project where you want the file created and then enter the name in the bottom panel labeled *File Name*. Before you click on finish you must enter the extension of the file you are creating. For example if you are creating a C++ DEVS model, you must end the file in “.cpp”

As soon as you select finish, the file is created and opened in the editor panel. Figure 18 illustrates the creation of a new file called *newFile*.

Eclipse has a “smart editing” functionality that will open the appropriate editor in the editor panel. When you select any file in your project for viewing or editing, it opens it in the editor panel to the affiliated editor. C++ files will be opened in CDT (Eclipse’ C++ Editor), text files will be opened in a basic text editor etc. If you edit a file you can simply hit the save button in the top left corner, if you feel you made a mistake, you can close the file and select “no” to not save it at all.

In Figure20, the file that we just created, *newFile*, is shown in the navigator view and it is also opened for editing.

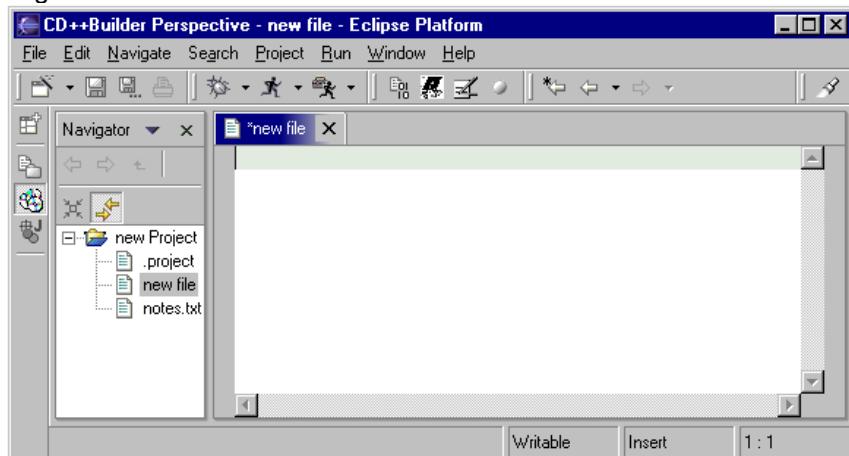


Figure20: CD++Builder with text editor opened.

Adding Files to projects

Files can be added to a project in multiple ways. One option is to drag and drop files from any

window outside of eclipse into your project (within the navigator window) which automatically copies that file into the project.

Another method is to use the import function. Select *File* and then click *import*. This will bring up the importing panel shown in Figure 21.

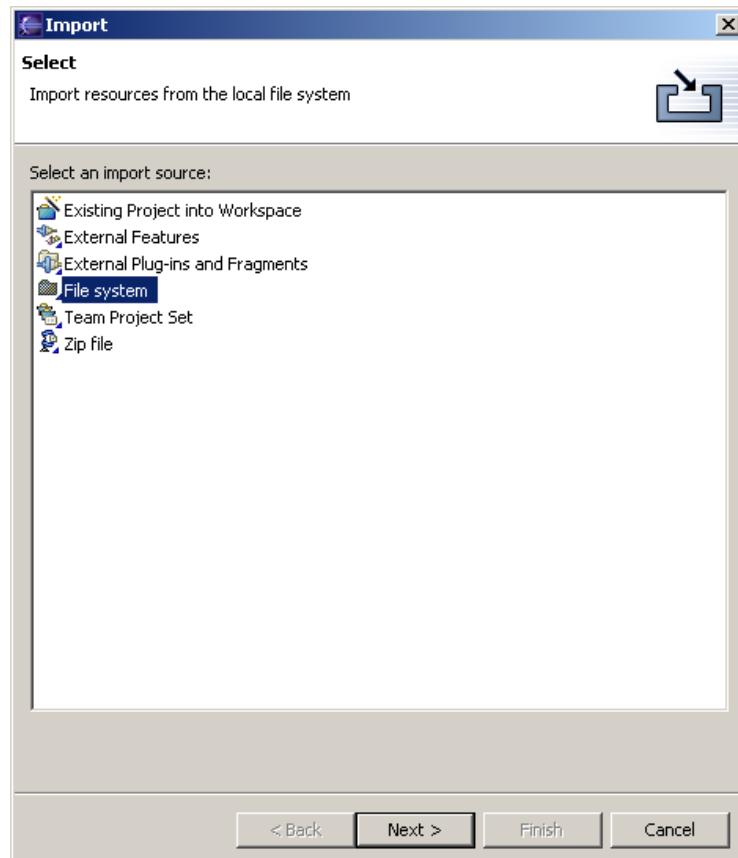


Figure 21: Import Wizard Panel

Here, select *file system* and then click *next*. The panel shown in Figure 22(a) will appear. On this screen you must specify the directory of the files you want to add by pressing the *Browse* button that is located at the top of the panel. When you click on *Browse*, the window in Figure 22(b) will appear. Select the folder containing the files you want to copy into your project. Click *OK*. The window from Figure 22(a) will now contain the name and subfolders of the folder you selected. If you don't want to copy the entire folder, you have the option to choose only selected files within the folder. Once you are done selecting the files you want to copy, click *Finish*.

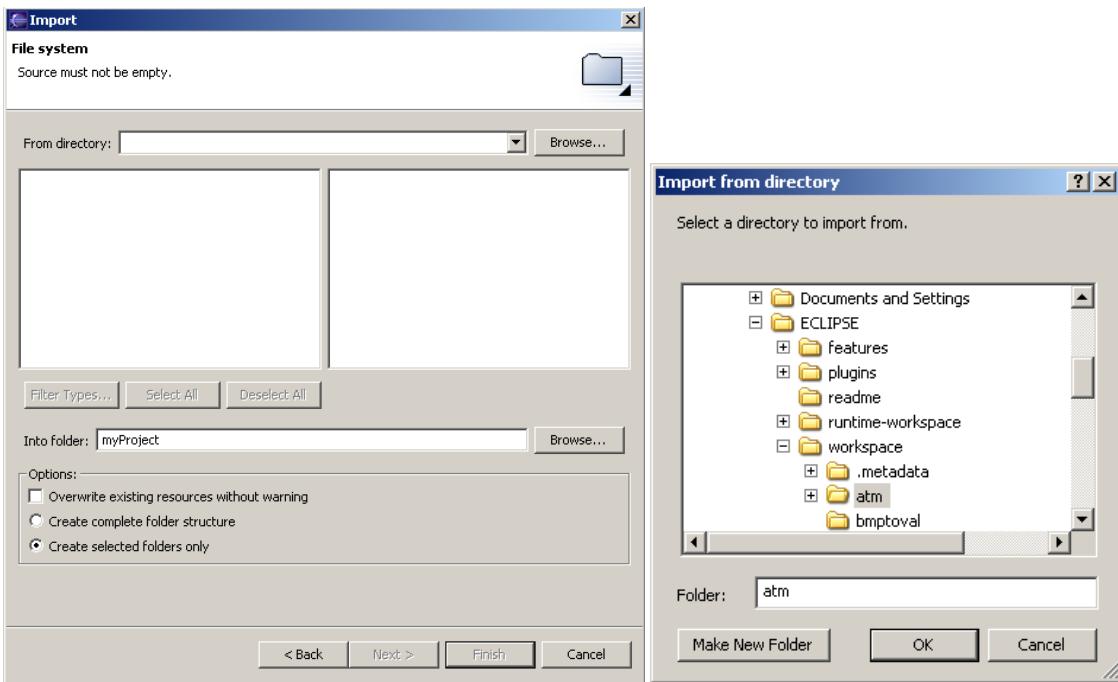


Figure 22 (a): File System Import Panel (b) Browse to select file system

One can also import files in the form of a zip file. On the Import Wizard panel in Figure 21, select “Zip File”. Click Next. The panel in Figure 23(a) appears. Click Browse. Browse to select the zip file you want to import. Once you’ve chosen the zip file to import, your panel will resemble Figure 23(b).

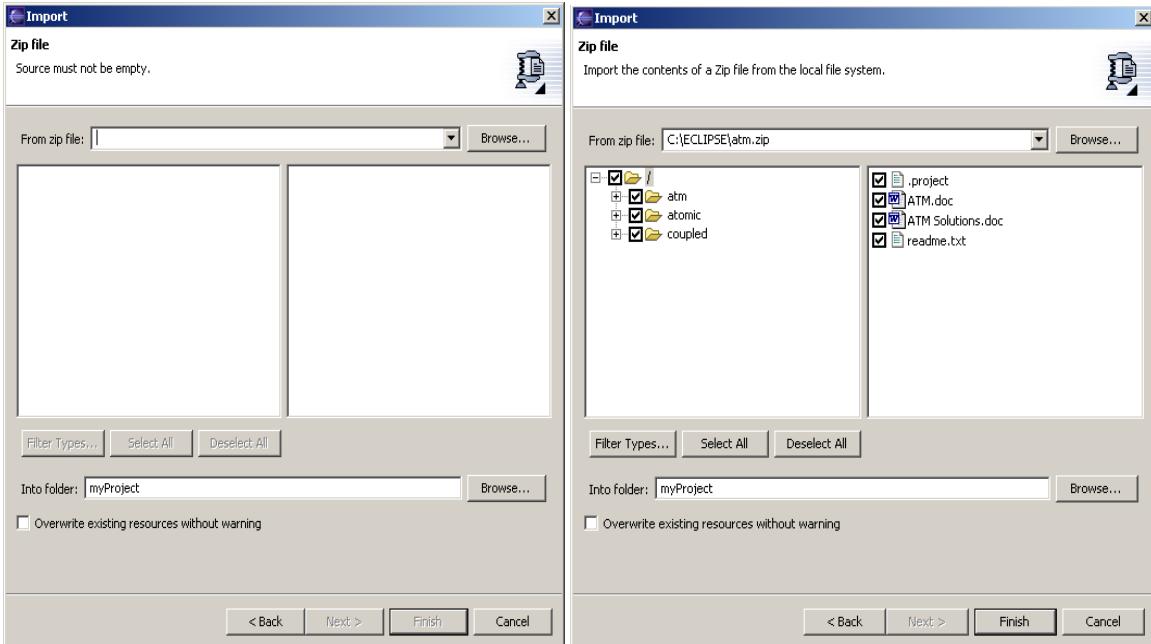


Figure 23 (a): Zip File Import Panel (b) Zip File details

Once again, you have the option to import all the files in the zipped folder, or you can select only the files you want to import, by checking the check box next to the name of the file.

Using the Navigator

The Navigator (the panel on the left side) is a component that enables you to view multiple projects and their indexes of all the files in each project. The format for viewing these files are in tree-fashion as shown in Figure 24. You can open the index of each project and view the files or folders in it. Opening a folder inside the project will expand that section to show its contents. You can move or copy files from project to project by using the drag and drop option or the import wizard option.

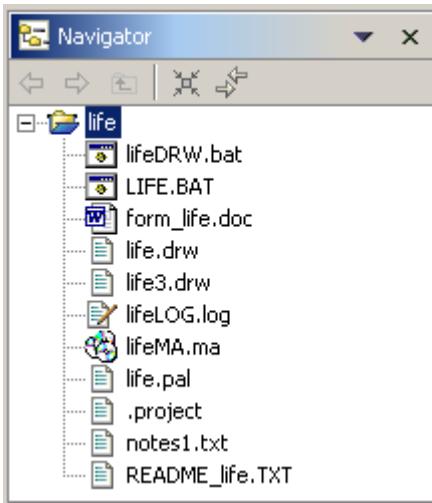


Figure 24: *Navigator view*

The navigator can also close projects that you may not want to view or use, but keeps a closed folder in case you may want to reopen it again. To close a folder, right click on the project you want to close, and select *close project*.

The arrow pointing down (next to the close button, on the top right corner) provides the user with options and settings for the projects. The sort option lets you sort the files by name or type. The filter option can be used to filter out specific file types. To do this click on the button with the arrow pointing down (next to the close button) and select *filters*.

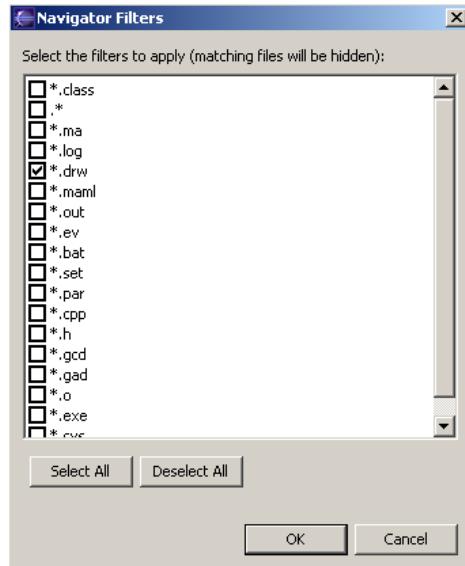
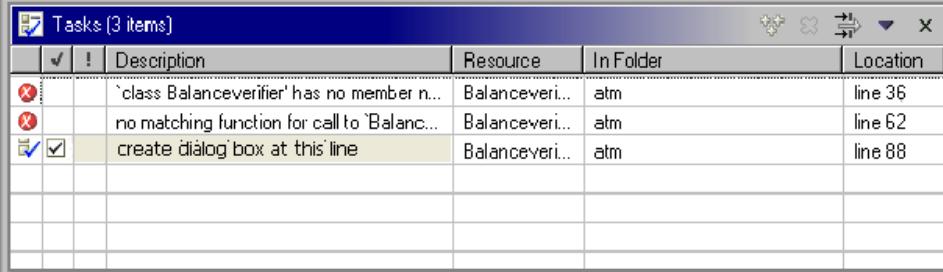


Figure 25: *Filter selection panel*

You can select which file types you want to filter out by checking off its box [Figure 25]. After selecting *Ok*, you will notice the navigator will show files that you have not checked.

Using the task list

The task list (on the bottom section of the CD++Builder perspective [seen in Figure 26]) is used to organize any notes, reminders or directions you have on your project. You can use it to leave notes on any line of code or even a reminder for a task you want to do later. It is weaved together with all editors supported in Eclipse to place task markers on any line of code. This can be done by opening a file and right clicking on the far left bar of the editor panel and selecting *add task*. Here you can enter the description of the task and its priority level. When you click on *Ok*, you can see that your task has been added to your task list. Double clicking on this task will automatically open up the file and bring you to the line of code.



	Description	Resource	In Folder	Location
	'class Balanceverifier' has no member n...	Balanceveri...	atm	line 36
	no matching function for call to 'Balanc...	Balanceveri...	atm	line 62
<input checked="" type="checkbox"/>	create dialog box at this line	Balanceveri...	atm	line 88

Figure 26: Task list view

To add a generic task that may not be code based, right click on the task list and select *new task*. This will prompt you with the similar panel as above asking you for the task description and its priority.

Another useful feature of the task list is to show errors generated by using the *Build* button to compile the C++ files. The errors are shown in the task list. Double clicking on one of them will automatically open up the file and take you to the line of code where the error occurred.

Using the Outliner

The outline shown in Figure 27 is used for C++ files. When you have opened a C++ file and it is the active part in the editor. The outliner outlines all the variables, functions and dependencies of that particular class. This can be used for quick referencing by double clicking on a function or variable it will bring you immediately to its location in the source code.

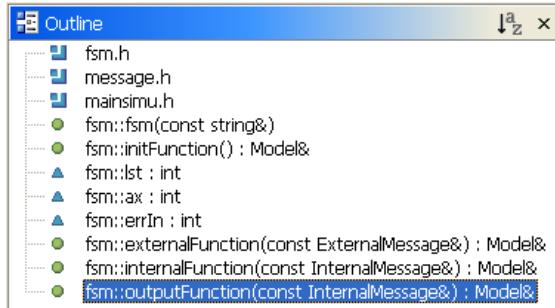


Figure 27: Outline view of fsm.h

Defining Atomic models in CD++

We will now show how to make use of the features mentioned previously by building an atomic

model from scratch. These files can be created in CD++Builder as explained in Section or a standard text editor.

If created in a standard text editor, these models can be executed in a Linux/Cygwin environment. It may be noted, CD++Builder has not yet been tested in a Linux environment. For installation under Cygwin, refer to Section Appendix A.

If created according to method described in Section , the sample model to be created is a device of temporary storage that uses a FIFO (First In First Out) mechanism. The source code of this model comes included with the CD++ toolkit. When you download the original zip files, you will find the queue.cpp and queue.h files containing the models discussed in this section. In CD++Builder, you will find it in the ...\\eclipse\\plugins\\CD++Builder_1.1.0\\internal folder.

Adding new models

The first step when implementing a new atomic model is to define a class derived from *Atomic* overloading the methods for transition handling. These methods are not public since only the class *Atomic* can invoke them. *Atomic* is an abstract class that declares a model's API and defines some service functions the user can use to write her model. The class *atomic* provides a set of services and requires a set of functions to be redefined. The services are functions that allow the model to tell the simulator the current state and duration.

These are the steps to add a new atomic model:

- a) Write a class derived from *Atomic* **overloading** the following methods:
 - ***initFunction***: Before calling this method, the sigma value is infinite and the state is passive.
 - ***externalFunction***: Called when an external event arrives in one of the model's output ports.
 - ***internalFunction***: Before calling this method, the sigma value is zero because the interval to the internal transition has expired.
 - ***outputFunction***: Called when an internal event arrives, before calling the internal transition function.
 - ***className***: the class name.
- b) Modify **register.cpp**, adding to the method Simulator::registerNewAtoms() the new atomic model, in the Queue example, we can see:

```
SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Queue>(), "Queue" )
```

Interacting with the Simulator

In each of the former methods you have to use some primitives to interact with the simulator in order to accomplish the common atomic model operations.

These primitives are:

- ***holdIn(state, time)***: The model changes its state into *state* by time *time* and when this interval expires a change of state has to occur. The state could be: *active* *passive*.
- ***passivate()***: The model change its state to *passive* and it will only be activated when an external event arrives.
- ***sendOutput(time, port, value)***: It sends an output message.

- **nextChange()**: It informs the time remaining before the next change of state (*sigma*).
- **lastChange()**: It informs the time of the last change of state.
- **state()**: It informs the model's state.
- **getParameter(modelName, parameterName)**: It gets the parameter *parameterName* value.

Since *Model* is an abstract base class, it defines the interface for message exchange. *Atomic* and *Coupled* classes are the only ones that can receive and send messages. The derived classes are responsible for overloading the initialization, internal transition, external transition and output methods.

The *Atomic* derived classes should not send any kind of message, except for the output values informed through the *sendOutput* method. The *Atomic* class is responsible for sending the Y and D messages to their parents using the *sigma* and *state* values.

An example: a model of a queue

Our model of a queue will hold any type of user defined value. The queue will have three input ports and one output port. Values to be stored will be received through the input port *In* and will later be sent through the port *Out*. The input ports *start-stop* and *next* will serve to regulate the flow of values through the output port. Figure 28 shows the structure of our model of a queue.

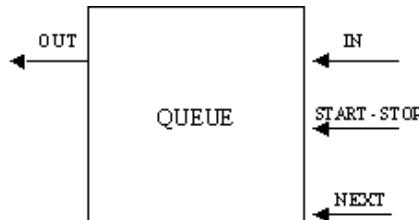


Figure 28: Structure of a Queue

Initially, the queue is empty. When the first value is received through the input port *In*, it will be stored in the queue and forwarded through the output port *Out* after a time defined by the user (*parameter preparationTime*). If a value is received and the queue is not empty, then it will be stored, but it will not be forwarded immediately. Instead, it will be sent through the output port *Out* only after a message is received through the port *next*.

A message received through the input port *start-stop* will temporarily disable the queue. If the queue is disabled, it will only respond to new events received through the input port *In*. Any value received will be stored, but no output will be ever sent until the queue is enabled again by sending an event to the *start-stop* port.

After this brief description, we are ready to begin writing our model. In CD++Builder, create a new project. Once the project is created, create a new .cpp file. First, we need to define a class to store the state of the queue. The queue will have two state variables: a list of elements and a Boolean to store the enabled/disabled status. Figure 29, lists the Queue state class declaration and definition.

```

class Queue : public Atomic {
public:
    Queue(); // Default constructor
    virtual string className() const;

protected:

```

```

Model &initFunction();
Model &externalFunction( const ExternalMessage & );
Model &internalFunction( const InternalMessage & );
Model &outputFunction( const InternalMessage & );

private:
    const Port &in;
    const Port &stop;
    const Port &done;
    Port &out;
    Time preparationTime;

    typedef list<Value> ElementList ;
    ElementList elements ;

    Time timeLeft;

}; // class Queue

```

Figure 29: Queue.h: model definition.

The example shows the variable definitions referring to the queue's ports and the time it takes to prepare the value before sending it through the out port. It also shows the definition of the list of values to hold the input data (*ElementList*) and the time remaining when the model is interrupted by a flow control signal (*timeLeft*).

The constructor (Figure 30) creates the input and output ports of the model and sets the variable *preparationTime*. The parameter *preparation* must be specified in the configuration file.

```

Queue::Queue()
    : preparationTime( 0, 0, 10, 0 )
    , in( this->addInputPort( "in" ) )
    , stop( this->addInputPort( "stop" ) )
    , done( this->addInputPort( "done" ) )
    , out( this->addOutputPort( "out" ) )
    {
        this->description( "Queue" );

        string time( Simulator::Instance().getParameter(
            this->description(), "preparation" ) );

        if( time != "" )
            preparationTime = time ;
    }

```

Figure 30: Queue.cpp: model constructor.

The *getParameter* method queries the coupled model file (*.ma file, described in the following section) and identifies the “preparation” parameter. The value of the parameter (a string), is converted into the initial preparation time.

The initialization function erases the queued data list (Figure 31). The following state change will take place when an external event arrives, which is why *sigma* remains constant. If you wish to modify the next state change use the *holdIn* method.

```

Model &Queue::initFunction(){
    elements.remove( elements.begin(), elements.end() ) ;
    return *this;
}

```

Figure 31: Queue.cpp: model initialization function.

The Queue class has three input ports through which it can receive external events (Figure 32). An event that arrives in the **in** port represents a new input value, which has to be queued. If it is the only one in the queue it has to be retransmitted immediately. Thus we schedule our internal event after the preparationTime, which represents the delay of the queuing done. An event that arrives in the port **done** indicates that the last element sent has been received and therefore it has to be erased from the queue. If there were more elements to be sent the first value in the queue should be prepared to be released. An event that arrives in the port **stop** indicates that the flow should be stopped or restarted. If the queue was in *active* state and the message value is not zero the queue will execute a working pause, here the time remaining to process the next state change is calculated (end of preparation time) and then the queue changes its state to *passive* calling the *passivate* method. If the queue was in *passive* state and the message value is zero then the queue restarts the work setting the next state change to the remaining processing time.

```
Model &Queue::externalFunction( const ExternalMessage &msg ){
    if( msg.port() == in ) {
        elements.push_back( msg.value() );
        if( elements.size() == 1 )
            this->holdIn( active, preparationTime );
    }

    if( msg.port() == done ) {
        elements.pop_front();
        if( !elements.empty() )
            this->holdIn( active, preparationTime );
    }

    if( msg.port() == stop )
        if( this->state() == active && msg.value() )
        {
            timeLeft = msg.time() - this->lastChange();
            this->passivate();
        }
        else
            if( this->state() == passive && !msg.value() )
                this->holdIn( active, timeLeft );

    return *this;
}
```

Figure 32: Queue.cpp: external transition function.

When the preparation time interval expires the outputFunction shown in Figure 33 is invoked and the first value in the queue has to be sent through the output port.

```
Model &Queue::outputFunction( const InternalMessage &msg ){
    this->sendOutput( msg.time(), out, elements.front() );
    return *this ;
}
```

Figure 33: Queue.cpp: output function.

After calling the output function the internal transition function shown in Figure 34 is invoked. Here there is nothing to do, except wait for the acknowledge at the *done* port.

```
Model &Queue::internalFunction( const InternalMessage & )
{
    this->passivate();
    return *this ;
}
```

Figure 34: Queue.cpp: internal transition function.

A new atomic model is created as a new class that inherits from the class *Atomic*. To tell CD++ that

a new atomic definition has been added, the model must be registered in the method `MainSimulator.registerNewAtoms()`. This method is located in the file, `register.cpp`, which is shown in the next figure (Figure 35):

```
void MainSimulator::registerNewAtoms() {
    SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Queue>() ,
"Queue" ) ;

    SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Generator>() ,
"Generator" ) ;
    SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<CPU>() ,
"CPU" ) ;
    SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Transducer>() ,
"Transducer" ) ;
    SingleModelAdm::Instance().registerAtomic( NewAtomicFunction<Trafico>() ,
"Trafico" ) ;
}
```

Figure 35: Content of `register.cpp`

Creating new atomic models for parallel simulation

A new atomic model is created as a new class that inherits from `Atomic`. To tell CD++ that a new atomic definition has been added, the model must be registered in the `ParallelMainSimulator.registerNewAtoms()` function. In addition, for an atomic model to support the TimeWarp protocol, a model's state has to be defined as a separate class that is derived from `AtomicState`. The current state is available through the function `getCurrentState()` which returns a pointer to the model state. States are managed by the Warped kernel, and are only valid through a simulation cycle. There is no guarantee a pointer returned during a simulation cycle will still be valid during the next one. In addition, the states are not created until the `initFunction` is called, so no state initialization code should be placed in the class constructor.

- **virtual Model &initFunction():**

The simulator invokes this method at the beginning of the simulation and after the model state has been initialized. All initialization should take place when this method is called. An active model should usually set the time for the next transition using the `holdIn` function. The `holdIn` function will be further explained later in this section.

- **virtual Model &externalFunction (const MessageBag &)**

virtual Model &externalFunction(const ExternalMessage &): These methods are invoked when one or more external events arrive from a port of the model. It corresponds to the \square_{ext} function of the DEVS formalism. The simulator calls the first function, the one that receives a message bag. By default, this function will iterate through all the messages in the bag and call the second one. This is provided for backward compatibility. If the modeler would like to have more control on the model's behavior when multiple simultaneous events are received, it is recommended the first function is overridden. If the model's behavior is simple enough for simultaneous events to be handled sequentially, then it will be enough to redefine the second function.

The interface for the `MessageBag` class is shown below (Figure 36).

```
class MessageBag {
public:
    MessageBag(); //Default Constructor
```

```

~MessageBag();

MessageBag &add( const BasicPortMessage* );

bool portHasMsgs( const string& portName ) const;

const MessageList& msgsOnPort( const string& portName ) const;

int size() const

MessageBag& eraseAll();

const VTime& time() const;

};

```

Figure 36: *MessageBag* class

- **virtual Model &internalFunction(const InternalMessage &)**: This method corresponds to the \square_{int} function of the DEVS formalism.
- **virtual Model &outputFunction(const CollectMessage &)**: This function is called before \square_{int} . It should send all the output event. Each output event is sent using the function sendOutput defined below.
- **virtual Model &confluentFunction (const InternalMessage &, const MessageBag &)**: It corresponds to the \square_{conf} function of the DEVS formalism. By default, it is set to the function seen in Figure 37.

```

Model &Atomic::confluentFunction ( const InternalMessage &intMsg, const
MessageBag &extMsgs )
{
    //Default behavior for confluent function:
    //Proceed with the internal transition and the with the external
    internalFunction( intMsg );

    //Set the elapsed time to 0
    lastChange( intMsg.time() );

    //Call the external function
    externalFunction( extMsgs );

    return *this;
}

```

Figure 37: *confluentFunction* method

- **virtual string className()**: Returns the name of the atomic class.

The following methods can invoke certain predefined primitives to allow interaction with the abstract simulator:

- **holdIn(state, time)**: indicates to the simulator that the model should stay in the same state during a time, and after that it will generate an internal transition.
- **passivate()**: indicates to the simulator that the model enters in passive mode and that it will only be reactivated when an external event arrives.
- **sendOutput(time, port, value)**: sends an output message through the port.
- **nextChange()**: this method allows to obtain the remaining time for its next state change

(sigma).

- **lastChange()**: this method allows to obtain the time in that the last state change took place.
- **state()**: this method obtain the actual phase of the model.
- **getParameter(modelName, parameterName)**: this method allows access to the parameters that configure the class. In figure 3.1, shows the model name between the clauses and the different parameter names that range from “value 1” to “valuen”.
- **virtual Port &addInputPort(const string &)**
virtual Port &addOutputPort(const string &): These methods add the input and output port of the model respectively.
- **ModelState* getCurrentState()**: This method get the current state of the model.
- **virtual int valueSize() const**: Returns the size of the class. It should be set to:
`return sizeof(className);`
- **virtual string asString()**: Returns a string that is used in the log file to log the value sent or received.
- **virtual BasicMsgValue * clone()**: Returns a pointer to a new copy of the message value. The function that receives the pointer will own it and afterwards delete it.
- **BasicMsgValue(const BasicMsgValue&)**: Copy constructor.

The state of a model is made of all those variables that can change during a simulation cycle. The basic state variables required by an atomic model are defined in the *AtomicState* class. A user can create a new class to define the state variables required by his model.

The *AtomicState* class declaration is shown below in Figure 38.

```
class AtomicState : public ModelState {
public:

    enum State
    {
        active,
        passive
    } ;

    State st;

    AtomicState(){};
    virtual ~AtomicState(){};

    AtomicState& operator=(AtomicState& thisState); //Assignment
    void copyState(BasicState *);
    int getSize() const;

};
```

Figure 38: The *AtomicState* class.

To access the current state the function `ModelState* getCurrentState()` should be used. The pointer that

is returned can be casted to the proper type.

An assignment operator and a copy constructor need to be provided for Warped to work properly. In addition, the method getSize should be overridden to return the size of the class. The set of services provided by the class atomic as well as the methods required to be redefined can be seen in section 3.1.1.

The user can define a new class for the output values. To define a new structure for output values, a new class that derives from BasicMsgValue has to be created. A class for sending and receiving real values is already provided. See Figure 39 for both the base class (BasicMsgValue) and the derived class (RealMsgValue) from the base class.

Warning: There is a restriction that applies. No pointers can be defined as part of the class. This is because message values are sent across a network when parallel simulation is used and pointers will be just copied as pointers. The data they are pointing to will not be copied.

```
class BasicMsgValue {
public:
    BasicMsgValue();
    virtual ~BasicMsgValue();
    virtual int valueSize() const;
    virtual string asString() const;
    virtual BasicMsgValue* clone() const;

    BasicMsgValue(const BasicMsgValue& );
};

class RealMsgValue : public BasicMsgValue {
public:
    RealMsgValue();
    RealMsgValue( const Value& val);

    Value v;
    int valueSize() const;
    string asString() const ;
    BasicMsgValue* clone() const;
    RealMsgValue(const RealMsgValue& );
};
```

Figure 39: The BasicMsgValue and RealMsgValue classes

The user needs to define the following functions:

- **virtual int valueSize() const;**
Returns the size of the class. It should be set to:
`return sizeof(className);`
- **virtual string asString();**
Returns a string that is used in the log file to log the value sent or received.
- **virtual BasicMsgValue * clone();**
Returns a pointer to a new copy of the message value. The function that receives the pointer will own it and afterwards delete it.
- **BasicMsgValue(const BasicMsgValue&)**
A copy constructor is required.

Once the state class has been defined, we are ready to implement the model itself. The Queue class declaration is shown in Figure 41.

```

class QueueState : public AtomicState {

public:

    typedef list<BasicMsgValue *> ElementList ;
    ElementList elements ;
    bool enabled;

    QueueState(){};
    virtual ~QueueState(){};

    QueueState& operator=(QueueState& thisState){
        (AtomicState &)*this = (AtomicState &) thisState;

        ElementList::const_iterator cursor;

        for(cursor = thisState.elements.begin();
            cursor != thisState.elements.end(); cursor++)

            elements.push_back( cursor->clone() );

        return *this;
    }

    void copyState(QueueState *){ *this = *((QueueState *) rhs);}

    int getSize() const { return sizeof(QueueState);}
};

```

Figure 40: QueueState class

The Queue model overloads the initialization methods, internal function, external transition and output function. In addition, its shortcut functions to access the elements of the current state.

```

class Queue : public Atomic{
public:
    Queue( const string &name = "Queue" );
    virtual string className() const { return "Queue" ;}
protected:
    Model &initFunction();
    Model &externalFunction( const MsgBag & );
    Model &internalFunction( const InternalMessage & );
    Model &outputFunction( const CollectMessage & );

    ModelState* allocateState()
    {   return new QueueState; }

private:
    Port &in, &done, &out;

    VTime preparationTime;

    QueueState::ElementList& elements(){ return
((QueueState*)getCurrentState())->elements; }

    bool enabled() const{   return ((QueueState*)getCurrentState())->enabled;
}

    void enabled (bool val){   ((QueueState*)getCurrentState())->enabled =
val; }

```

```
}; // class Queue
```

Figure 41: The Queue class declaration

The initFunction has to set the initial state for the queue, as shown in Figure 42. The elements of the list will be erased and the enabled will be set to true.

```
Model &Queue::initFunction()
{
    enabled( true );
    return *this;
}
```

Figure 42: initFunction for the Queue model

The externalFunction will be activated every time one or more events are received. For the queue model, this function will have to insert into the queue all values received through port *In*, schedule an output if a value is received through the port *next* and enable or disable the queue if an event is received through port *start-stop*, as detailed in Figure 43. It is important to notice that it is the modeler's responsibility to set which message will have the highest priority when more than one is received. For our queue model, it can be seen from Figure 43 that the *start-stop* messages will have higher precedence than the *done* and *in* messages.

```
Model &Queue::externalFunction( const MsgBag & bag )
{
    if ( portHasMsgs( "start-stop" ) )
    {
        enabled ( !enabled() );
        if ( !enabled() )
            passivate();
    }

    if ( enabled() && portHasMsgs( "done" ) )
    {
        elements().pop_front();
        holdIn( AtomicState::active, preparationTime );
    }

    if ( portHasMsgs( "in" ) )
    {
        MessageList::const_iterator cursor;
        cursor = bag.msgOnPort( "in" ).begin();

        for ( ; cursor != bag.msgsOnPort( "in" ).end() ; cursor++)
            elements().push_back( cursor.value() );

        //If the queue was empty, schedule the next transition
        if ( enabled() && elements.size()==msgsOnPort("in").size() )
            holdIn( AtomicState::active, preparationTime );
    }
}
```

Figure 43: External transition function for the queue model

The output function is called before an internal transition (Figure 44). In our queue model, the output function should send the first value in the list through the output port. The internal transition function

will cause the model to go into a passive state the model will wait for an external event to take place.

```
Model &Queue::outputFunction( const CollectMessage &msg )
{
    sendOutput( msg.time(), out, elements.front() );
    return *this;
}

Model &Queue::internalFunction( const InternalMessage & )
{
    passivate();
    return *this;
}
```

Figure 44: Methods for the Output Function and the Internal Transition of the Queue

The sendOutput function will delete the pointer it receives, so all memory previously allocated to store the queue values will be reclaimed. If we wanted to use the queue for a network model, the queue would store IP packets. In this case, an IP packet class derived from BasicMsgValue should be defined.

Coupled models

After defining the atomic models for a given application, they can be combined into a multicomponent model. Coupled models are defined using a specification language specially defined with this purpose. The language was built following the formal definitions for DEVS coupled models. Therefore, each of the components defined formally for DEVS coupled models can be included. Optionally, configuration values for the atomic models can be included.

The **[top]** model always defines the coupled model at the top level. As showed in the formal specifications presented earlier, four properties must be configured: components, output ports, input ports and links between models. The following syntax is used:

- **Components:**

```
components: model_name1[@atomicclass1] model_name [@atomicclass2] ...
Lists the component models that make the coupled model. If this clause is not specified, an error will occur. A coupled model might have atomic models or other coupled model as components. For atomic components, an instance name and a class name must be specified. This allows a coupled model to use more than one instance of an atomic class. For coupled models, only the model name must be given. This model name must be defined as another group in the same file.
```

- **Out:**

```
out : portname1 portname2 ...
Enumerates the model's output ports. This clause is optional because a model may not have output ports.
```

- **In:**

```
in : portname1 portname2 ...
Enumerates the input ports. This clause is also optional because a coupled model is not required to have input ports.
```

- **Link :**

```
link : source_port[@model1] destination_port[@model1]
Defines the links between the components and between the components and the coupled model itself. If name of the model is omitted it is assumed that the port belongs to the coupled model being defined.
```

A model definition is shown below.

```
[top]
components : transducer@Transducer generator@Generator Consumer
Out : out
Link : out@generator arrived@transducer
Link : out@generator in@Consumer
Link : out@Consumer solved@transducer
Link : out@transducer out

[Consumer]
components : queue@Queue processor@Processor
in : in
out : out
Link : in in@queue
Link : out@queue in@processor
Link : out@processor done@queue
Link : out@processor out
```

Figure 45: Example for the definition of a DEVS coupled model

CD++Builder allows the user to easily import and export maml files. .maml files are a representation of coupling files in XML format. Exporting a “.ma” file into maml format is done automatically every time you save your “.ma” file. The saving feature writes a maml file with the same name as the ma file in maml format. This is automatically added to your project.

Importing maml files can be simplistic as well. When you have a maml file in your project, right clicking it and selecting convert to ma, will create a coupling file implementation of the maml file. This coupling file is written and created in your project and should appear in the projects' contents

As it was mentioned above, atomic models must be coded. In addition, an atomic model might have user-defined parameters that must be specified within the .ma file. If this is the case, the parameters are specified in a group with the model's name (the model's name as defined in the components clause, not the atomic class name).

```
[model_name]
var_name1 : value1
.
.
.
var_namen : valuen
```

Figure 46. User defined values for atomic models

The parameter names are defined by the model's author and must be documented. Each instance of an atomic model can be configured independently of other instances of the same kind.

The next example shows two instances of the atomic class *Processor* with different values for the user-defined parameters.

```
[top]
components : Queue@queue Processor1@processor Processor2@processor
.

.

.

[processor]
distribution : exponential
mean : 10

[processor2]
distribution : poisson
mean : 50
```

```
[queue]
preparation : 0:0:0:0
```

Figure 47: Example of setting parameters to DEVS atomic models

Cell-DEVS models

CD++ includes a specification language allowing the description of Cell-DEVS models. These definitions are based on the formal specifications defined earlier, and can be completed by considering a few parameters: size, influences, neighborhood and borders. These are used to generate the complete cell space. The behavior of the local computing function is defined using a set of rules with the form: VALUE DELAY {CONDITION}. These indicate that when the *CONDITION* is satisfied, the state of the cell changes to the designated *VALUE*, and it is *DELAYed* for the specified time. If the condition is *false*, the next rule in the list is evaluated until a rule is satisfied or there are no more rules. In the latter case, an error is raised, indicating that the model specification is incomplete. The existence of two or more rules with same condition but with different state value or delay is also detected, avoiding the creation of ambiguous models. In these situations, the simulation is aborted.

In CD++, Cell DEVS models are a special case of coupled models. Then, when defining a cellular model, all the coupled model parameters are available. In addition there exist some parameters that are of cellular models. These parameters define the dimensions of the cell space, the type delay, the default initial values and the local transition rules.

Note: The blank spaces before and after the ":" are required.

These parameters are:

- **type** : [CELL | FLAT]
Defines the abstract simulator to be used. If **cell** is specified, there will be one DEVS processor for each cell. Instead, if **flat** is specified, one flat coordinator will be used. CD++ currently supports the **cell** option only.
- **width** : integer
Defines the width of the cellular space. As it is the case with height, the **width** parameter is provided for backward compatibility and implies that a 2-dimensional cellular space will be used. For an n-dimensional cell space the **dim** parameter should be used. **width** and **height** can not be used together with **dim**. If such a situation exists, an error will be reported.
- **height** : integer
Defines the height of the cellular space model. The same restrictions that were given for **width** apply. For 1 dimension models, **height** should be set to 1.
- **dim** : (x_0, x_1, \dots, x_n)
Defines the dimensions of the cellular space. All the x_i values must be integers.
Dim can not be used together with any of the **width** and **height** parameters.
The vector that defines the dimension of the cellular model must have two or more elements. For a one-dimensional cellular model, the following form should be used; $(x_0, 1)$.
When referencing a cell, all references must satisfy:

$$(y_0, y_1, \dots, y_n) \quad 0 \leq y_i < x_i \quad i = 0, \dots, n$$
with y_i an integer value
- **In** : Defines the input ports for a cellular model.
- **Out** : Defines the output ports the cellular model.

- **Link** : Defines the components coupling. For a coupled cell model, the components are cells. To define the couplings, cell references must be used for the model name. A cell reference is of the form:

CoupleCellName(x_1, x_2, \dots, x_n)

Valid link definitions are of the form:

Link : outputPort inputPort@cellName (x_1, x_2, \dots, x_n)
 Link : outputPort@cellName (x_1, x_2, \dots, x_n) inputPort
 Link : outputPort@cellName (x_1, x_2, \dots, x_n)inputPort@cellName(x_1, x_2, \dots, x_n)

- **Border** : [WRAPPED | NONWRAPPED]
 Defines the type of border for the cellular space. By default, NOWRAPPED is used. If a nonwrapped border is used, a reference to a cell outside the cellular space will return the undefined value (?).
- **Delay** : [TRANSPORT | INERTIAL]
 Specifies the delay type used for all cells of the model. By default the value TRANSPORT is assumed.
- **DefaultDelayTime** : integer
 Defines the default delay (in milliseconds) for inputs received from external DEVS models. If a portInTransition is specified, then this parameter will be ignored for that cell.
- **Neighbors** : cellName ($x_{1,1}, x_{2,1}, \dots, x_{n,1}$)... cellName ($x_{1,m}, x_{2,m}, \dots, x_{n,m}$)
 Defines the neighborhood for all the cells of the model. Each cell ($x_{1,i}, x_{2,i}, \dots, x_{n,i}$) represents a displacement from the centre cell (0,0,..., 0)
 A neighborhood can be defined with any valid list of cells and is not restricted to adjacent cells.
 It is possible to use more than one **neighbors** sentence to define the neighborhood.
- **Initialvalue** : [Real | ?]
 Defines the default initial value for each cell. The symbol ? represents the undefined value. There are several ways of defining the initial values for each cell. The parameter **initialvalue** has the least precedence. If another parameter defines a new value for the cell, then that value will be used.
- **InitialRowValue** : row_i value₁...value_{width}
 Defines the initial value for all the cells in row i.
 Precondition:
 0 ≤ row_i < Height (where Height is the second element of the dimension defined with **Dim**, or the value defined with **Height**).
 Can only be used for bidimensional models. For n-dimensional models the **initialCellsValue** or **initialMapValue** parameters are preferred.
 This clause is used for backward compatibility. All values are single digit values in the set {?, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The first digit will define the value for the first cell in the row, the second for second cell and so on. No spaces are allowed between digits.
- **InitialRow** : row_i value₁... value_{width}

Same as **initialrowvalue**, but values can now be any member of the set $\square \cup \{?\}$.
 Each value in the list must be separated by a blank space from the next one.

- **InitialCellsValue** : *fileName*
 Defines the filename for the file that contains a list of initial value for cells in the model. Section 6.1 defines the format for these files. **initialcellsvalue** can be used with any size of cellular models and will have more precedence than **initialrow** and **initialrowvalue**.
- **InitialMapView** : *fileName*
 Defines the filename for the file that contains a map of values that will be used as the initial state for a cellular model. Section 6.2 defines the format for these files.
- **LocalTransition** : *transitionFunctionName*
 Defines the name of the group that contains the rules for the default local computing function.
- **PortInTransition** : *portName@ cellName (x₁, x₂,...,x_n) TransitionFunctionName*
 It allows to define an alternative local transition for external events. By default, if this parameter is not used, when an external event is received by a cell its value will be the future value of the cell with a delay as set by the **defaultDelayTime** clause.

Section 12.3 illustrates the use of the **portInTransition** clause.

- **Zone** : *transitionFunctionName { range₁[..range_n] }*
 A zone defines a region of the cellular space that will use a different local computing function. A zone is defined giving as a set of single cells or cell ranges. A single cell is defined as (x₁,x₂,...,x_n), and a range as (x₁,x₂,...,x_n)..(y₁,y₂,y_n). All cells and cell ranges must be separated by a blank space. As an example,
 $\text{zone} : \text{pothole} \{ (10,10)..(13, 13) (1,3) \}$
 tells CD++ that the local transition rule pothole will be used for the cells in the range (10,10)..(13,13) and the single cell (1,3). The zone clause will override the transition defined by the **localtransition** clause.

The following Figure 48 illustrated the .ma file of the life game(previously described in section 1.1):

```
[top]
components : life

[life]
type : cell
width : 20
height : 20
delay : transport
defauultDelayTime : 100
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1) life(0,0) life(0,1)
neighbors : life(1,-1) life(1,0) life(1,1)
initialvalue : 0
initialvalue : 5 00000001110000000000
initialvalue : 7 00000100100100000000
initialvalue : 8 00000101110100000000
initialvalue : 9 00000100100100000000
initialvalue : 11 00000001110000000000
```

```

localtransition : life-rule

[life-rule]
rule : 1 1 100 { (0,0) = 1 and trueCount = 5 }
rule : 1 1 100 { (0,0) = 0 and trueCount = 3 }
rule : 0 1 100 { t }

```

Figure 48: Example for the definition of a Cell-DEVS life model

Supporting files

Defining initial cell values using a .val file

Within the definition of a cellular model, the *InitialCellValue* parameter defines a file name with the initial values for the cells. This is a plain text file. Each line of the file defines a value for a different cell. The format of this file is shown in Figure 49.

```

(x0,x1,...,xn) = value_1
...
...
(y0,y1,...,yn) = value_m

```

Figure 49: Format of the file used to define the initial values of a cellular model

The extension **.VAL** is normally used for this kind of files. The file is processed in sequential order, so if there are two values defined for the same cell, the latest one will be used.

The dimension of the **tuple** should match the dimensions of the cellular space.

For the definition of the initial values of a cellular model, a single file should be used, which cannot contain initial values for other cellular models.

It is not necessary to define an initial value for each cell. If no value is defined in this file, then the value defined by the parameter *InitialValue* in the *.ma* file will be used. Figure 50 shows a short fragment of a .val file for a cellular space of 4 dimensions.

```

(0,0,0,0) = ?
(1,0,0,0) = 25
(0,0,1,0) = -21
(0,1,2,2) = 28
(1, 4, 1,2) = 17
(1, 3, 2,1) = 15.44
(0,2,1,1) = -11.5
(1,1,1,1) = 12.33
(1,4,1,0) = 33
(1,4,0,1) = 0.14

```

Figure 50: Example of a file for the definition of the initial values for a Cellular Model

Defining initial cell values using a .map file

If the *InitialMapValue* parameter is used, then the initial values for a cellular model are specified in a .map file. This file contains a map of cell values, as shown in Figure 51:

```
value_1  
...  
value_m
```

Figure 51: .map file format

Each value of the .map file will be assigned to a cell starting with the origin cell (0,0,...0). For a three-dimensional cellular model of size (2, 3, 2), the values will be assigned in the following order:

(0,0,0) (0,0,1) (0,1,0), (0,1,1) (0,2,0) (0,2,1) ... (1,2,0) (1,2,1)

If there are not enough values in the file for all the cells in the model, the simulation will be aborted. If instead there are more values than cells, the remaining values will be ignored.

The *toMap* tool creates a .map file from a .val file.

External events file

External events are defined in a plain text file with one event per line. Each line will be of the format:

HH:MM:SS:MS PORT VALUE

Where:

- | | |
|--------------------|----------------------------------------------------------------------------------------|
| HH:MM:SS:MS | is the time when the event will occur. |
| Port | is the name of the port from which the event will arrive. |
| Value | is the numerical value for the event. Can be a real number or the undefined value (?). |

Example:

```
00:00:10:00 in 1  
00:00:15:00 done 1.5  
00:00:30:00 in .271  
00:00:31:00 in -4.5  
00:00:33:10 inPort ?
```

Figure 52: File with external events

Partition file

A partition file is required for parallel simulation. For each atomic model, the partition file defines the machine that will host its associated simulator. For coupled models, CD++ will decide where the coordinators will be running.

A partition file, usually referred to as a .par file, has lines with the following format:

MachineNumber : modelName1 modelName2 cell(x,y) cell(x,y)...(x2, y2)

A line starts with a machine number (machine numbers start at 0) followed by a colon and a

list of names separated by spaces. Different lines may start with the same machine number. The list of names following a machine number is the list of atomic instances that will be hosted by that machine. For cellular models, a single cell may be specified or a range of cells may be given. A cell range is described with name of the coupled cell model followed by the first cell in the range, two dots, and the last cell in the range.

As an example, consider the following partial definition of a model:

```
[top]
components : superficie generadorCalor@Generator generadorFrio@Generator
link : out@generadorCalor inputCalor@superficie
link : out@generadorFrio inputFrio@superficie

[superficie]
type : cell
width : 100
height : 100
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : superficie(-1,-1) superficie(-1,0) superficie(-1,1)
neighbors : superficie(0,-1) superficie(0,0) superficie(0,1)
neighbors : superficie(1,-1) superficie(1,0) superficie(1,1)
initialvalue : 24
in : inputCalor inputFrio
```

Figure 53: Partial definition of the heat diffusion model

If we wanted to run this model in a cluster of nine machines, then the following is a valid partition:

```
0 : generadorCalor generadorFrio
0 : superficie(0,0)..(32,32)
1 : superficie(0,33)..(32,65)
2 : superficie(0,66)..(32,99)
3 : superficie(33,0)..(65,32)
4 : superficie(33,33)..(65,65)
5 : superficie(33,66)..(65,99)
6 : superficie(66,0)..(99,32)
7 : superficie(66,33)..(99,65)
8 : superficie(66,66)..(99,99)
```

Figure 54: Valid partition for the heat diffusion model over 9 machines

A valid partition must specify one and only one location for each atomic and each cell. If more than one machine or no machine is specified for a model, then an error will be raised and the simulation will be aborted.

Output Files

Output events

If the command line option **-o** is given, all the output events generated by the simulator are written to the specified file. There will be one event per line, and lines will have the following format:

HH:MM:SS:MS PORT VALUE

Following is a small example of an output file.

```

00:00:01:00 out 0.000
00:00:02:00 out 1.000
00:00:03:50 outPort ?
00:00:07:31 outPort 5.143

```

Figure 55: Example of an Output file

Format of the Log File

A log file keeps a record of all the messages sent between DEVS processors. A log is created when the `-l` command line argument is used. If no log modifiers are specified, all received messages are logged. Otherwise, only those messages set by the log modifiers will be logged.

When a filename for the log is given, there will be one file per DEVS processor and one file with the list of all the names of the files that have been created. This latter file will be named with the name given after the `-l` parameter. All other files will be named with the name after the `-l` parameter followed by the DEVS processor id.

Each line of the file shows the number of the LP that received the message, the message type, the time of the event, the sender and the receiver. In addition, messages of type **X** or **Y** will include the port through which the message was received and the value received. For messages of type **D**, the remaining type for the next transition will be shown. A '...' for this field will indicate infinity.

The numbers between brackets show the ID of the DEVS processor and are provided for debugging purposes only.

As an example, the log files for the following model will be shown.

```

[top]
components : superficie generadorCalor@Generator generadorFrio@Generator
link : out@generadorCalor inputCalor@superficie
link : out@generadorFrio inputFrio@superficie

[superficie]
type : cell
width : 5
height : 5
...

```

Figure 56: Partial definition of the heat diffusion model

When running this model with the `-lcalor.log` parameter, the following are the contents of `calor.log`.

```

[logfiles]
ParallelRoot : calor.log00
top : calor.log29
superficie : calor.log01
superficie(0,0) : calor.log02
superficie(0,1) : calor.log03
superficie(0,2) : calor.log04
superficie(0,3) : calor.log05
superficie(0,4) : calor.log06
superficie(1,0) : calor.log07
superficie(1,1) : calor.log08
superficie(1,2) : calor.log09
superficie(1,3) : calor.log10

```

```

superficie(1,4) : calor.log11
superficie(2,0) : calor.log12
superficie(2,1) : calor.log13
superficie(2,2) : calor.log14
superficie(2,3) : calor.log15
superficie(2,4) : calor.log16
superficie(3,0) : calor.log17
superficie(3,1) : calor.log18
superficie(3,2) : calor.log19
superficie(3,3) : calor.log20
superficie(3,4) : calor.log21
superficie(4,0) : calor.log22
superficie(4,1) : calor.log23
superficie(4,2) : calor.log24
superficie(4,3) : calor.log25
superficie(4,4) : calor.log26
generadorcalor : calor.log27
generadordfrio : calor.log28

```

Figure 57: Calor.log

This is a list of the models and their corresponding files. If more than one file is created (as is the case of coupled models with more than one coordinator), all of them are listed. The log messages received by the coordinator superficie will be logged into the file calor.log01, which is shown next.

```

0 I / 00:00:00:000 / top(29) para superficie(01)
0 D / 00:00:00:000 / superficie(0,0)(02) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,1)(03) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,2)(04) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,3)(05) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,4)(06) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(1,0)(07) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(1,1)(08) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(1,2)(09) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(1,3)(10) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(1,4)(11) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(2,0)(12) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(2,1)(13) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(2,2)(14) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(2,3)(15) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(2,4)(16) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(3,0)(17) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(3,1)(18) / 00:00:00:000 para
superficie(01)

```

```

0 D / 00:00:00:000 / superficie(3,2)(19) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(3,3)(20) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(3,4)(21) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(4,0)(22) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(4,1)(23) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(4,2)(24) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(4,3)(25) / 00:00:00:000 para
superficie(01)
0 D / 00:00:00:000 / superficie(4,4)(26) / 00:00:00:000 para
superficie(01)
0 @ / 00:00:00:000 / top(29) para superficie(01)
0 Y / 00:00:00:000 / superficie(0,0)(02) / out / 24.00 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,0)(02) / 00:00:00:000 para
superficie(01)
0 Y / 00:00:00:000 / superficie(0,1)(03) / out / 24.00 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,1)(03) / 00:00:00:000 para
superficie(01)
0 Y / 00:00:00:000 / superficie(0,2)(04) / out / 24.00 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,2)(04) / 00:00:00:000 para
superficie(01)
0 Y / 00:00:00:000 / superficie(0,3)(05) / out / 24.00 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,3)(05) / 00:00:00:000 para
superficie(01)
0 Y / 00:00:00:000 / superficie(0,4)(06) / out / 24.00 para
superficie(01)
0 D / 00:00:00:000 / superficie(0,4)(06) / 00:00:00:000 para
superficie(01)
...
...
0 X / 00:00:00:000 / top(29) / inputcalor / 1.00 para superficie(01)
0 X / 00:00:00:000 / top(29) / inputfrio / 1.00 para superficie(01)
0 * / 00:00:00:000 / top(29) para superficie(01)

```

Figure 58. Fragment of calor.log01

Partition Debug Info

The partition debug info file lists all the DEVS processors that are taking part in the simulation, their IDs and the machine they are running on. This file is useful to know where the coordinators for coupled models are placed. One partition debug info file is created by each LP. The files will be named with the text after the command line **-D** argument followed by the LP number.

Figure 59 shows a fragment of a partition debug file generated when running the model described in Figure 53 with the partition shown next.

```

0 : generadorCalor generadorFrio
0 : superficie(0,0)..(2,4)
1 : superficie(3,0)..(4,4)

```

Figure 59: Partition for the heat diffusion model of Figure 53

```
Model: ParallelRoot
Machines:
    Machine: 0  ProcId: 0 < master >

Model: top
Machines:
    Machine: 0  ProcId: 30 < master >

Model: superficie
Machines:
    Machine: 0  ProcId: 1 < master >
    Machine: 1  ProcId: 2 < local >

Model: superficie(0,0)
Machines:
    Machine: 0  ProcId: 3 < master >

...
Model: superficie(3,0)
Machines:
    Machine: 1  ProcId: 18 < local > < master >

Model: superficie(3,1)
Machines:
    Machine: 1  ProcId: 19 < local > < master >

Model: superficie(3,2)
Machines:
    Machine: 1  ProcId: 20 < local > < master >

Setting up the logical process
Total objects: 31
Local objects: 11
Total machines: 2

About to create the LP
LP has been created. Now registering processors.
Registering processor superficie(2)
Registering processor superficie(3,0)(18)
Registering processor superficie(3,1)(19)
Registering processor superficie(3,2)(20)
Registering processor superficie(3,3)(21)
Registering processor superficie(3,4)(22)
Registering processor superficie(4,0)(23)
Registering processor superficie(4,1)(24)
Registering processor superficie(4,2)(25)
Registering processor superficie(4,3)(26)
Registering processor superficie(4,4)(27)

Current processors:
Processor Id: 2      Description: superficie
Model Id: 2 superficie(02)
Parent Id: 30

...
Processor Id: 27      Description: superficie(4,4)
```

```

Model Id: 27 superficie(4,4)(27)
Parent Id: 2
All objects have been registered!
Initializing Object superficie(2): OK
Initializing Object superficie(3,0)(18): OK
Initializing Object superficie(3,1)(19): OK
Initializing Object superficie(3,2)(20): OK
Initializing Object superficie(3,3)(21): OK
Initializing Object superficie(3,4)(22): OK
Initializing Object superficie(4,0)(23): OK
Initializing Object superficie(4,1)(24): OK
Initializing Object superficie(4,2)(25): OK
Initializing Object superficie(4,3)(26): OK
Initializing Object superficie(4,4)(27): OK
After Initialize....OK

```

Figure 60: Partition debug information file calor.pardeb01 (LP 1)

Output generated by the Parser Debug Mode

When the simulator is invoked with the option **-p**, the debug mode for the parser is activated. In debug mode, the parser will write the parse tree as it reads the rules. All tokens that are successfully processed are shown and if there is a syntax error, the place were the error was detected is specified.

Figure 61 shows the output generated for the *Game Life* model as implemented in section 12.1.

```

***** BUFFER *****
1 100 { (0,0) = 1 and (truecount = 3 or truecount = 4) } 1 100 { (0,0) =
0 and truecount = 3 } 0 100 { t } 0 100 { t }
Number 1 analyzed
Number 100 analyzed
Number 0 analyzed
Number 0 analyzed
OP_REL parsed (=)
Number 1 analyzed
AND parsed
COUNT parsed (truecount)
OP_REL parsed (=)
Number 3 analyzed
OR parsed
COUNT parsed (truecount)
OP_REL parsed (=)
Number 4 analyzed
Number 1 analyzed
Number 100 analyzed
Number 0 analyzed
Number 0 analyzed
OP_REL parsed (=)
Number 0 analyzed
AND parsed
COUNT parsed (truecount)
OP_REL parsed (=)
Number 3 analyzed
Number 0 analyzed
Number 100 analyzed
BOOL parsed (t)

```

```

Number 0 analyzed
Number 100 analyzed
BOOL parsed (t)

```

Figure 61: Output generated in the Parser Debug Mode for the Game of Life

Rule evaluation debugging

Using the `-v` command line argument, a debug mode for cell rules evaluation is enabled. This will cause the simulator to log all intermediate values for each rule as it is evaluated.

Figure 62 shows a fragment of the output generated for the Game of Life model of section 12.1. Line numbers have been added to make the following explanations clear.

The first two lines indicate the beginning of a new evaluation. Line 2 begins the evaluation of the first rule for the first cell. Each evaluated argument is listed with the partial result for the expression. Line 2 shows the evaluation of the cell reference $(0,0)$, which turned out to be 0. In line 3, the integer constant 1 is evaluated, which is later compared to 0, evaluating to 0 (false). *BinaryOp* indicates that a binary operation is being performed. The operator name will be included between brackets, as well as the value of each of the operands. Line 13 shows the final result for the condition of the rule, which was false in this case.

```

00
+-----+
01 New Evaluation:
02 Evaluate: Cell Reference(0,0) = 0
03 Evaluate: Constant = 1
04 Evaluate: BinaryOp(0, 1) = (=) 0
05 Evaluate: CountNode(1) = 1
06 Evaluate: Constant = 3
07 Evaluate: BinaryOp(1, 3) = (=) 0
08 Evaluate: CountNode(1) = 1
09 Evaluate: Constant = 4
10 Evaluate: BinaryOp(1, 4) = (=) 0
11 Evaluate: BinaryOp(0, 0) = (or) 0
12 Evaluate: BinaryOp(0, 0) = (and) 0
13 Evaluate: Rule = False
14
15 Evaluate: Cell Reference(0,0) = 0
16 Evaluate: Constant = 0
17 Evaluate: BinaryOp(0, 0) = (=) 1
18 Evaluate: CountNode(1) = 1
19 Evaluate: Constant = 3
20 Evaluate: BinaryOp(1, 3) = (=) 0
21 Evaluate: BinaryOp(1, 0) = (and) 0
22 Evaluate: Rule = False
23
24 Evaluate: Constant = 1
25 Evaluate: Rule = True
26
27 Evaluate: Constant = 100
28 Evaluate: Constant = 0
29
+-----+
30 ...
31 ...

```

```

32 ...
33 ...
34
+-----+
35 New Evaluation:
36 Evaluate: Cell Reference(0,0) = 1
37 Evaluate: Constant = 1
38 Evaluate: BinaryOp(1, 1) = (=) 1
39 Evaluate: CountNode(1) = 4
40 Evaluate: Constant = 3
41 Evaluate: BinaryOp(4, 3) = (=) 0
42 Evaluate: CountNode(1) = 4
43 Evaluate: Constant = 4
44 Evaluate: BinaryOp(4, 4) = (=) 1
45 Evaluate: BinaryOp(0, 1) = (or) 1
46 Evaluate: BinaryOp(1, 1) = (and) 1
47 Evaluate: Rule = True
48
49 Evaluate: Constant = 100
50 Evaluate: Constant = 1
51
+-----+
52 ...
53 ...
54 ...

```

Figure 62: Fragment of the output generated by the debug mode for the Evaluation or Rules

6.6 Model simulation through the console

In this section you will be guided on running a simulation using the command line. Once again the existing model, called life will be used. The life example can be downloaded at <http://www.sce.carleton.ca/faculty/wainer/wbgraf> as mentioned in section 1. To start the simulation, unzip the life example to a folder. Copy the simulator file, simu.exe, into the folder, where the life example was unzipped. The simu.exe file is created every time a new atomic model is created, compiled and registered. (CD++ is provided with an original simu.exe, which can simulate CellDEVS models.) Run the batch file (demo.bat) to simulate the model. This batch file will execute simu.exe with the defined parameters.

To simulate the life model manually (without batch file) type the following command in the command line:

```
simu -mlife.ma -t00:01:00:000 -llife.log
```

Once the simulation is finished, a log file is created and can be viewed. The file can also be simulated using the CD++Builder toolkit on Eclipse. Further information about simulating models using CD++Builder is given in section 7.1.

Note: The example simulation conducted in this section (simulating the model life2d) is an example of how to simulate Cell-DEVS models. To simulate a DEVS model, the model must be complied. Section 7.1.1 provides details on how to compile DEVS models.

To configure the execution of the simulator, the following parameters are valid:

-h: shows this help:

```
simu [-ehlmotdpvbfqrsw]
e: events file (default: none)
h: show this help
```

I: message log file (default: /dev/null)
m: model file (default : model.ma)
o: output (default: /dev/null)
t: stop time (default: Infinity)
d: set tolerance used to compare real numbers
p: print extra info when the parsing occurs (only for cells models)
v: evaluate debug mode (only for cells models)
b: bypass the preprocessor (macros are ignored)
f: flat debug mode (only for flat cells models)
r: debug cell rules mode (only for cells models)
s: show the virtual time when the simulation ends (on stderr)
q: use quantum to calculate cells values
w: sets the width and precision (with form xx-yy) to show numbers

-e: External events filename. If this parameter is omitted, the simulator will not use external events. The format used to describe the external events is showed in the section 6.3.

-I: Log filename. This file is used to store the messages received and emitted by each model within the simulation. If this parameter is omitted, the simulator will not generate activity log. If you wish to get the log on standard output, you should write **-I).** The format used by the log is described in the section 7.2.

-m: Model description filename. This parameter indicates the name of the file that contains the description of all models to simulate. If this parameter is omitted, the simulator will try to load the models from the *model.ma* file.

-o: output filename. This parameter indicates the name of the file that will be used to store the output generated by the simulator. If this parameter is omitted, the simulator will not generate any output. If you wish to get the results on standard output, simply write **-o**. The format of this output is showed in the section 7.1.

-t: Sets the maximum time to simulate. If this parameter is omitted, the simulator will stop only when it will not have more events (internal or external). The format used to set the time is HH:MM:SS:MS, where:

HH: hours
MM: minutes (0 to 59)
SS: seconds (0 to 59)
MS: thousandth of second (0 to 999)

-d: Defines the tolerance used to compare real numbers. The value passed with the **-d** parameter will be used as the new tolerance value. By default, the value used is 10^{-8} .

-p: Shows additional information on parsing the cell model's rules. The parameter must be accompanied with the filename that will be used to store the detail. This mode is useful when a syntax error occurs on complex rules.
The format used to store the output is showed in the section 7.4.

-v: Enable the debug mode on the evaluation of all cell model's rules. For each rule to be evaluated it will be showed the results of the evaluation of each function and operator that they compose it. In addition, this mode evaluates the rules in complete form, that is,

it doesn't use the rule's optimization. The parameter must be accompanied with the filename that will be used to store the rule's evaluation. The format of the output generated when this mode is enabled is showed in the section 7.5.

- b**: Bypass the preprocessor. When this parameter is set, the macros will be ignored.
- f**: Enable the debug mode on flat cell models. This allows showing the state of a flat-coupled model on each time change. When you used flat models, the simulation process does not send messages between the atomic cells that compound it, and then, the log will not store these messages. When you run the *DrawLog*, it will be unable to show the state of the model at each time. The parameter must be accompanied with the filename that will be used to store the states. If you wish to show the results on the standard output, simply write **-f**.
- r**: Enable the debug mode that validates the rules used to define the behavior of the cells models. When this mode is enabled, the simulator checks for the existence of multiple valid rules at runtime. If this condition is true, the simulation will be aborted. This mode is available only in standalone mode. There are special cases to consider: if you are using a stochastic model (i.e. the model uses random numbers generators) must happen that multiple rules will be valid, or than none of them will be. In both cases, the simulator will notify this situation to the user, showing a warning message on standard output, but the simulation will not be aborted. For the former case, the first valid rule will be considered. For the second case, the cell will have an undefined value (?), and the delay time will be the default delay time specified for the model. If this parameter is not used when the simulator is invoked, the mode is disabled and only will be considered the first valid rule.
- s**: Show the simulation's end time on stdErr.
- q**: Allows to use a *quantum* value. This permits quantifying the value returned by the local computing function evaluated on each cell of the model. Thus, all the values will be rounded to the near maximum multiple of the quantum value minor than the original value. This mechanism decreases the number of messages transmitted in the simulation, but the results of the simulation will not be exact. For example, if the quantum value is 0.01 and the value returned by the local computing function is 0.2371, the state of the cell will be 0.23. The value used as quantum must be declared next to the parameter **-q**, for example: to set the quantum value as 0.01 the parameter must be **-q0.001**. If the *quantum* value is 0 or the parameter **-q** is not used, the use of the quantum will be disabled, and the value returned by the local computing function will be directly the value of the cell.
- w**: Allows to set the wide and precision of the real values displayed on the outputs (log file, external events file, evaluation results file, etc). By default, the wide is 12 characters and the precision is of five digits. Thus, of the 12 characters of wide, 5 will be for the precision, 1 for the decimal point, and the rest will be used for the integer part that will include a character for the sign if the value is negative. To set new values for the wide and precision, the **-w** parameter must be used, followed of the number of characters for the wide, a hyphen, and the number of characters for the decimal part. For example to use a wide of 10 characters and 3 for the decimal digits, you must write **-w10-3**. Any numerical value that must be showed by the simulator will be formatted using these values, and it will be rounded if necessary. Thus, if a cell has the value 7.0007 and the

parameter –w10–3 is declared on the invocation of the simulator, the value showed for the cell on all outputs will be 7.001, but the internal value stored will not be affected.

The drawlog command is another simulation tool provided by CD++. It is used to view the state of a cellular model after every simulation cycle as the simulation advances. Please refer to section 9 for more detailed information about the drawlog command.

Model Simulation

Simulation through Eclipse

CD++Builder plugin offers many tools to simulate/view CD++ models. This section describes how to use CD++Builder to create a model from a simulator. In order to use CD++Builder, you must use the CD++Builder perspective. This can be done by selecting “Window” on the top menu bar, selecting “Open Perspective” and then clicking on “Other”. The following perspective screen will pop up where you can select the CD++Builder Perspective



Figure 63: Perspective selection panel

The CD++ tool set provided by the CD++Builder plugin are the following.

Compiling a new model

To compile a CD++ model select any file in the project you want to compile and click on the “Build” button. The project must contain DEVS models only. This button will automatically create a makefile for you that is unique for your project and then it runs the make command on the makefile. This takes the cpp file from the project and compiles it. It then creates the simu executable that is necessary to run the simulation button. Before the compilation takes place, the build tool will ask if you want to run this tool in verbose mode.

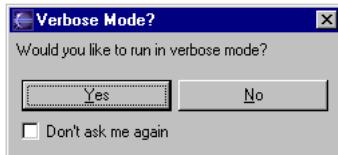


Figure 64: Verbose Mode Message

If you click yes, it will list out all the directions and messages the tool outputs when it is compiling.

Simulating a model

To simulate a project, you can click on the simu button . This will bring up a panel (shown in figure 63 where you can specify your parameters for running a simulation.

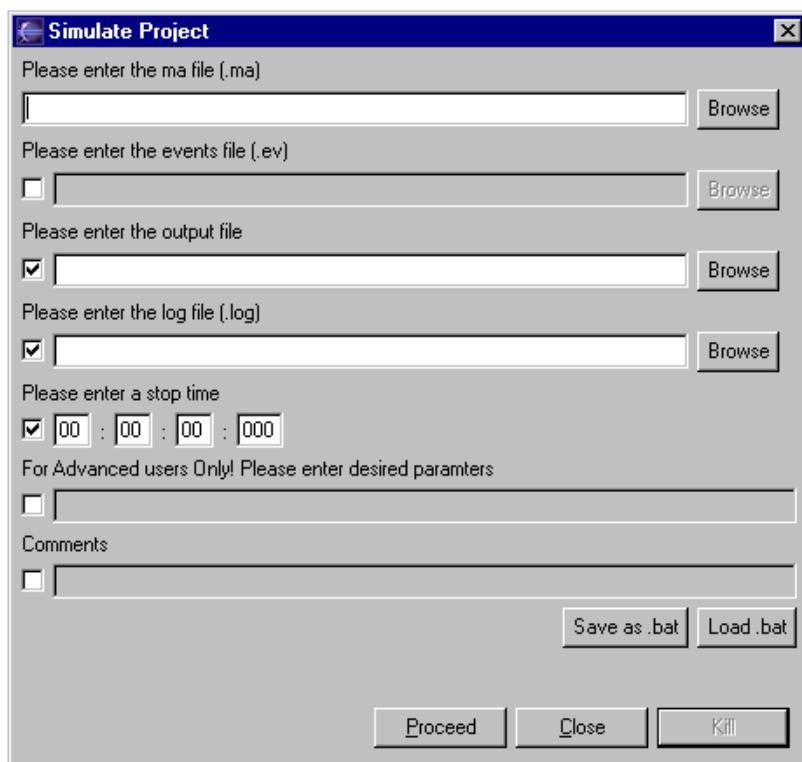


Figure 65: Simulation panel

You can enable each of the parameters by checking its respective box next to its name. Some parameters such as .ma, .ev, .out and .log, require a file input. If you know the name of your file that is in your project, you can type them in. If you would like to choose from a list of files, you can select the "browse" button that will present you with a list of all the file types.

The last three parameters are the stop time, advanced settings and comments. The stop time is separated into 4 different types - hour, minute, second and millisecond. To enter a particular type of the stop time, you can simply enter the time or you can use the up/down arrow from the keyboard to set the desired time. This time indicates when you want the simulation to stop.

The advanced parameter box is used for users who want to run less-common parameters in the simulation. Finally, the comments section is used to enter fascinating comments about the current

selection of parameters. This portion is generally used for saving your settings to a file or saving your settings to a batch file.

When you save settings to a file/batch, you will have to specify the name of the file you want to save it to. When you load settings from a file/batch, you will specify its location. By clicking proceed, the simulation will start and send all the information to the bottom component of the panel. Clicking on the output text and moving up and down with the up and down keys, will let you review the simulation and its results. If your coupling file and/or events file is non existent or corrupted, the program will send an error pop up window to tell you. (Note – you must be in the current project of the project you want to simulate.)

Creating drawlog file for models

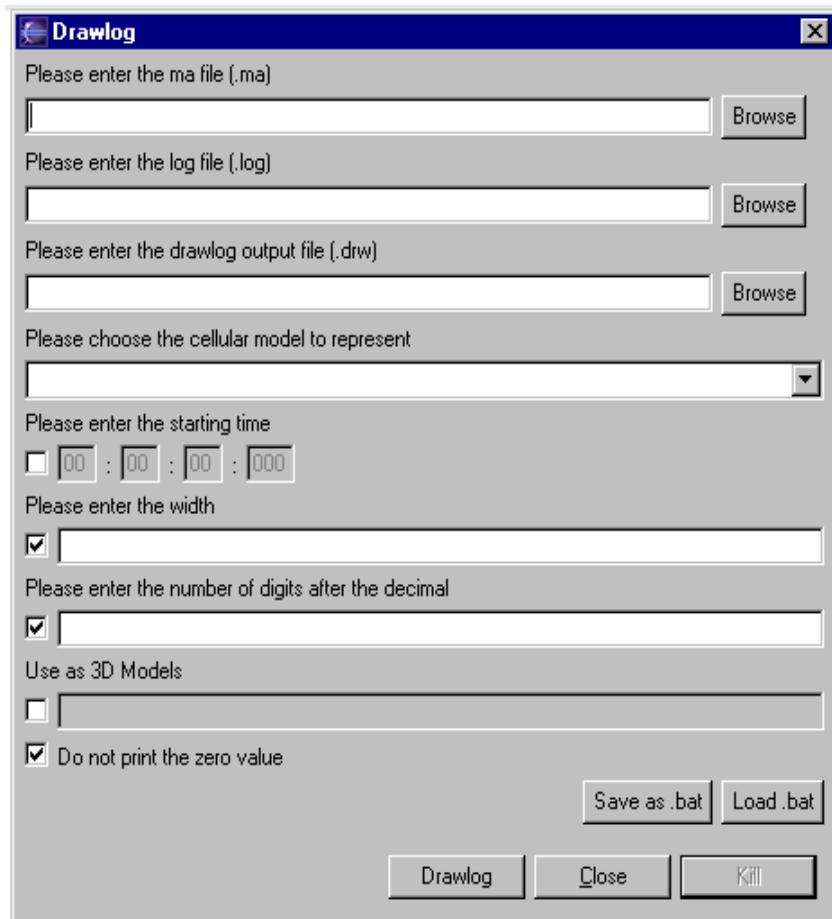


Figure 66: *Drawlog* panel

The DrawLog utility is used to view the state of a cellular model after each simulation cycle as the simulation advances. Using the log as input, drawlog parses the Y messages to update the state of each cell in the model. When a simulation cycle finishes, the state of the whole model is printed.

To start the drawlog tool, you can click on the drawlog button . This will bring up a panel (shown in Figure 66 where you can specify your parameters for running the drawlog just like the simulation button).

You can enable each of the parameters by checking its respective box next to its name. Parameters such as .ma, .log and the output file .drw, are required inputs. If you know the name of these files that are in your project, you can type them in. If you would like to choose from a list of files, you can select the “browse” button that will present you with a list of all the file types.

Once a .ma file is selected, a drop down menu will be available from the cellular model text field allowing the user to choose the desired model to be drawn. The other parameters that can be entered are the stop time, the width and precision used to represent numeric value, the number of slice shown when representing a 3D model and the choice to print the zero value.

The stop time is also separated into 4 different types - hour, minute, second and millisecond. To enter a particular type of the stop time, you can simply enter the time or you can use the up/down arrow from the keyboard to set the desired time. This time indicates when you want the simulation to stop.

Similar to the SIMU button, an user has the choice to save the settings used in the drawlog to a batch file. When you save settings to a file/batch, you will have to specify the name of the file you want to save it to. When you load settings from a file/batch, you will specify its location. By clicking proceed, the simulation will start and send all the information to the bottom component of the panel. Clicking on the output text and moving up and down with the up and down keys, will let you review the simulation and its results. If your coupling file and/or events file is non existent or corrupted, the program will send an error pop up window to tell you. (Note – you must be in the current project of the project you want to run the draw log.)

Once a .ma file is selected, a drop down menu will be available from the cellular model text field allowing the user to choose the desired model to be drawn.

The drop down menu is blank (empty) when a .ma file is chosen if working with a DEVS model. Must indicate to use a Cell-DEVS model.

CD++ Modeler

To launch the CD++ Modeler, click on the CD++ Modeler button . The CD++ Modeler is a GUI, which is used for creating atomic models and coupled models for the ND-C++ tool. The basic function of the GUI includes: create atomic model, create coupled model, retrieve parent class of the coupled model, and run external DOS-style command. The GUI also includes a simple text editor. This GUI is shown in Figure 67. For more information about the CD++ Modeler please refer to the user manual in Appendix C (DRAFT VERSION).

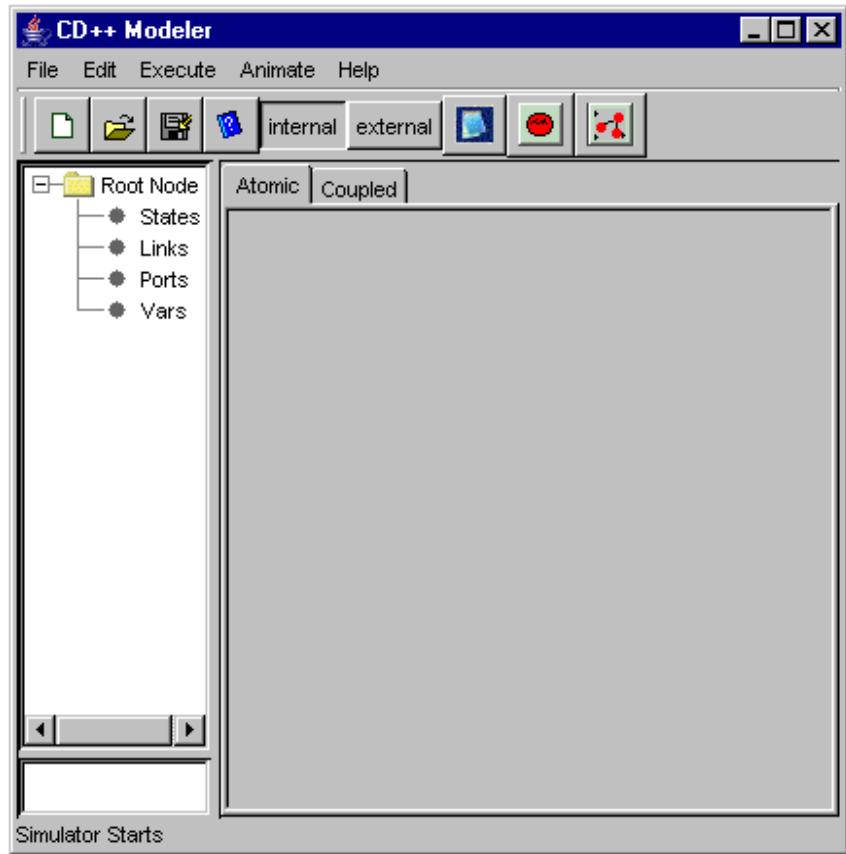
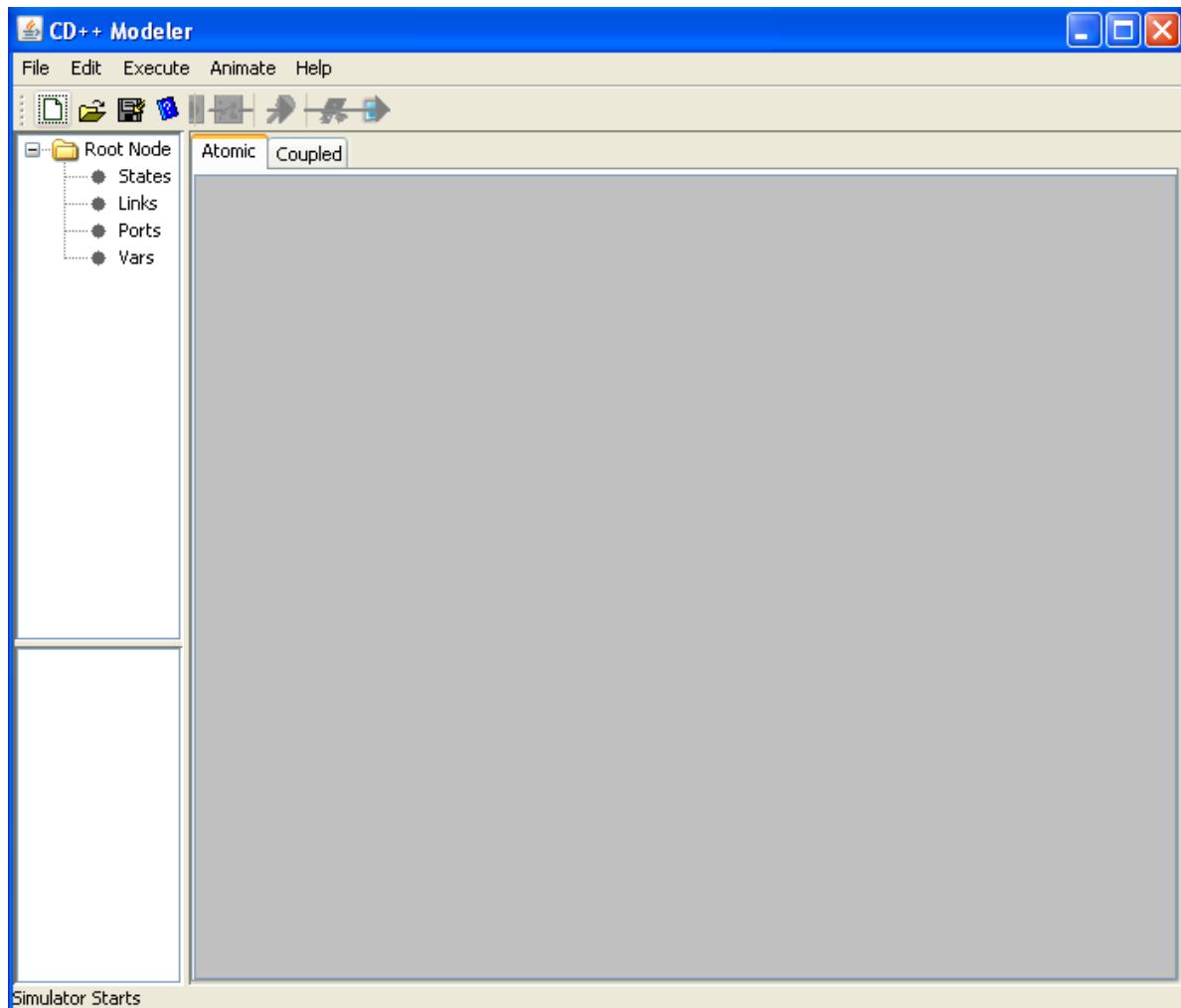


Figure 67: *CD++ Modeler* panel

This GUI is shown in Figure 67.

This GUI is not the same as the GUI that was opened from the current CD++ Builder. The buttons are missing as well, some of the buttons are disabled when all the buttons seem to be enabled on the figure seen on the documentation.



CD++ Modeler from the plug in snapshot

Simulation Server

This document describes briefly the steps to execute CD++ in server mode, and how to carry out a simulation over a network (the simulator found in the distribution webpage DOES NOT provide this service. If interested, contact Gabriel.Wainer@sce.carleton.ca). These services can be installed under a Linux platform

The first step to run a simulation is specifying the port where CD++ will be listening. In Linux you can do that (Figure 68), adding a new line in /etc/services database. The line format is as follows:

<name service> <port number>/tcp

In our case, the service name is *simulator* and the port number can be chosen by the user (it must be a free port). It is strongly suggested that you use port numbers above 1024 to avoid port collisions. For instance,

```

IW /etc/services
# Kerberos (Project Athena/MIT) services
# Note that these are for Kerberos v4, and are unofficial.
#
klogin      543/tcp          # Kerberos `rlogin'
kshell      544/tcp          # Kerberos `rsh'
kerberos-adm 749/tcp         # Kerberos `kadmin' (v5)
kerberos4    750/udp         kdc   # Kerberos (server) udp
kerberos4    750/tcp          kdc   # Kerberos (server) tcp
kerberos-master 751/udp       # Kerberos admin server udp
kerberos-master 751/tcp       # Kerberos admin server tcp
krbupdate    760/tcp          kreg  # BSD Kerberos registration
kpasswd     761/tcp          kpwd  # BSD Kerberos `passwd'
eklogin     2105/tcp         # Kerberos encrypted `rlogin'
#
msft-gc-ssl 3269/tcp        # Microsoft Global Catalog with LDAP/SSL
#
# Unofficial but necessary (for NetBSD) services
#
supfilesrv  871/tcp          # SUP server
supfiledbg  1127/tcp         # SUP debugging
#
# AppleTalk DDP entries (DDP: Datagram Delivery Protocol)
#
rtmp        1/ddp            # Routing Table Maintenance Protocol
nbp         2/ddp            # Name Binding Protocol
echo        4/ddp            # AppleTalk Echo Protocol
zip         6/ddp            # Zone Information Protocol
#
simulator  6002/tcp          # CD++ service

```

Figure 68: Adding a new line in /etc/services

in the `/etc/services` file makes the `simulator` listen to the TCP port 6002 for incoming simulation requests.

CD++ server has a daemon behavior, it uses a `fork()` system call to produce a child process before running a specific simulation and it dies when the simulation is over. The child process that it has forked recently takes the place of its parent and stays listening through the desired port. When a new incoming simulation arrives, the described process is repeated; the CD++ server is forked and the simulation request is satisfied. In the case of concurrent requests, the port can queue up to 5 simulation requests and the others will be rejected. The simulation service never dies.

To run CD++ in server mode you have to invoke it without parameters (Figure 69) the executable file name is `simu`. Keep in mind that CD++ puts in `stdout` (commonly the screen) some information; so remember to redirect the default output to a file.

```

cd++# ./simu
cd++# ps
  PID TTY      TIME CMD
 5070 pts/1  00:00:00 bash
 5080 pts/1  00:00:00 simu
 5100 pts/1  00:00:00 ps
cd++#

```

Figure 69: Shows how to execute CD++ in server mode. Note that the "simu" process remains in execution after the command has been issued

When a simulation is executed, a `model .ma`, an `event list` (optional) and a `stop time` (optional) can be specified. The way that you can do this is sending a stream to the TCP port number (which was

setting before) as follows:

1. Send the model (.ma) text file, line by line.
2. Send a delimiter line using only a dot character (".").
3. Send the event list (.ev) file (send a blank line if your top model does not need external events).
4. Send a delimiter line using only a dot character (".")
5. Send a line specifying the stop time (00:00:00:00)

CD++ returns the result thought the same TCP port with the following format

1. The log file (X,Y,* and done messages among components)
2. A delimiter line using only a dot character
3. The output file

Note that a line is a string with both end line and carriage-return characters and you cannot specify any CD++ switch.

The simulator server requires a client side. The client can be downloaded from CD++ distribution webpage, and it requires the Java Virtual Machine version 1.4.1 to execute.

To start the client, download it from <[>](#)

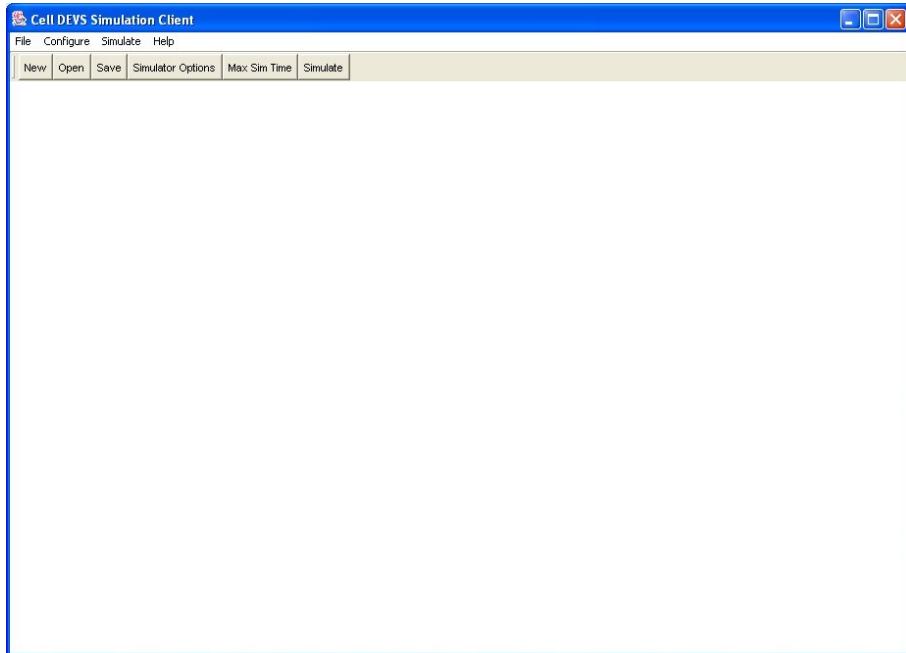


Figure 70: Simulation Client Display Window

To open a new model, select “New” from the “File” menu (or press “New” on the toolbar).

An empty child window should be displayed as follows:

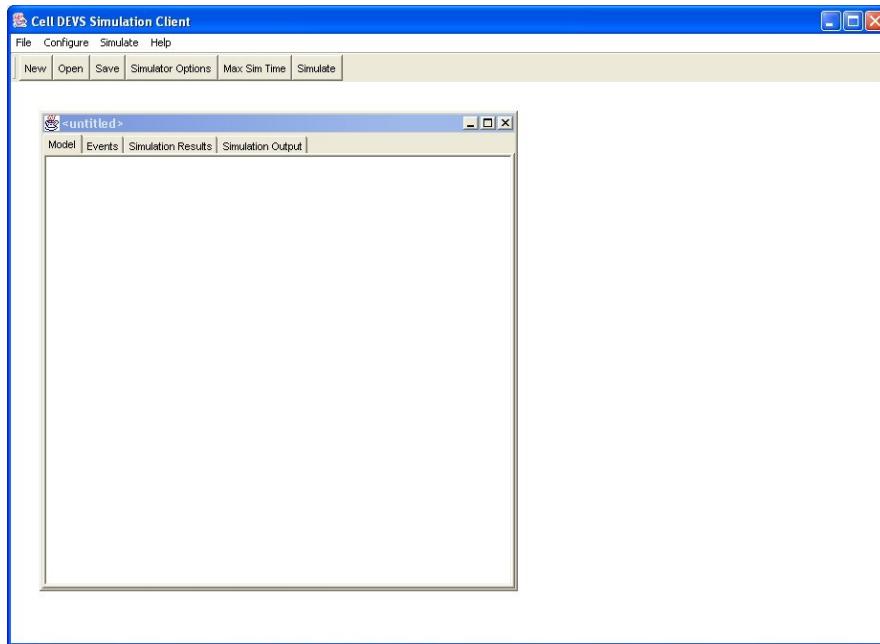


Figure 71: Creating a new empty Child Window

To open file in a new window

1. Select “Open” from the “File” menu (or press the “Open” button on the toolbar).
2. Select the file to be opened (refer to Figure 72).
3. Ensure the “Open File in New Window” checkbox is checked.
4. Press “Open”.

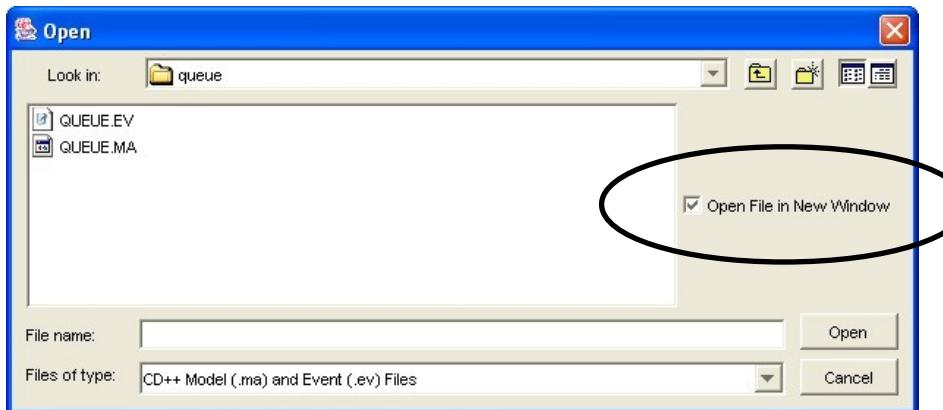


Figure 72: Open File Dialog box

Note: Only files with the extension “.ma” or “.ev” will be loaded into a window. If the file does not have these extensions, then the application does not know the file type, and will not know where to load it.

Note: If there is a model window already open and the “Open File in New Window” checkbox is not selected, the selected file will be opened in the existing window.

The new file should appear in its own window as follows:

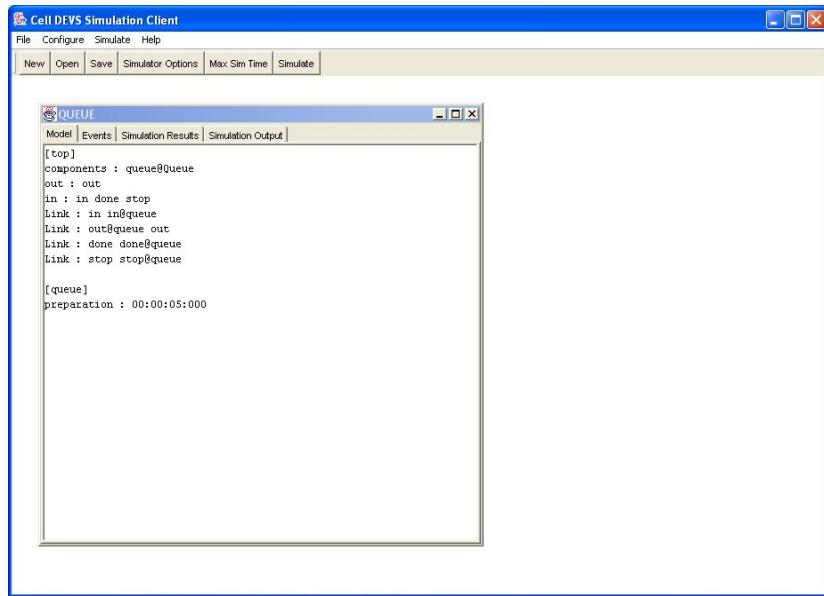


Figure 73: File opened in individual window

Opening a file in an existing window

1. Ensure that a child window is selected.
2. Select “Open” from the “File” menu (or press the “Open” button on the toolbar).
3. Select the file to be opened.
4. Ensure the “Open File in New Window” checkbox is not checked (Figure 74):



Figure 74: Uncheck the “Open File in New Window” checkbox

5. Press the “Open” button:

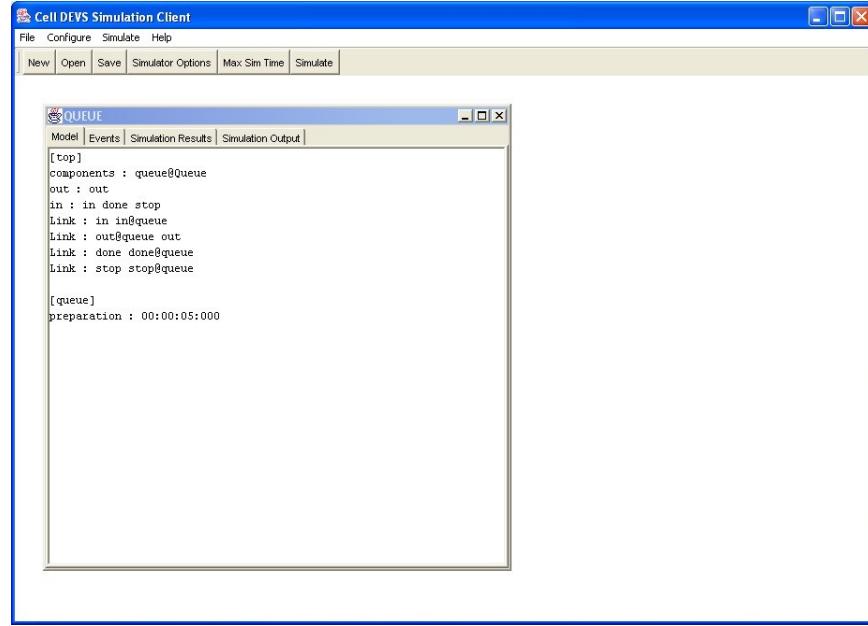


Figure 75: File opened in an existing window

To Save a file

1. Ensure that the window with the file to be saved is displayed:

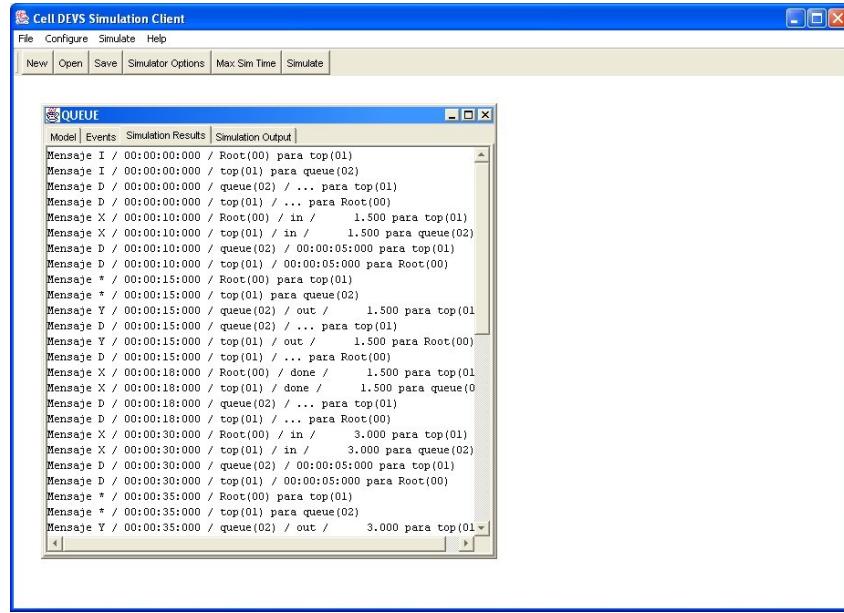


Figure 76: File to be saved is displayed

2. Select “Save” from the “File” menu (or press the “Save” button on the toolbar).
3. Select the file to be saved:

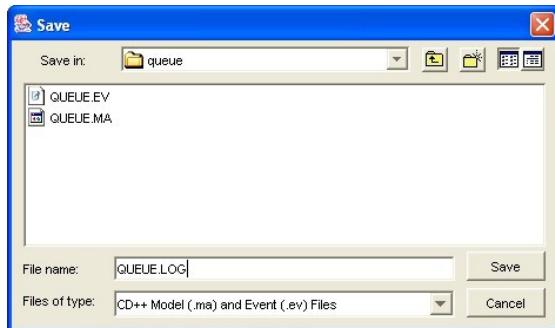


Figure 77: Selecting the file to be saved

4. Press the “Save” button.

To edit the simulation options

1. Select “Simulator Options” from the “Configure” menu (or press the “Simulator Options” button on the toolbar). The following dialog box is displayed:



Figure 78: Editing simulation options

2. Enter the IP address and port of the CD++ Simulator Server.
3. Press the “OK” button.
4. Select “Max Sim Time” from the “Configure” menu (or press the “Max Sim Time” button on the toolbar). The following dialog box is displayed:



Figure 79. Setting the Max Sim Time

1. Enter the maximum simulator time using the CD++ time format.
2. Press the “OK” button.

To execute a simulation,

1. Select “Simulate” from the “Simulate” menu (or press the “Simulate” button on the toolbar).
2. The model and event files will be sent to the simulation server. After the model is executed, the results (log and output files) will be displayed in the window:

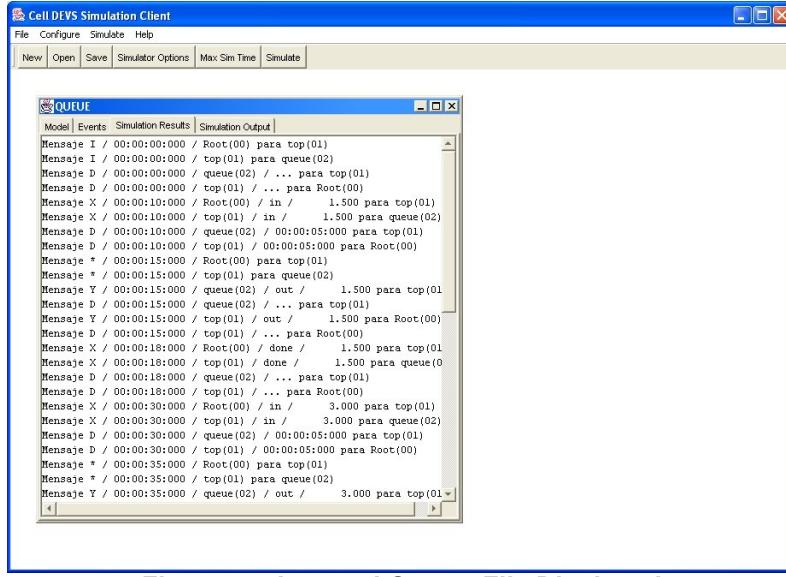


Figure 80: Log and Output File Displayed

Note: A valid IP address and port is required for the simulation to be executed.

Note: If the simulator executes the model, but no results are return, the following is displayed:

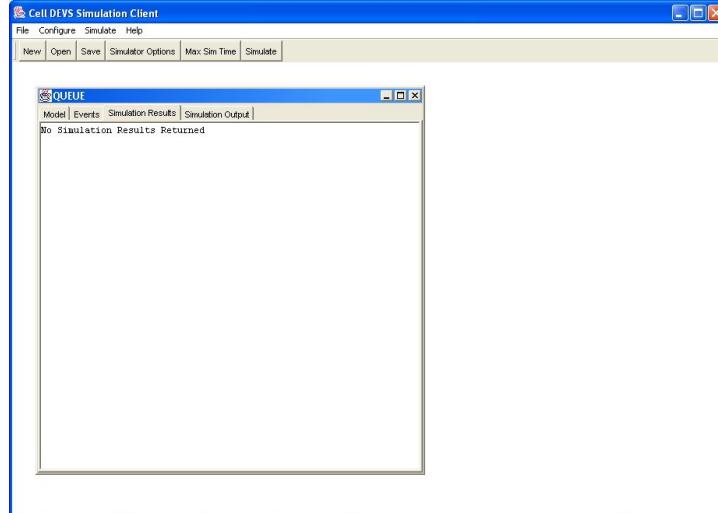


Figure 81. Empty display due to no returned results

The service link is not provided to be tested.

Parallel simulator

To run CD++, type the following command:

```
./mpirun -np n ./cd++ [-ehlmotdvhfrspqw]
```

Here n indicates the number of machines that will be required. It is important this is the same number of machines specified in the partition file or the simulation will not work.

Usage:

```
./cd++ [-ehlmotdvhfrspqw]
e: events file (default: none)
```

```

h: show this help
l: logs all messages to a log file (default: /dev/null)
L[I*@XYDS]: log modifiers (logs only the specified messages)
m: model file (default : model.ma)
o: output (default: /dev/null)
t: stop time (default: Infinity)
d: set tolerance used to compare real numbers
p: print extra info when the parsing occurs (only for cells models)
D: partition details file (default: /dev/null)
P: parallel partition file (will run parallel simulation)
v: evaluate debug mode (only for cells models)
b: bypass the preprocessor (macros are ignored)
f: flat debug mode (only for flat cells models)
r: debug cell rules mode (only for cells models)
s: show the virtual time when the simulation ends (on stderr)
q: use quantum to compute cell values
y: use dynamic quantum (strategy 1) to compute cells values
Y: use dynamic quantum (strategy 2) to compute cells values
w: sets the width and precision (with form xx-yy) to show numbers

```

Figure 82. CD++ command line options

The command line options allowed are:

- efilename**: External events filename. If this parameter is omitted, the simulator will not use external events. The format for external event files is described in section 6.3
- lfilename**: Log filename. When this parameter is specified, all messages received by each DEVS processor will be logged. If filename is omitted (only -l is specified) all log activity will be sent to the standard output. But if a filename is given, one log file will be created for each DEVS processor. The file **filename** will list all models and the name of the corresponding logfiles. These file will be named **filename.XXX** where XXX is a number. When this option is used and no addition log modifiers are defined, all received messages are logged.

The log file format is described in the section 7.2

- L[I*@XYDS]**: allows to define which messages will be logged. This option is useful to reduce the log overhead. The following messages are supported:

I :	Initialization messages
*	(* ,t) Internal messages.
@:	(@,t) Collect messages
X:	(q,t) External messages
Y:	(y,t) Output messages
D:	(done,t) Done messages
S:	All sent messages

When using drawlog, only Y messages are required. Use the -LY option to reduce execution time.

- mfilename**: Model filename. This parameter indicates the name of the file that contains the model definition. If this parameter is omitted, the simulator will try to load the models from the *model.ma* file.
- Pfilename**: Partition definition filename. A partition file is used to specify the machine where each atomic model will run on. Only the location of the atomic models needs to be specified. CD++ will then determine where the coordinators should be placed.

This file is only required for parallel simulation. If standalone simulation is used, this setting will be ignored.

The format for a partition file is described in section 6.4.

-ofilename: output filename. This parameter indicates the name of the file that will be used to store the output generated by the simulator. If this parameter is omitted, the simulator will not generate any output. If you wish to get the results on standard output, simply write **-o**.

The format for the generated output is described in section [7.1](#).

-Dfilename: debug filename for partition debug information. When this option is used, one file for each LP will be created. This file will list all the identification of all DEVS processors running on it.

-t: Sets the simulation finishing time. If this parameter is omitted, the simulator will stop only when there are no more events (internal or external) to process. The format used to set the time is HH:MM:SS:MS, where:

HH: hours
MM: minutes (0 to 59)
SS: seconds (0 to 59)
MS: thousandths of second (0 to 999)

-d: Defines the tolerance used to compare real numbers. The value passed with the **-d** parameter will be used as the new tolerance value.
By default, the value used is 10^{-8} .

-pfilename: Shows additional information when parsing a cell's local transition rules. The parameter must be accompanied with the name of the file that will be used to store the detail. This mode is useful when a syntax error occurs on complex rules.

The format used to store the output is showed in the section [7.4](#).

-vfilename: Enables verbose evaluation of the local transition rules. For each rule that is evaluated, the result of each function and operator will be showed. In addition, this mode will cause complete evaluation of the rules, i.e. it doesn't use rule optimization. The parameter must be accompanied with the filename that will be used to store the evaluation results.

The format of the output generated when this mode is enabled is described in section [7.5](#).

-b: Bypass the preprocessor. When this parameter is set, the macros will be ignored.

-r: Enables the rule checking mode. When this mode is enabled, the simulator checks for the existence of multiple valid rules at runtime. If this condition is true, the simulation will be aborted. This mode is available in standalone mode.

There are a few special cases to consider: if a stochastic model is used (i.e. a model that uses random numbers generators) it might either happen that multiple rules are be valid or that none of them is. In any case, the simulator will notify this situation to the user, showing a warning message on standard output, but the simulation will not be aborted. For the first case, the first valid rule will be considered. For the second case, the cell will have an undefined value ([?](#)), and the delay time will be the default delay time specified for the model.

If this parameter is not used when the simulator is invoked, the mode is disabled and only will be considered the first valid rule.

-s: Show the simulation's finishing time on stderr.

-qvalue: Sets the value for the *quantum*.

The value used as quantum must be declared next to the parameter **-q**, for example: to set the quantum value as 0.01 the parameter must be **-q0.001**.

If the *quantum* value is 0 or the parameter **-q** is not used, the use of the quantum will be disabled, and the value returned by the local computing function will be directly the value of the cell.

-w: Allows to set the wide and precision of the real values displayed on the outputs (log file, external events file, evaluation results file, etc).

By default, the wide is 12 characters and the precision is of five digits. Thus, of the 12 characters of wide, 5 will be for the precision, 1 for the decimal point, and the rest will be used for the integer part that will include a character for the sign if the value is negative.

To set new values for the wide and precision, the **-w** parameter must be used, followed of the number of characters for the wide, a hyphen, and the number of characters for the decimal part. For example to use a wide of 10 characters and 3 for the decimal digits, you must write **-w10-3**.

Any numerical value that must be showed by the simulator will be formatted using these values, and it will be rounded if necessary. Thus, if a cell has the value 7.0007 and the parameter **-w10-3** is declared on the invocation of the simulator, the value showed for the cell on all outputs will be 7.001, but the internal value stored will not be affected.

Utility programs

The tools presented in this section are available to use in CD++Builder. Select the Cell-DEVS perspective from the Perspectives menu.

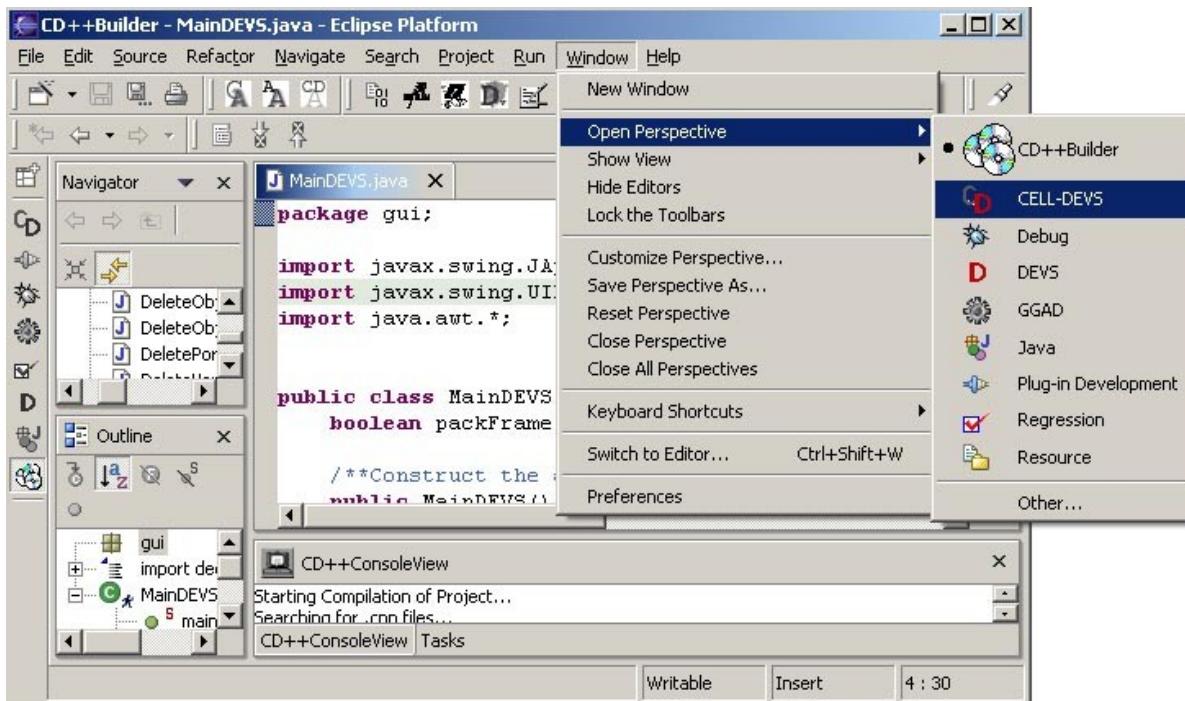


Figure 83. Cell-DEVS perspective

The toolbar contains buttons activating the different utilities presented in this section.



Figure 84: Cell-DEVS toolbar.

Coupled Animate, Atomic Animate, Cell-Dev Animate

The three animation buttons [in order from left to right] coupled animate, atomic animate and cell-dev animate [in order from left to right], are all sub functions of CD++ Modeler. These buttons are placed in the CD+ + Plug-in for ease of access.

Coupled Animate (A): Allows one to animate coupled models given a log file and a graphical coupled model definition file (*.gcm file). The message values sent/received via ports within a coupled model can be visualized using the *.gcm file of the coupled model. The visualization displays the output values with the corresponding output ports, superimposed on the graphical interpretation of the coupled model. See section 8.16.4 for more information on how to animate coupled devs.

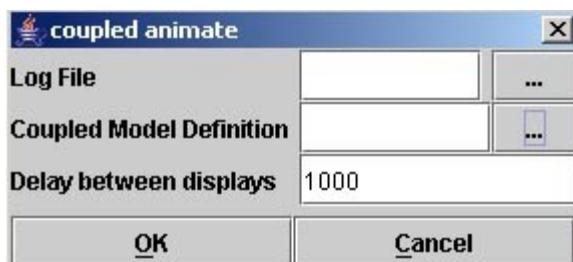


Figure 85: Coupled animate dialog box.

Atomic Animate (A): Allows one to graph the activities of each atomic model within a DEVS model. A DEVS model must include at least one atomic model. After simulating the DEVS model, a .log file is generated. The .log file records all the messages sent between DEVS components. This includes all messages sent/received by all atomic models in the coupled model. The message values sent/received by a specific atomic model can be extracted from the .log file and visualized. Thus the atomic animate requires only a log file as a parameter to be run [Note: *.gem as a parameter will also work]. See section for more information on how to animate atomic models.

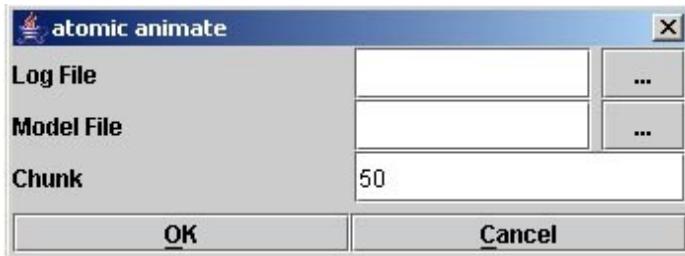


Figure 86: **Atomic animate dialog box.**

Cell-DEVS Animate (CD): Allows one to animate cell values of an atomic cellular model in the form coloured cells that appear at specific time intervals.
See section 8.16.5 for instructions on how use Cell-DEVS animate.

Drawlog

The DrawLog utility is used to view the state of a Cell-DEVS model after each simulation cycle as the simulation advances. Using the log as input, drawlog parses the Y messages to update the state of each cell in the model. When a simulation cycle finishes, the state of the whole model is printed.

From the Cell-DEVS or CD++Builder Perspective toolbar, click . The Drawlog dialog box in Figure 87 will pop up. You can manually fill out the required text boxes. Or, you can choose to load a pre-saved batch file (*.bat) in order to fill out the required fields with a specified set of parameters. To load a batch file, click on *Load.bat*. From the open file dialog box, browse through the list and choose the appropriate drawlog batch file. Make sure you choose a batch file appropriate for the drawlog tool. If you choose the wrong file, a message box will appear showing an error. You have to click on *Load.bat* again, and choose the appropriate file.

Note: Batch files intended for drawlog will contain the word 'drawlog' in it. To check, open the batch file with any text editor, or double click on it within the CD++Builder (Eclipse) environment, and view its contents.

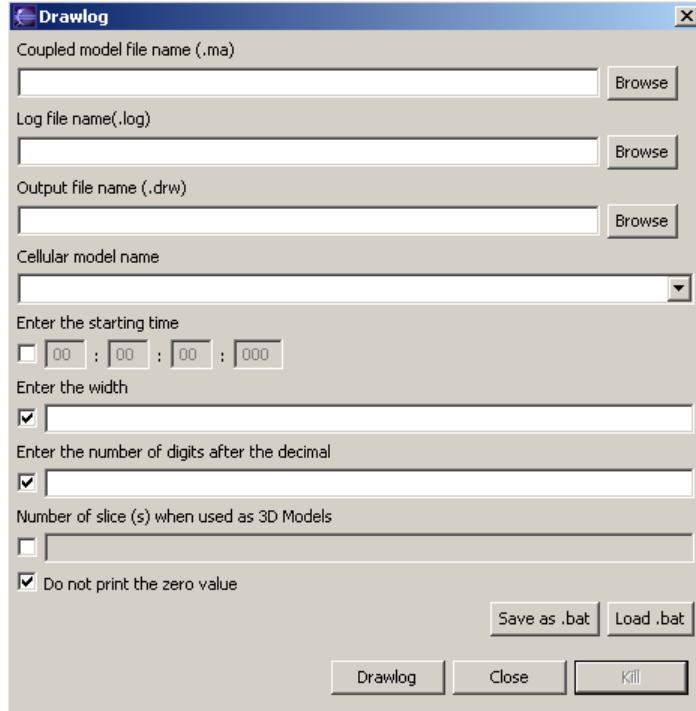


Figure 87: Drawlog Dialog Box

Once you have the required fields filled, you have the option to save the current settings. If you want to save these settings, click on *Save as .bat*.

The data retrieved from the dialog box in Figure 87 corresponds to the command line arguments

described in Figure 88. Using the drawlog button of the CD++Builder toolkit provides the user with the benefit of not having to manually pass command line parameters.

Drawlog can read the log from a file or from the standard input (command line). Its command line parameters are shown next:

```
drawlog -[?hmtclwp0]

where:
?
    Show this message
h
    Show this message
m
    Specify file containing the model (.ma)
t
    Initial time
c
    Specify the coupled model to draw
l
    Log file containing the output generated by SIMU
w
    Width (in characters) used to represent numeric values
p
    Precision used to represent numeric values (in characters)
0
    Don't print the zero value
f
    Only cell values on a specified slice in 3D models
```

Figure 88. Help shown by DrawLog

-?: similar to **-h**.

-m: Specifies the filename that contains the definition of the models. This parameter is required

-t: Starting time. Sets the time for the first state output. If not specified, 00:00:00:000 will be used.

-c: Name of the cellular model to represent. This parameter is compulsory because a .ma file may define more than one cellular model.

-l: Name of the log file. If this parameter is omitted, *Drawlog* will take the data from the standard input.

-w: Allows to define the print width, in number of characters, for numeric values. This width will include the decimal point and sign. For example, **-w7** defines a fixed size for each value of 7 positions. Small numbers will be padded with spaces.

By default, *Drawlog* uses a width of 10 characters. For correct results a width that is bigger than the precision chosen (defined with the parameter **-p**) + 3 is recommended.

-p: Defines the number of digits to be displayed after the decimal point. If a value of 0 is used, then all the real values will be truncated to integer values. This parameter is generally used in combination with the option **-w**.

As an example, consider using the command line arguments **-w6 -p2**. This will set the width to 6 characters, with 2 characters after the decimal point.

By default, *DrawLog* assumes 3 characters for the precision.

-0: When this option is specified, a value of 0 zero will no be shown.

-f: Draws a 3D model as a 2D model. Only the specified plane will be drawn. To draw

plane 0, -f0 should be used.

Figure 102 shows two different ways of starting drawlog. The first uses a log file as input. The second one, instead, takes its input from the standard input.

```
drawlog -mlife.ma -cliffe -llife.log -w7 -p2 -0  
or  
simu -mlife.ma -l- | drawlog -mlife.ma -cliffe -w7 -p2 -0
```

Figure 89. Examples for the invocation to DrawLog

When parallel simulation is used, the standard input can not be directly used by drawlog because log messages may arrive out of order. Therefore, it is necessary to sort the messages first. A utility called logbuffer (described later) has been written for that purpose.

The output format of *DrawLog* will depend on the number of dimensions of the cellular model.

- Output for bidimensional cellular models.
- Output for three-dimensional cellular models.
- Output for cellular models with 4 or more dimensions.

Bidimensional cellular models

A 2 dimensions model will be displayed as a matrix of values. Figure 90 shows a fragment of the output generated by DrawLog for a two-dimensional model of size (10, 10). The number width has been set to 5 and the precision to 1.

Line : 238 - Time: 00:00:00:000									
0	1	2	3	4	5	6	7	8	9
0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
1 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
2 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
3 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
4 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
5 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
6 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
7 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
8 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
9 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									

Line : 358 - Time: 00:00:01:000									
0	1	2	3	4	5	6	7	8	9
0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
1 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
2 24.0 24.0 35.8 24.0 24.0 24.0 24.0 24.0 -6.3 24.0									
3 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
4 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
5 24.0 24.0 24.0 24.0 24.0 39.5 24.0 24.0 24.0 24.0									
6 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
7 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									
8 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 -4.0 24.0									
9 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0 24.0									

Figure 90. Fragment of the output generated for a bidimensional cellular model

In the case of three-dimensional models, a matrix representation will be used. Each matrix is one plane of the cell space. The first plane shown will correspond to $(x,y,0)$, the second one to $(x,y,1)$, and so on.

Figure 91 shows the output of *Drawlog* when used to draw a cellular space of size $(5,5,4)$ with a number width of 1, a precision of 0 and zero values not displayed.

```

Line : 247 - Time: 00:00:00:000
    01234      01234      01234      01234
+---+ +---+ +---+ +---+
0| 1      0|      0| 1      0|
1| 1 1 1  1| 11 1  1| 1| 111  1| 1| 11
2| 1      2| 11     2| 1 11  2| 1
3|         3| 1       3| 1       3| 1
4| 1 1| 4| 1 1| 4| 1 1| 4| 1
+---+ +---+ +---+ +---+
Line : 557 - Time: 00:00:00:100
    01234      01234      01234      01234
+---+ +---+ +---+ +---+
0|      0| 11 11| 0| 1 11| 0| 11
1|      1|      1| 1| 1     1| 1
2|      2| 1 1   2| 1     2| 1 11
3| 1     3| 11    3| 1 11  3| 1 1
4|      4|      4|      4|      4
+---+ +---+ +---+ +---+
Line : 829 - Time: 00:00:00:200
    01234      01234      01234      01234
+---+ +---+ +---+ +---+
0|      0| 1      0| 1 1     0|
1| 1     1| 1      1| 1 11   1| 1
2|      2|      2| 1 1     2| 1
3|      3|      3| 1 1     3| 1
4| 4     4| 1      4| 1 11   4| 1
+---+ +---+ +---+ +---+

```

Figure 91. Fragment of the output generated for a three-dimensional cellular model

For cellular models of 4 or more dimensions, the matrix representation will not be used. Instead, the values for each cell will be listed. The options defined with **-p**, **-w** and **-0** will be ignored.

Figure 92 shows a fragment of the output generated by *DrawLog* for a model of size $(2, 10, 3, 4)$.

```

Line : 506 - Time: 00:00:00:000
(0,0,0,0) = ?
(0,0,0,1) = 0
(0,0,0,2) = 9
(0,0,0,3) = 0
(0,0,1,0) = 21
...   ...   ...
...   ...   ...
(1,9,1,0) = 0
(1,9,1,1) = 4.333
(1,9,1,2) = 0

```

```

(1,9,1,3) = -2
(1,9,2,0) = 6
(1,9,2,1) = 0
(1,9,2,2) = 7
(1,9,2,3) = 0

Line : 789 - Time: 00:00:00:100
(0,0,0,0) = 0
(0,0,0,1) = 0
(0,0,0,2) = 13.33
(0,0,0,3) = 0
(0,0,1,0) = 5.75
...
...
...
(1,9,1,0) = 6.165
(1,9,1,1) = 2
(1,9,1,2) = 0
(1,9,1,3) = 1.14
(1,9,2,0) = 0
(1,9,2,1) = 0
(1,9,2,2) = 5.25
(1,9,2,3) = 0

```

Figure 92. Fragment of the output generated for a model with dimension 4

Batch files intended for drawlog contain DRW in the name

DEVS View

DEVS View is a tool that will create 3D animation from the log file generated by CD++.

Within the CD++ Builder toolkit, from the Cell-DEVS perspective (Note: DEVS view is found in all CD Builder perspectives), click on . This will copy the required DEVS View files into your workplace folder [current project folder] and initiates DEVS View.

DEVS View Tutorial

The following is a step by step tutorial for creating visualization in DEVS. This tutorial shows the steps required to create the bounce visualization described in Section 4.4.2.

- 1) Visit http://www.sce.carleton.ca/faculty/wainer/wbgraf/samplesmain_1.htm download and load the model, *BOUNCING BALL*.
- 2) Select any file within the navigator panel of Eclipse and click on the DEVS View button . The DEVS View program should open and look like the following (See Figure 93):

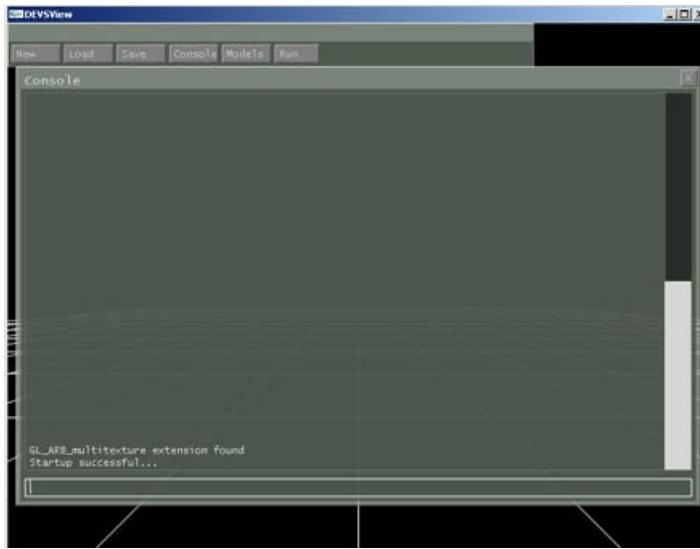


Figure 93: DEVS View Window.

- 3) Create a new visualization by entering the command ‘vis_new bounce’
- 4) Import the .ma file by entering the command ‘vis_import_mafile bouncema.ma’
- 5) Import the log file by entering the command ‘vis_import_logfile bouncelog.log’
- 6) Verify the events were loaded correctly. Enter the command ‘print VIS_EL’, you should see a list of the events loaded from the log file.
- 7) Verify the model was loaded correctly. Enter the command ‘print VIS_ML’, you should see the Cell-DEVS model rebota listed, with the dimensions (16, 20, 1), input ports: /initial and out, output ports: out, neighbourchange
- 8) Save the file as bounce by entering the command ‘vis_save bounce’
- 9) Open the file bounce.vis in wordpad (it handles the end of lines better than notepad).
- 10) Verify the file has the blocks described in Appendix C
- 11) Search for ‘CellSpaceSize’ and set the values of ‘X’, ‘Y’, and ‘Z’ in the complex block to 25.0, 25.0, and 1.0 respectively. The result should look like this:

```
{CellSpaceSize
  {X F 25.0 }
  {Y F 25.0 }
  {Z F 1.0 }
}
```
- 12) Search for ‘Label’ and the set the value of ‘Label’ to ‘’0’ (The ‘’0’ sequence is equivalent to an empty string). The 3d text drawing implementation is currently too slow to use with a large cell space, so erasing the label speeds up the program significantly. The result should look like this:

{Label S }0 }

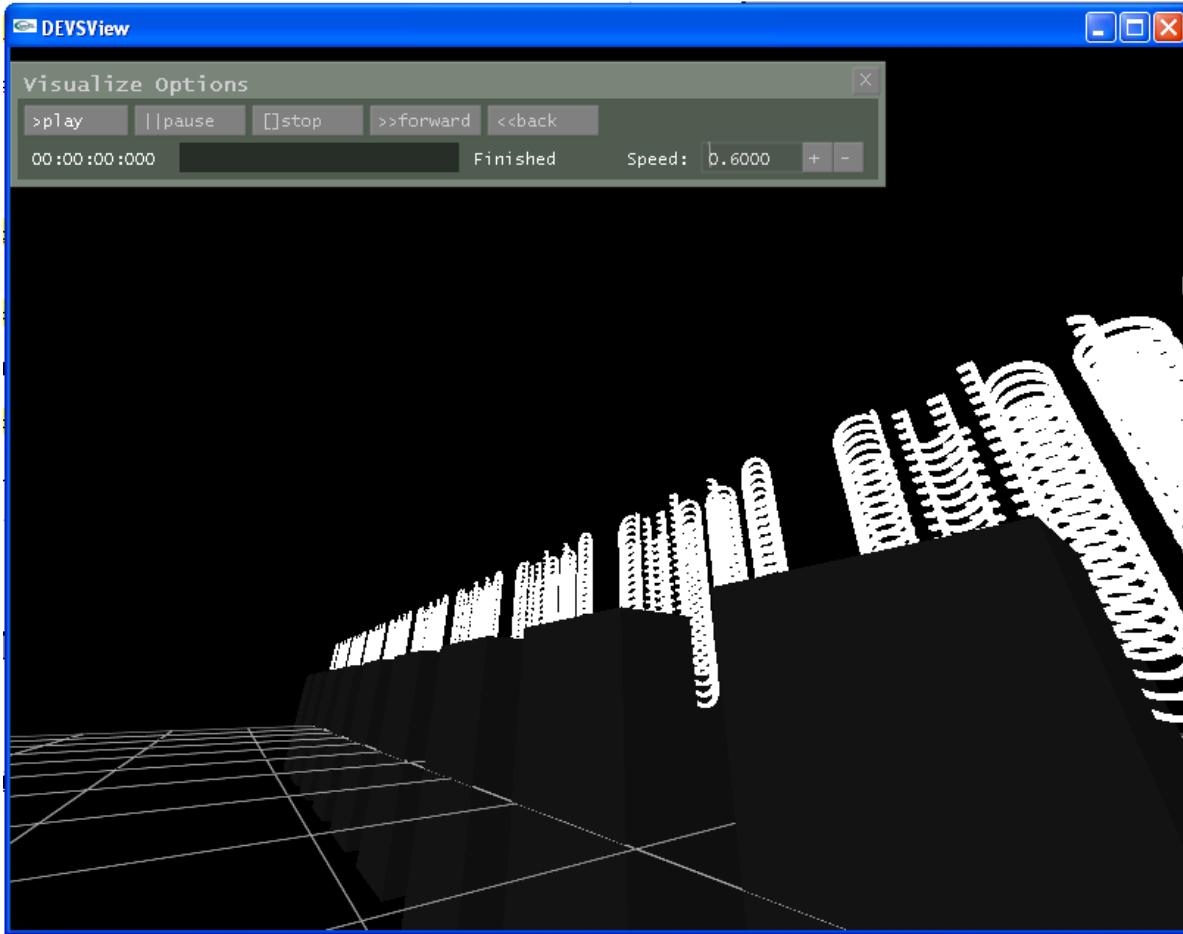
- 13) Save the file and return to DEVSView
- 14) Load the bounce visualization file by entering the command ‘vis_close’ followed by ‘vis_load bounce’
- 15) Click on the Models button in the toolbar. Select the rebota model and click on the ‘Edit...’ button.
- 16) Click on the ‘Add...’ button next to the visual state list. Erase the label of the visual state in the Visual state edit panel that pops up. Set the color of the visual state to red (Red=1.0,Green=0.0,Blue=0.0)
- 17) Select the first visual state in the list (The one that you didn’t just set to red). Click on the ‘Edit...’ button next to the visual state list. Set the colour of this visual state to a dark grey(Red=0.1, Green=0.1, Blue=0.1)
- 18) Add a transition rule by clicking on the ‘Add...’ button next to the transition rule list.
- 19) Select S2 in the ‘Go to State’ list. Select ‘Any’ from the ‘If in State’ list. Select ‘out (O)’ (the O means its an output port) from the port list. Select the ‘Range of Values’ option from the ‘If this rule is triggered’ list. For the lower value select 0.01, for the upper value select 20.0. This creates a rule which transitions the cell visual state from black to red whenever a cell outputs a value from 0.01 to 20.0. A cell space is occupied by a ball if its number is greater than 0.
- 20) Add another transition rule, which transitions from S2 to S1 if the cell outputs a value of 0.0 on the ‘out (O)’ port.
- 21) Save the visualization
- 22) Close the Model edit panel and the model list panel. Click on the Run button in the toolbar.
- 23) Use ‘W’, ‘A’, ‘S’, ‘D’ to move the camera. Left click on the window and drag the mouse to rotate the camera.
- 24) Press play to view the visualization. You should see an animation similar to what is shown in Figure 22.

For further information on how to use DEVS View browse to [eclipse_root_folder]/plugins/CD+
+Builder_1.1.0/DEVSview/ and open the document file *Venhola Report Final [DEVSview Documentation].doc*.

When vis_import_logfile bouncemalog.log is typed on the console, “Unable to add model to model list” is being displayed

- Verify the file has the blocks described in Appendix C
- Appendix C has nothing to do with the block of codes in the bounce.vis file.
 - Press play to view the visualization. You should see an animation similar to what is shown in Figure 22.

Figure 22 does not have any simulations
Simulation not running



LTRANS (Lattice Translator)

Ltrans is a tool that implements the two functions mentioned before. This implementation only works in 2D models and only using nearest neighborhood. The last restriction is a necessary condition so that LTRANS works correctly. CD++ has a specification language to define the cells behavior based on rules and a neighborhood definition; LTRANS translates hexagonal or triangular rules to square CD++ compatible rules. LTRANS receives a set of rules based on a hexagonal or triangular geometry and translates it in rules based on square geometry to be included in a model to be simulated with CD++.

Within the CD++Builder toolkit, from the Cell-DEVS perspective, click on . This will invoke the dialog box in Figure 94.

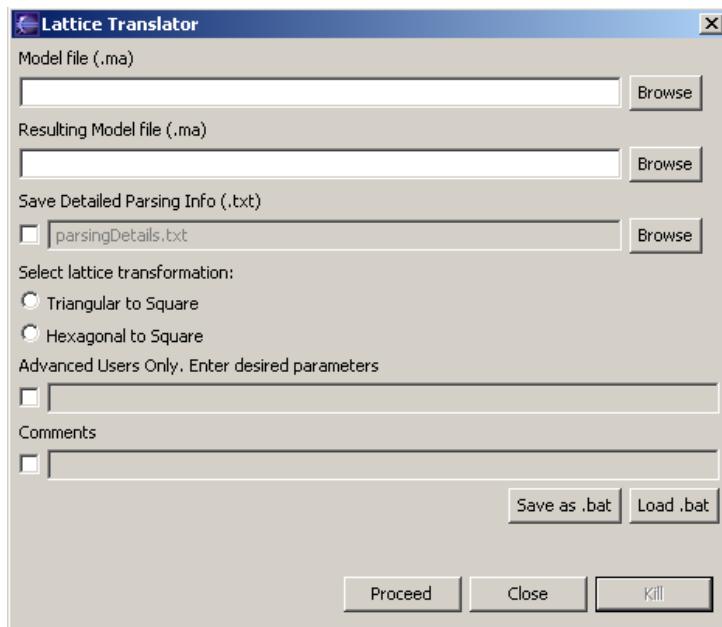


Figure 94: Lattice Translator Dialog Box

You can click on *Load.bat* and load an existing batch file. If an appropriate batch file does not exist, you can manually enter each parameter. The parameters in this dialog box correspond to equivalent command line arguments as described below.

Text box for *Model file (.ma)* corresponds to **-m**.

Text box for *Resulting Model file (.ma)* corresponds to **-o**.

Save detailed *Parsing Info (.txt)* corresponds to **-p**.

The radio buttons *Select lattice transformation* corresponds to **-t**.

These options are described below as command line arguments.

Advanced users are given the option to input optional parameters as they feel necessary in the text box titled *Advanced Users Only*.

To run Ltrans from command line, type:

Itrans [-hmotp]

where

- h: show help
- m: model file (default : modelH.ma)
- o: model translated file (default: model.ma)
- t: mapping type (default: Hexagonal)
- p: parser debug filename

The command line options allowed are:

-mfilename : Model file. This parameter indicates the name of the file that contains the Rules to be translated. If this parameter is omitted, the simulator will try to load the rules from the modelH.ma file.

-ofilename : Model Translated file. This parameter indicates the name of the file that contains the translated Rules. If this parameter is omitted, the simulator will try to save the rules to the model.ma file.

-t[hexagonal/triangular] : Mapping type. This parameter indicates the type of mapping hexagonal to square or triangular to square.

-pfilename: Shows additional information when parsing a cell's local transition rules. The parameter must be accompanied with the name of the file that will be used to store the detail. This mode is useful when a syntax error occurs on complex rules.

A model file is used to define the rules to be translated. This file consists in a set of rules based on hexagonal o triangular geometry. The language used to modeling cell's behavior in a hexagonal or triangular geometry is the same that is used in CD++ but the only difference is the way that a neighbor is referenced. In CD++ a cell (belonging a 2D space) is referenced using a tuple (x,y) where x (row) and y (col) are relative position of a cells. As Ltrans only support nearest neighbors, was necessary define nearest neighbors for hexagonal and triangular geometry. Figure 95 shows the way to define nearest neighbors in each geometry. For both geometries each nearest neighbor is referenced using [n] where n is the number assigned to each nearest neighbor.

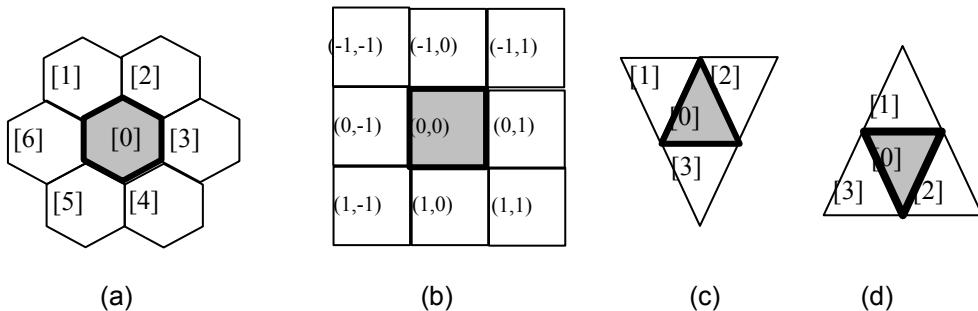


Figure 95. (a) nearest neighbors used for hexagonal geometry. (b) nearest neighbors used for square geometry (used in CD++). (c) and (d) nearest neighbors used for triangular geometry, note that there are two different kind of nearest neighbor and depends on the orientation of the cell

The cellular space in a Cell-DEVS model is named grid or lattice. This lattice is a homogeneous and regular set of cells with a specific geometry. CD++ only allows square geometry, but there are others kind of geometries that could be used to modeling different phenomena. These are:

- **Triangular:** The advantage of this type of geometry is that every cell has a limited number of nearby neighbors (three) which in some models is important. On the other hand, the disadvantages are the difficulty of representation and visualization.
- **Hexagonal:** The advantage of this type of geometry is the higher **isotropy**, that means that the simulations are more natural and in some cases it is absolutely necessary to simulate certain phenomena. The disadvantage is that it is very difficult to represent and to visualize.

Bearing in mind these advantages and disadvantages of every geometry a translator was developed to be used in conjunction with CD++, using two geometries translation functions [\[Wei97\]](#) for 2D lattice.

[Akward Sentence] Different kind of functions can be writing but at the time of visualization's results the interpretation could be not easy. The main idea is to use a function that shifts alternate rows in opposite directions, as shown in Figure 96. That function maintains the boundary conditions in the square lattice. The visualization is very simple, ignoring the factor that introduces the shift of every second row, a cell in hexagonal space can be found in the square lattice (see colors).

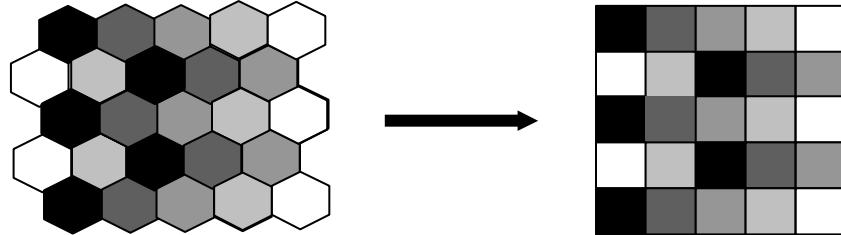


Figure 96. Shift mapping of the hexagonal lattice to the square lattice.

Let (x,y) be the position of a cell, where x represents the row and y represents the column (remember that the function only can be applied in 2D space). The neighborhood relation is transformed differently depending on whether the row index x is even or odd, as shown in Figure 97.

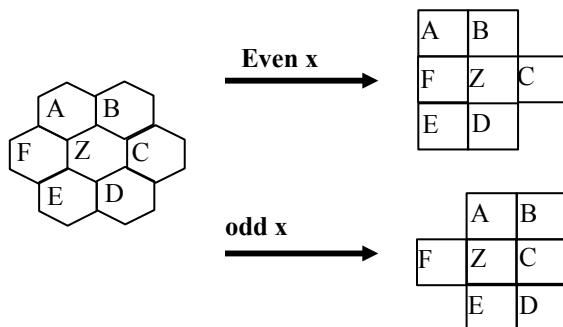


Figure 97. Neighborhood relation in hexagonal to square mapping function

The mapping of the triangular lattice to the square lattice is similar to the shift mapping for the hexagonal lattice. In the triangular case, every second cell has a different orientation. The mapping function is shown in Figure 98. Each row of triangles is mapped to one row of square depending on the parity of $x+y$.

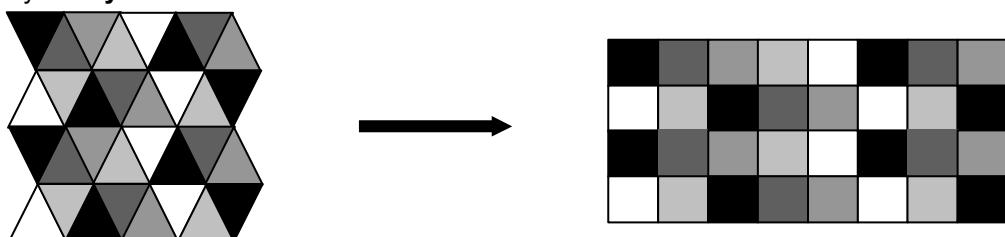
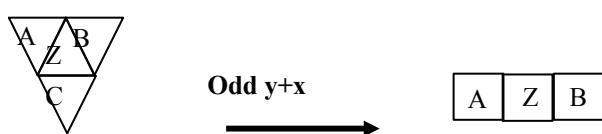


Figure 98. Visualization mapping of the triangular lattice to the square lattice.

The nearest neighborhood mapping is shown in Figure 99 For odd mapping where is cell C????



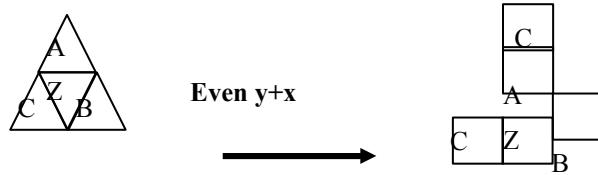


Figure 99. Nearest neighbors in the triangular mapping.

This translated file consists in the set of rules that can be simulated with CD++. Before to simulate the model, it must be completed with the other parameters that define a Cell-DEVS model (space dimension, type of border, default delay, etc.). Note: the neighborhood definition added must be the nearest neighbors as is shown in Figure 95.

The Life Game was presented in Scientific American by the well-known mathematician Martin Gardner. In this game, living cells will live or die. The rules for life evolution are as follows:

- An active cell will remain in this state if it has two or three active neighbors.
- An inactive cell will pass to active state if it has two active neighbors exactly.
- In any other case, the cell will die

The rules that define the Cell's behavior mentioned above in a hexagonal geometry is as follows:

```
rule: 1 100 { [0] = 1 and (truecount = 3 or truecount = 4) }
rule: 1 100 { [0] = 0 and truecount = 2 }
rule: 0 100 { t }
```

Figure 100. File lifegame.rules

Then we use LTRANS to translate the rules in a hexagonal space geometry as follow:

```
C:> ltrans -mlifegame.rules -thexagonal -olifegame.rules.HtoS
```

After applied LTRANS we obtain the following result:

```
rule: 1 100 { ( ( (0,0) = 1 ) and ( ( if((truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) ) = 3 ) or ( if((truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) ) = 4 ) ) and even(cellpos(1)) }

rule: 1 100 { ( ( (0,0) = 1 ) and ( ( if((truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) ) = 3 ) or ( if((truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) ) = 4 ) ) and odd(cellpos(1)) }

rule: 1 100 { ( ( (0,0) = 0 ) and ( if((truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) ) = 2 ) ) and even(cellpos(1)) }
rule: 1 100 { ( ( (0,0) = 0 ) and ( if((truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) ) = 2 ) ) and odd(cellpos(1)) }

rule: 0 100 { t and even(cellpos(1)) }

rule: 0 100 { t and odd(cellpos(1)) }
```

Figure 101. File lifegame.rules.HtoS

Finally using the LTRANS result we construct the final model to be simulated in CD++.

```
[top]
components : life
[life]
type : cell
width : 20
height : 20
delay : transport
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1) life(0,0) life(0,1)
neighbors : life(1,-1) life(1,0) life(1,1)
initialvalue : 0
initialrowvalue : 1 00010001111000000000
initialrowvalue : 2 00110111100010111100
initialrowvalue : 3 00110000011110000010
initialrowvalue : 4 00101111000111100011
initialrowvalue : 10 01111000111100011110
initialrowvalue : 11 00010001111000000000
localtransition : life-rule
[life-rule]
rule: 1 100 { ( ( (0,0) = 1 ) and ( ( if((truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0)))) = 3 ) or ( if((truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0)))) = 4 ) ) and even(cellpos(1)) }
rule: 1 100 { ( ( (0,0) = 1 ) and ( ( if((truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0)))) = 3 ) or ( if((truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0)))) = 4 ) ) and odd(cellpos(1)) }
rule: 1 100 { ( ( (0,0) = 0 ) and ( if((truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0))) < 0 , 0 , (truecount - (if((-1,1) = 1,1,0)) - (if((1,1) = 1,1,0)))) = 2 ) ) and even(cellpos(1)) }
rule : 1 100 { ( ( (0,0) = 0 ) and ( if((truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0))) < 0,0,(truecount - (if((-1,-1) = 1,1,0)) - (if((1,-1) = 1,1,0)))) = 2 ) ) and odd(cellpos(1)) }
rule: 0 100 { t and even(cellpos(1)) }
rule: 0 100 { t and odd(cellpos(1)) }
```

Figure 102. Model file used as CD++ input

Setting Random Initial States – MakeRand

MakeRand is a tool to create a .val file with a random initial state for a cellular model. To use this utility tool from the CD++Builder toolkit, click on  from the cell DEVS perspective. The following dialog box will appear.

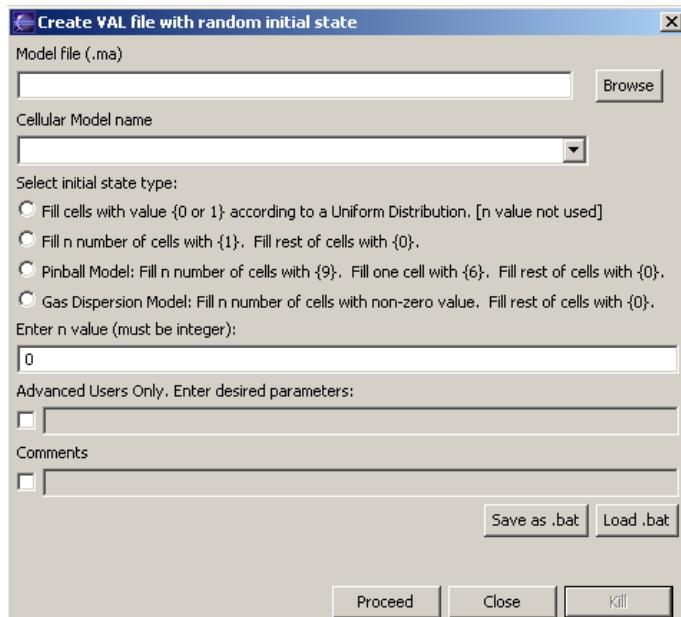


Figure 103. MakeRand command line options

The user is given the option to load a pre-saved batch file (click *Load.bat*), or save the current settings as a batch file (Click *Save.bat*). The fields in this dialog box correspond to the command line arguments as the previously described buttons.

Text field *Model file (.ma)* corresponds to command line argument **-m**

Drop down menu *Cellular Model name* corresponds to command line argument **-c**

Radio buttons for *Select Initial State Type* corresponds to command line argument **-s**

Text field *Enter n value (must be an integer)* is to specify the value of n for the radio button options.

The dialog box gives the user the benefit of not having to enter the parameters manually. The command line arguments corresponding to the fields in the dialog box are described below. It maybe noted, this utility tool can also be accessed as command like input.

Usage:

```
makerand -[?hmcs]

where:
  ?      Show this message
  h      Show this message
  m      Specify file containig the model (.ma)
  c      Specify the Cell model within the .ma file
  s      Specify the value set
        s0  = Use the values 0 & 1 (Uniform Distribution)
        s1-n = Use the value 1 for n cells & 0 for the rest
        s2-n = Makes random states for the Pinball Model
        s3-n = Random states for the Gas Dispersion Model
```

Figure 104. MakeRand command line options

-h: Show help

-?: similar to **-h**.

-m: Specifies the filename for the model definition file (.ma) . To be used to fill the initial values.

-c: Name of the cellular model. This parameter is required because the size of the

model needs to be known.

-s: Specifies the type of initial state to be created:

-s0: For each cell of the model, a value will be chosen randomly belonging to the set {0, 1} with the same probability for each value.

-s1-n: Indicates that the model initially will have n cells with value 1 (distributed randomly according to an uniform distribution) and the rest of the cells will have the value 0. If n is bigger than the quantity of cells of the model, then an error message is displayed and the initial state will not be generated.

For example, if we have a 40x40 cellular and we want 75% of the cells (1200 cells) to have an initial value of 1, and the remaining cells an initial value of 0, then **-s1-1200** should be used.

-s2-n: A value between 1 and 8 will be randomly generated and randomly place inside the cellular space. In addition, n cells will be randomly chosen to represent the walls. The rest of them will have an initial value of 0.

-s3-n: Creates an initial state for the gas dispersion model with n particles.

The output will be created in a .val file with the same name as the model file.

ACTUAL COMMAND DOS PROMPT

```
C:\eclipse\plugins\CD++Builder_1.1.0\internal>makerand -?
makerand -[?hmcir]
```

where:

? Show this message

h Show this message

m Specify file containig the model (.ma)

c Specify the Cell model within the .ma file

i Generate integer numbers randomly. The format is:

-iquantity[+|-]minNumber[+|-]maxNumber

For example: -i100-10+5 will sets 100 cells with integer values in the interval [-10,5] with uniform distribution

r Generate real numbers randomly. The format is:

-rquantity[+|-]minNumber[+|-]maxNumber

and its behaviour is similar to the -i parameter.

As you can compare the two, they do not satisfy

The commands from the manual is correct. But the third s (s2-n) seems to be doing something else than what is in the manual.

-s2-n: A value between 1 and 8 will be randomly generated and randomly placed inside the cellular space. In addition, n cells will be randomly chosen to represent the walls. The rest of them will have an initial value of 0.

The text marked in yellow above is what is said on the manual. On the dialog box itself, it says Pinball Model: Fill n number of cells with {9}, Fill one cell with {6}. Fill rest of cells with {0}.

It creates the .val file as mentioned in the dialog box but there is no cell with the value 6.

This was tested using life model.

Converting .VAL files to Map of Values – ToMap

To use this utility tool from the CellDEVS perspective, click **MAP**. Similar to all the previous utility tools, this tool can also be accessed as command line input.

The tool *ToMap* allows to creates a .map file from a .val file.

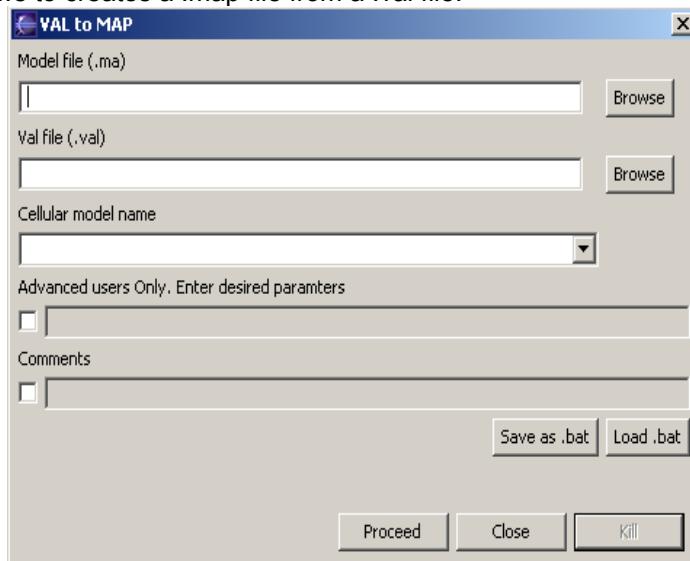


Figure 105. Dialog Box for toMap

The fields from the dialog box correspond to individual command line arguments, giving the user the benefit of not having to type in arguments at the command prompt manually.

The text field *Model file (.ma)* corresponds to the command line argument **-m**.

The text field for *Val file (.val)* corresponds to the command line argument **-i**.

The drop down field *Cellular Model Name* corresponds to the command line argument **-c**.

Advanced users are given the option to enter extra parameters as desired.

Usage:

```
toMap -[?hmci]

where:
  ?  Show this message
  h  Show this message
  m  Specify file containing the model (.ma)
  c  Specify the Cell model within the .ma file
```

i Specify the input .VAL file

Figure 106. Command line arguments for toMap

- ?: same as -h. Shows the command line help.
- m: Specifies the filename (.ma file) with the model definition.
- c: Name of the cellular model.
- i: Specifies the name of the .val file that contains the list of values that it will be used for the creation of the .map file.

ToMap uses all values in the .val file to create a map of values. If the .val file does not specify a value for every cell, then the default value, as specified by the *InitialValue* parameter, will be used.

The output file will have the same name as the .ma file but the extension .map will be used instead.

Bitmap Translator

For Cell DEVS, we can initialize cells by two ways: either initializing directly in ma file or in value file (val file). In many cases, initial data can be available in the form of image, and the goal of this utility is to convert image data into value file. Also we can use multiple images to initialize different planes in z-direction in case of 3D cell DEVS models. The Bitmap Translator tool can be accessed from the CD++Builder toolkit by clicking on . The following dialog box will appear.

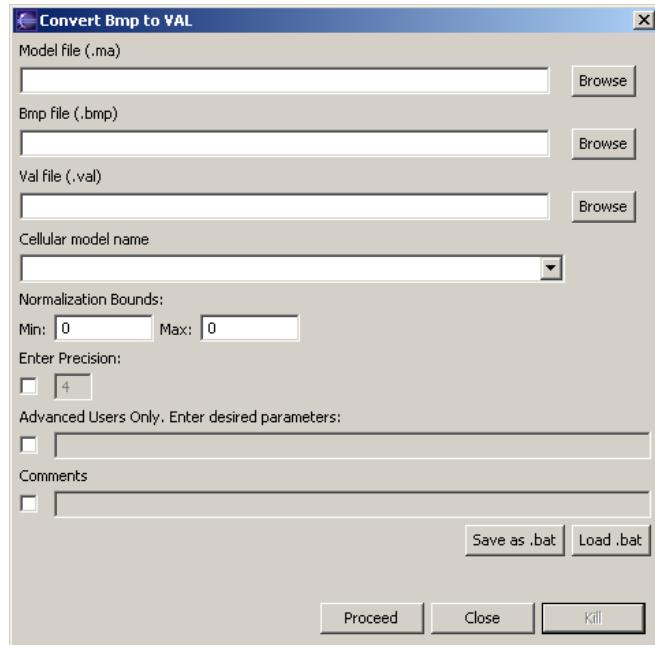


Figure 107. Bitmap translator dialog box

The scope of this utility is standard 24-bit bitmap images. If image data is available in other forms,

we can first convert that into 24-bitmap using any standard Image Viewer tool.

The parameters entered in the dialog box in Figure 107 correspond to the individual command line arguments described in Figure 108.

The tool bmptoval allows converting bitmap image data into value file. The possible parameters are

```
Welcome to Bitmap Translator: Version 1.0
-----
bmptoval -[?hmcbupv]

Where:
?      Show usage help
h      Show usage help
m      Specify file containing the model (.ma)
c      Specify the coupled model having dimension of model
b      Bitmap file to be translated (.bmp)
l      Initial value for normalization
u      Maximum value for normalization
p      Specify precision of index
v      Specify Value file name
```

Figure 108. Possible parameters for bmptoval conversions

- h: show the above help
- ?: same as h
- m: Specifies the file name that contains the definition of coupled model. This parameter is mandatory.
- c: Name of the cellular model to represent. This parameter is mandatory because the file specified with -m can contains the description of many models. Only cellular models are allowed.
- b: Name of the bitmap image file. This parameter can be repeated for multiple bitmap image files. This parameter is mandatory.
- l: Initial value for normalization. This is mandatory. User can convert color index to normalized scale and this parameter represents lower index for that scale range.
- u: Upper value for normalization. This is mandatory. User can convert color index to normalized scale and this parameter represents maximum index for that scale range.
- p: This is optional parameter to define the precision, in integers, of the normalized color index in the value file. Bitmap Translator assumes precision of 4 by default. This parameter should be more than 0.

```
Using one image file
./bmptoval -mEdge.ma -cEdge -bImage1.bmp -l0 -u100 -p3

Using multiple image files
./bmptoval -mEdge.ma -cEdge -bImage1.bmp -bImage2.bmp -l0 -u100 -p3
```

Figure 109. Using one or more image files to convert

Note: Multiple images should be used only for 3D CELL DEVS and size of 3rd dimension should not be less than number of images.

After entering all the necessary parameters in the dialog box shown in Figure 107, click *Proceed* to obtain a valid map of the bmp file.

- There is no bitmap images to create a BMP TO VAL FILE ?
- Why do you need a .ma file when .bmp image file values are translated to a .val file in a text

Convert to CD++

This tool is used to convert an existing .val file into a valid CD++ .ma file. To access this tool click



The following dialog box will appear.

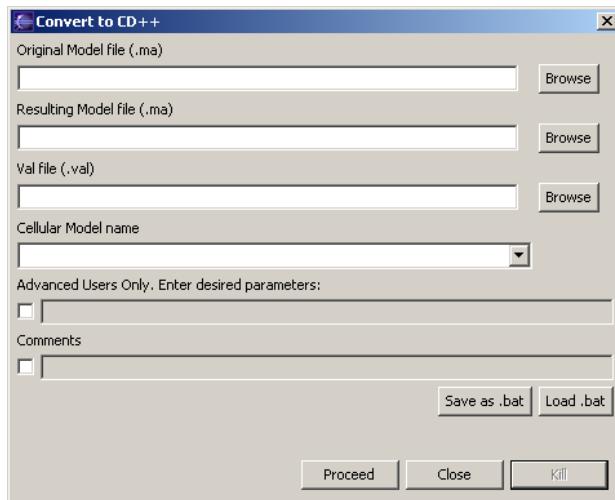


Figure 110: Convert to CD++ dialog box

- No description of the Button
- There is no parameter description
- If .val file is converted to CD++ (.ma file), why do you need two parameters of .ma (Original and resulting model file)

Message counter

The program CONTART accept a simulation log file (.log) and gives the user the number of messages used on that simulation. This program can be accessed from the CD++Builder environment by clicking on



The following dialog box will appear.

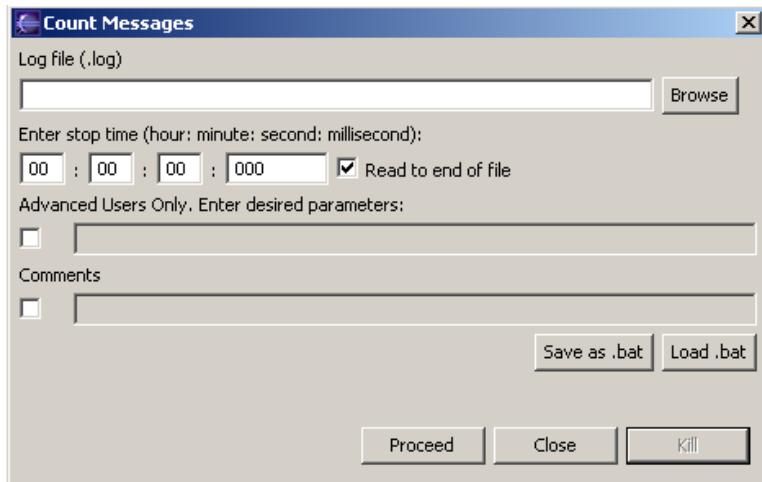


Figure 111. Dialog box for CONTART

Depending on the usage, this tool can receive one or two arguments. The Log file(.log) text box specifies what file to read from. The user is given the option to specify a stop time, or to read all data till the end of the file, by using the corresponding checkbox.

The fields correspond to equivalent command line arguments. The program CONTART receives one or two arguments (depending of the use).

The name of the LOG file

<optional> The Time until you want to count messages.

contart output.log

It also allows the advanced users to enter desired parameters.

This will show you:

```
Archivo a contar mensajes: output.log
Cantidad de mensajes en output.log hasta 00:02:00:000(EOF) -> 264878
#*=72290
#X=48000
#Y=24290
#D=120294
#I=4
```

Figure 112. CONTART output with time not specified

This means that in the log output.log, until EOF (because the title "hasta 00:02:00:000(EOF) means that end of file was reached) there are a total of 264878 messages and this is the detail:

72290 messages of type “*”

48000 messages of type “X”

24290 messages of type “Y”

120294 messages of type “D”

4 messages of type “I”

If you use the second argument, the program will count messages until the indicated time is reached

or EOF (the first that occurs).

With the same log as Example1, we can do:

```
contart output.log 00:01:10:250
```

And this will show you:

```
Archivo a contar mensajes: output.log
Hasta: 00:01:10:250
Cantidad de mensajes en output.log hasta 00:01:10:250 -> 155011
#*=42301
#X=28100
#Y=14201
#D=70405
#I=4
```

Figure 113. CONTART output with time specified

This means that in the log output.log, until 00:01:10:250 simulation time (because the second parameter is in use “hasta 00:01:10:250”) there are a total of 155011 messages and the detailed messages. If you use, for this example,

```
contart output.log 00:03:00:000
```

You will get the same results as in Example1, because EOF will be reached before the 3 minutes indicated on the 2nd parameter, and you will see the title EOF as in example1.

This is useful when you have to compare logs of simulations ended at different simulation-time.

Error Calculation

The program ERRORQ accepts a DRAWLOG output and generates a new output (on standard output –must be redirected to a file--) with six columns. The DRAWLOG output must contain lines and time titles (parameters –f or –e). Other wise, comparison is not possible, because a different offset on each file can occur. This tool can be accessed from the Cell-DEVS perspective by clicking



. This is the *compare drawlog files* button. Once the button is clicked, the following dialog box will appear.

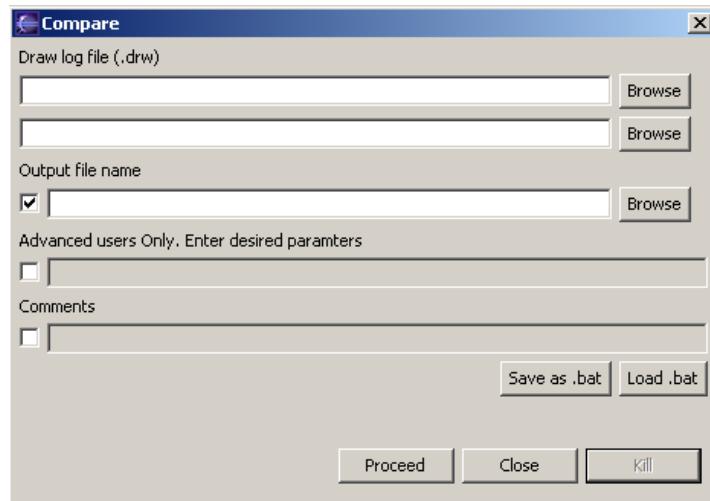


Figure114: Compare Drawlog Files Dialog Box

Enter the two drawlog files to be compared in the first two text boxes. If you want to view the output in a file, enter the name of a .out file. If you want to view the output in the console view, uncheck the checkbox next to the output file textbox. Advanced users are given the option to enter desired parameters. Once all the required fields are checked, click *proceed*. The output is explained below.

The output generated includes:

- 0) Counter of times simulations.
- 1) Time of the block-simulation in comparison
- 2) $(1-s/q)/n$ = Error introduced on the indicated time. Not summarized (only the error generated on this time).
- 3) $\text{sum}[(1-s/q)]/n$ = The same as 2 but accumulated until current time.
- 4) $(s-q)/n$ = The same as 2 but with a different formula. Not summarized (only the error generated on this time).
- 5) $\text{sum}[s-q]/n$ = The same as 4 but accumulated until current time.

Columns showed

Column 0

Line Counter

Column 1

Simulation Time

Column 2

$(1-s/q) / n = \sum (1-s/q_i)/n$ (for $0 \leq i \leq n$ = number of cells)

s_i = Value of cell i on the output without quantum.

q_i = Value of cell i on the output with quantum.

n = number of cells

This value is set to 0 after each block-time.

Column 3

(is the same as 2 but accumulated)

$\text{sum}[(1-s/q)]/n = \sum [\sum (1-s_i/q_i)]_j/n$ (for $0 \leq i \leq n$ = number of cells, $0 \leq j \leq \text{current time}$)

s_i = Value of cell i on the output without quantum.

q_i = Value of cell i on the output with quantum.

n = number of cells

Column 4

$(s-q) / n = \sum (s_i-q_i)/n$ (for $0 \leq i \leq n$ = number of cells)

s_i = Value of cell i on the output without quantum.

q_i = Value of cell i on the output with quantum.

n = number of cells

This value is set to 0 after each block-time.

Column 5

(is the same as 4 but accumulated)

$\text{sum}[(s-q)]/n = \sum [\sum (s_i-q_i)]_j/n$ (for $0 \leq i \leq n$ = number of cells, $0 \leq j \leq \text{current time}$)

s_i = Value of cell i on the output without quantum.

q_i = Value of cell i on the output with quantum.

n = number of cells

When the error is accumulated, means the error accumulated until time indicated in column 1.

The program ERRORQ receives two arguments:

- 1) The name of the original draw output file (original means without quantum)
- 2) The name of the draw output file obtained with quantum

Both files must be generated with DrawLog without the option -f, because the time is needed to synchronize the files and calculate the error. For example:

```
errorq heat.drw heatQ.drw
```

will show you:

Archivo original: Heat.drw quantificado:HeatQ.drw

Dimension detectada del modelo: 10 x 10					
Descripcion de Columnas					
0) Contador de bloques comparados					
1) Tiempo de simulacion del bloque en comparacion					
2) $(1-s/q)/n =$ Error '/' introducido en el bloque time. Sin acumular.					
3) $\sum[(1-s/q)]/n =$ Error '/' acumulado hasta time.					
4) $(s-q)/n =$ Error '-' introducido en el bloque time. Sin acumular.					
5) $\sum[s-q]/n =$ Error '-' acumulado hasta el bloque time.					
t	time	$(1-s/q)/100$	$\sum[(1-s/q)]/100$	$(s-q)/100$	$\sum[(s-q)]/100$
0	00:00:00:000	0.00000000	0.00000000	0.00000000	0.00000000
1	00:00:01:000	0.20257142	0.20257142	3.94700000	3.94700000
2	00:00:02:000	0.25475000	0.45732142	4.71100000	8.65800000
3	00:00:03:000	0.36366666	0.82098809	6.06400000	14.7220000
4	00:00:03:050	0.36103666	1.18202476	6.02600000	20.7480000
5	00:00:04:000	0.46530000	1.64732476	7.31400000	28.0620000
6	00:00:04:050	0.46885000	2.11617476	7.37500000	35.4370000
7	00:00:05:000	0.54695000	2.66312476	8.31600000	43.7530000
8	00:00:05:050	0.55075000	3.21387476	8.37500000	52.1280000
9	00:00:06:000	0.58753333	3.80140809	8.81300000	60.9410000
10	00:00:06:050	0.59160000	4.39300809	8.87400000	69.8150000

Figure 115. Error Outputs

if you do (for example, in MSWindows):

Errorq output1.drw outputqq.drw > error.csv

This will generate a comma separated value (because the program shows the columns separated with commas) file and can be opened with Excel or a similar application.

The coordinates which do not have any numbers in them are given a default value of zero

Extract from Drawlog Files

This tool is meant for extracting specific cells from a drawlog file and listing the values of the cells in an output file. This tool can be accessed from the Cell-DEVS perspective by clicking . Once the button is clicked, the following dialog box should appear.

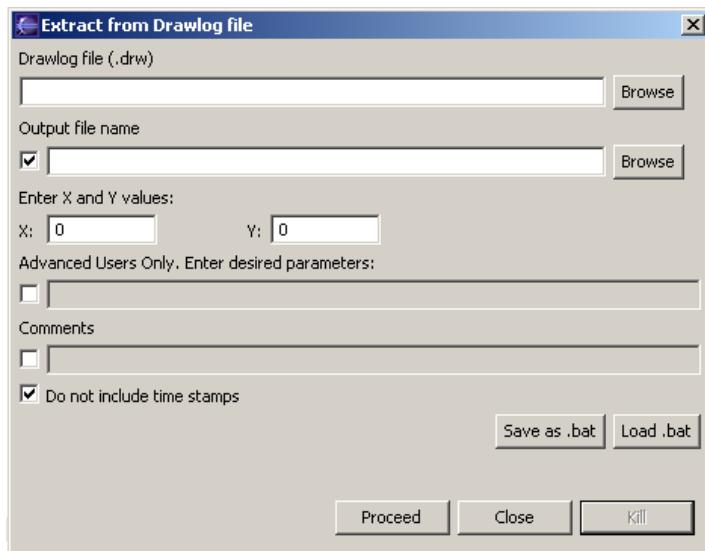


Figure 116: Extract dialog box

Similar to every other button on this perspective, the user may choose to load or save a batch file. If you are using this button for the first time and don't have a pre-saved batch file, you have to enter information for every data field manually.

In the text box labelled *Drawlog file (.drw)*, enter the file from which you want to extract data. You may browse to find the file, or type the name of the file.

Once you fill out the first text box, the text box labelled *Output file name* automatically assumes a file name corresponding to the name of the drawlog file to be extracted from. For example, if you enter *test.drw* for the drawlog file, the automated output file name will be *test.out*. The user can change this file name, or browse to choose an existing output file. If you uncheck the checkbox next to this textbox, the output will be printed on the console view.

The **X** and **Y** values correspond to the location of the cell on the drawlog file. For example, if the drawlog file consists of the two windows in Figure 87, then the extract tool will scan those two windows to find the data value in the (**X**, **Y**) location for both the windows and print it out in the output file or the console view. If **X**=8, and **Y**=2, the outputs will be 24.0 and -6.3.

Advanced users are given the option to enter desired parameters in the next text box.

The last checkbox labelled *Do Not Include Timestamps*, is to determine whether the user wants to view the timestamps or not. If this checkbox is checked, the time stamps are ignored. If it is not checked, the timestamps are printed.

After filling out all the necessary fields, the user must click on proceed to run the program. The following figures demonstrate the file output generated by the life model described earlier.

The drawlog file used in this example is generated using the default .bat (lifeDRW.bat) file that comes with the model life2d.zip.

The output generated in life.out without timestamps:

```
Values extracted from file: life.drw Cell=(8:5)
1
0
1
1
0
```

The output generated in life.out with timestamps:

```
Values extracted from file: life.drw Cell=(8:5)
00:00:00:000 1
00:00:00:100 0
00:00:00:200 1
00:00:00:300 0
```

If the output file checkbox is not checked, the same output will be shown in the console view.

GrafCell

The program GRAFCELL accepts a DRAWLOG file (.drw) and displays graphics representing the data in the file, on the screen with all the cells and the function graphics for each one on a Grid. To use this tool from the CD++Builder toolkit, click on  , from the cell DEVS perspective.

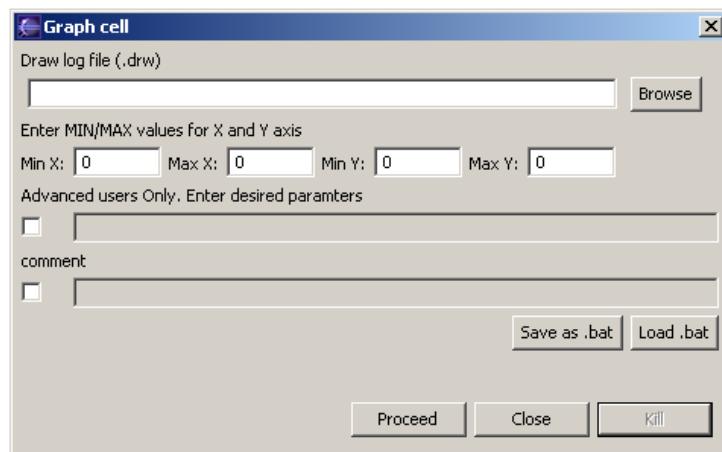


Figure 117. Dialog box for Graph Cell

The program GRAFCELL receives five or six arguments (depending of the use). Each of the text

boxes in the dialog box above corresponds to a command line input to display the graphics.

Enter the name of the drawlog file to be graphed in the first text box. The next 4 text boxes are for min and max values. After entering all necessary parameters, click *Proceed*.

The purpose of this tool is to graphically represent the data of a drawlog file. For example: in Figure 117, the max value of **X** should be 2, since there are only two distinct times being displayed. The min value should be 0, or 1 depending on what the user chooses the origin of the graph to be. The max **Y** value should be 39.5 or above. And the min **Y** value should be -6.3 or lower. When displayed, a 10X10 grid will appear, with each cell on the grid representing each cell from the drawlog display. The points plotted on the cell will represent the relative position of the two distinct time slots.

Advanced users are allowed to enter an additional parameter **-t**. The parameters used for this program are described below.

- 1) The name of the DRAWLOG output file.
- 2) The minimum value for x (most times is 0 –cero—because of the starting time of a simulation is cero).
- 3) The minimum value for y. This is the minimum value that a cell can reach on the simulation.
- 4) The maximum value for x. This is the maximum time of the simulation, converted to milliseconds, but however, not all simulation times are showed on the drawlog, so you will need to adjust this parameter trying with different ones until the correct scale of the grid is showed.
- 5) The maximum value for y. This is the maximum value that a cell can reach on the simulation.
- 6) <Optional> -t With “-t” argument, GrafCell will show the current time of the simulation when drawing. The time will be converted to milliseconds and divided by the default cell delay, but however, not all times simulations are showed on drawlog, so a better adjustment will be necessary.
If the drawlog file does not include the titles and times, a counter will be showed. WARNING: With this option, the drawing can be VERY SLOW, depending on the model size.
- 7)

Arguments 2, 3, 4 and 5 are only to adjust the scale of the grid. You can try different ones to have a nicer view.

```
GrafCell output.log 0 -16.6 35000 94.3
```

This will show you the graphics on a newer window screen. GrafCell will automatically detect the number of cells.

Restrictions: GrafCell will work properly only with nxm (and only with n=m) outputs (with drawlog outputs of one slide (parameter **-f** or **-e**).

To graph an output of a simulation with quantum, it is better to use a DRAWLOG output generated with **-f** option, because **-f** option shows outputs for all time simulations, and that produces better graphics. However, this is a suggestion and the improvement depends on the model.

Note: This tool is currently being updated. There are a few known bugs, which are being fixed. Users are advised against using this tool with 3D models, since this tool is meant for 2D models only. Using it on 3D models will display inaccurate results.

Proceeding with the graph cell gives an error.
 “grafcell.exe has encountered a problem and needs to close. We are sorry for the inconvenience.”
 with the Bounce Model

You cannot type in negative values in the min and max value boxes as well as decimal points. Having negative sign and typing a number gets rid of the negative sign and typing a number with a decimal point gets rid all the values before the decimal point including the decimal point making it a blank box.

Parlog

Parlog is a utility used to assess the parallelism of a running model. It uses the model log as input and counts the number of (*,t) messages received by each LP during a simulation cycle. After a simulation cycle has been completed, a list with the number of messages received by each LP will be printed.

Parlog reads the log from the standard input. *LogBuffer* should be used for correct results.
 Usage:

PARLOG: An utility to determine the level of parallelism
usage: parlog [-?hmP]

where:

? Show this message
h Show this message
P Partition file name

Figure 118. Parlog command line options

- h : Displays help.
- ? : Displays help.
- P: Specifies the partition file name. This parameter is required because parlog needs to know how many LPs are being used.

Figure 119 shows the output generated by parlog with a model running in four machines.

Time/LP 0	1	2	3	4
00:00:00:000	629	626	626	626
00:00:10:000	5	0	2	3
00:00:11:000	12	3	12	14
00:00:12:000	31	7	32	35
00:00:13:000	60	13	62	66
00:00:14:000	99	21	102	107
00:00:15:000	148	31	152	158
00:00:16:000	207	43	212	219
00:00:17:000	276	57	282	290
00:00:18:000	351	73	358	367
00:00:19:000	428	91	436	446
00:00:20:000	509	131	495	486
00:00:21:000	543	192	531	522
00:00:22:000	575	254	563	554
00:00:23:000	603	317	591	582
00:00:24:000	625	376	614	606
00:00:25:000	627	450	625	626

Figure 119: Parlog output for a 4 machines partition.

Parlog commands are not found on the MS DOS
There is no parlog button found on the CD++ Cell-Devs perspective.

Logbuffer

Logbuffer is a utility that buffers log messages received through the standard input, sorts them according to their time, and outputs them to the standard output. It should be used when running *drawlog* or *parlog* piped with the simulator.

To run logbuffer use,

```
logbuffer [-b]  
-bn           Sets the size of the buffer. The default size is 200.
```

Both *drawlog* and *parlog* require that, for correct results to be obtained, that log messages be processed in the order determined by their timestamps. When parallel simulation is run and the log is sent to the standard output, there is no guarantee that messages will be displayed in the same order that they were generated. Therefore, a sorted buffer is needed.

Logbuffer has an internal buffer of a user defined size, which is always kept sorted. When the simulation is started, this buffer is empty. Every new message that arrives is buffered, and no output is sent till the buffer is full. Once it is full, every new message that arrives causes a new message to be sent to the standard output. When the simulation finishes, all buffered messages are sent.



Figure 120: *Logbuffer receives a message with timestamp 3 and then two messages with timestamp 2. Logbuffer sorts and send in the correct order.*

Logbuffer can only guarantee correct results for misplaced messages that occur within a distance smaller than the size of the buffer.

```
>./mpirun -np 4 ./pcd -mcalor.ma -Pcalor.par4 -t00:01:00:000 -l | ./logbuffer -b5000 |  
.drawlog -mcalor.ma -csuperficie -w6-p2 > calor.drw  
  
> ./mpirun -np 4 ./pcd -mcalor.ma -Pcalor.par4 -t00:01:00:000 -l | ./logbuffer -b5000 | ./parlog  
-Pcalor.par4 > calor.p
```

Figure 121: *Running pcd with logbuffer.*

SimuModelica

The SimuModelica button uses MCD++ which is an extension to the N-CD++ toolkit itself, that allows modeling and simulation of simple electrical circuits under the discrete event paradigm. The circuits are developed using the *Modelica* specification language. The simulation environment accomplishes model simulation based on the QSS and QBG theory, providing the extensions needed to the N-

CD++ toolkit to simulate dynamical systems. It can, therefore be used to simulate a model file, with extension *.mo, under the CD++ simulation environment.

To use this feature to run a Modelica *.mo file follow the steps given below:

1. Create a new project following the instructions given in section and name it as **circuit.mo** as shown below.



Figure 122 A new project called 'circuit'

2. Add the Modelica *.mo file into this project following the instructions given in section .
3. If no *.mo file is added, an attempt to run the simulation gives the error in the figure given below.



Figure 123 MO file not found error.

4. After this click the SimuModelica button (), which gives the GUI dialog box given in Figure 124.

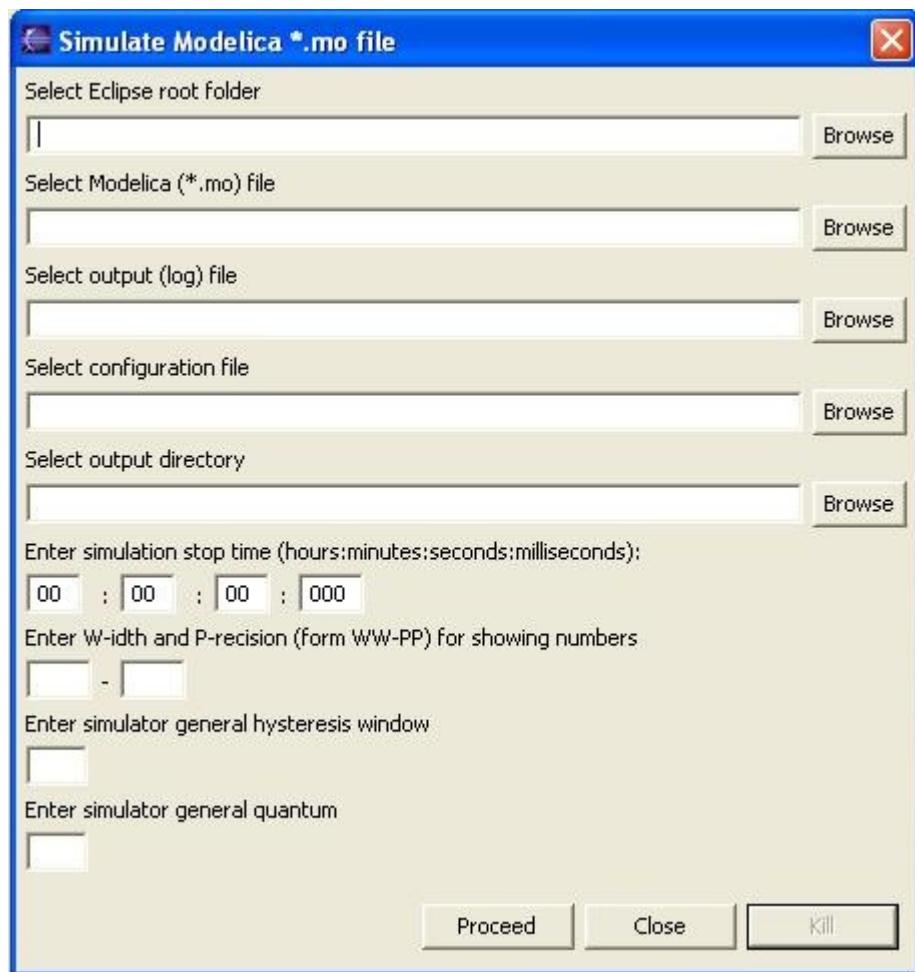


Figure 124: The SimuModelica dialog box

5. Fill in all the parameters and click **Proceed**.
6. The following files will be generated after the simulation is complete.
- 7.

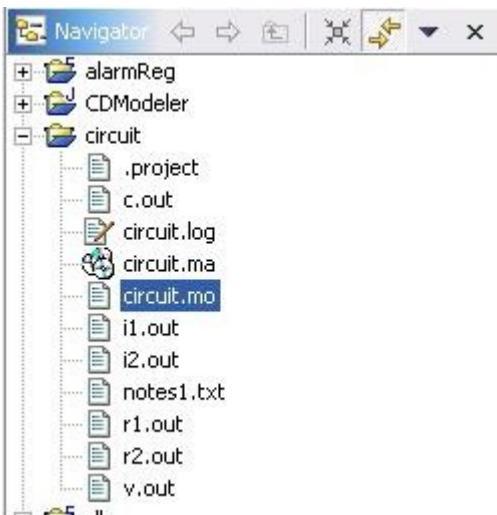


Figure 125 Files generated after simulation is complete.

The first two fields are imperative, while the rest are optional because they end up using default values if left empty. If any fields are left empty apart from the first two, then the following message box pops up.



Figure 126: Message box indicating that some fields were left empty

The following table summarizes the properties of each individual parameter:

Parameter	Description	Required	Default value
Eclispe root folder	Specifies the location of the folder where eclipse is installed.	Yes	n/a
Modelica(*.mo) file	Modelica source file	Yes	n/a

Output log file	File logging all the steps accomplished by the N-CD++ simulator.	Optional	No CSV log files are generated for the models.
Configuration file	Configuration file that specifies particular quantum and hysteresis window values per component. The values described in this file override the general quantum and hysteresis window sizes.	Optional	
Output directory	A unique log file for every component within the circuit will be generated and stored under the directory specified by this option. Every component log file name will be created as <component name>.log. This file will log the time, voltage and current values calculated by the object in CSV format	Optional	/dev/null (no log)
Simulation stop time	The duration for which the simulation runs	Optional	00:01:00:000 (1 minute)
Width and precision	Simulator width and precision to show numbers	Optional	12-05
Simulator Quantum	General quantum value applied to all the quantizable models within the circuit.	Optional	0.01 (floating value)
Simulator Hysteresis	General window size applied to all the quantizable models within the circuit.	Optional	0.01 (floating value)

Cannot find any .mo (Modellica files) to test this button.

CD++Modeler

Using the CD++ Modeler Graphical User Interface (GUI), atomic and coupled models can be created for the CD++ toolkit. The basic functions of the GUI include: creating atomic and coupled models, exporting the models to different formats, and animating the simulations. The GUI also includes: - a simple text editor to directly modify Cell-DEVS models, - the RUN/SIMU commands to execute the simulation with the CD++ toolkit, and - the DRAWLOG command to generate the draw information from the simulation. This section describes in detail how to use the GUI for inputting DEVS model, and for visualizing the results of the Cell-DEVS, Atomic-DEVS, and Coupled-DEVS models.

Since the GUI is coded in Java (JDK1.4.1), it is platform portable and can be run in various environments, such as Eclipse or JBuilder (with JDK 1.4.1). Although this tool is a standalone program, it can also be accessed through CD++Builder by clicking on .

To install and run the standalone GUI:

Note: If you are using CD++Builder, you don't have to install the standalone version. You can access CD++ Modeler directly from the Cellae's perspective, by clicking on .

1) Download the cdModeler.zip file, which contains source code, class files, and some example files. Decompress the cdModeler.zip file in the Windows environment.

2) Install JDK 1.4.1 or above.

3) Set up the JAVA_HOME environment variable to point to the JDK directory.

To do this, modify the PATH environment variable by adding " %JAVA_HOME%\bin; ".

Note: To modify environment variables, go to Control Panel > System > Advanced > Environment Variables.

In the cdModeler directory, double click the run.bat file to start the program.

When the program has started, it will display a dialog as seen in Figure 127.

Using CD++Modeler

Once the CD++Modeler tool is running, the following screen will appear.

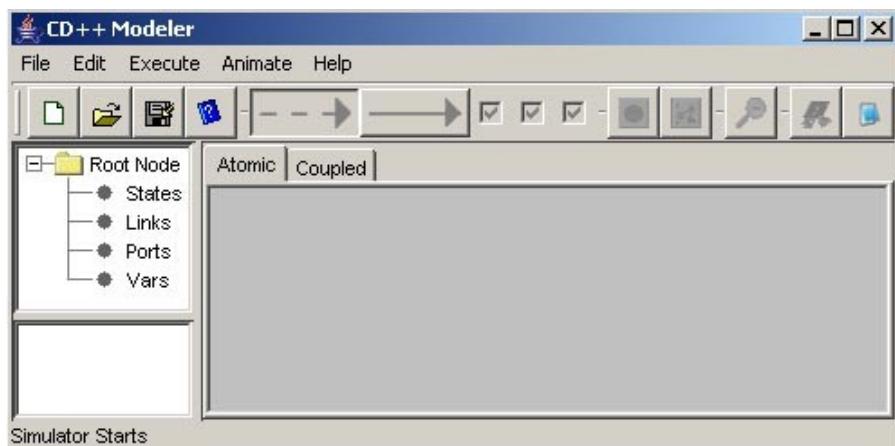


Figure 127: CD++ Modeler window.



Figure 128: CD++ Modeler menu bar.



Figure 129: CD++ Modeler button bar.

The button bar contains the following buttons (in order from left to right):

1. New
2. Open
3. Save
4. Help
5. Internal Link - only used for the atomic model design space
6. External Link - only used for the atomic model design space
7. Show Link Expressions - only used for the atomic model design space
8. Show Link Actions - only used for the atomic model design space
9. Show Link Ports - only used for the coupled model design space
10. Add new Atomic Model Unit - only used for the coupled model design space
11. Add new Coupled Model Unit - only used for the coupled model design space
12. Close Exploded Unit - only used for coupled model design space
13. Simulate - only used for coupled model design space
14. Editor - Doesn't work when modeler is exported as a JAR

Please see the Help files (in Atomic>Distribution and Coupled>Distribution) for a more detailed description of the button bar.

The following sections explore the functions of the GUI with examples.

Creation of DEVS Models (using the design space)

Models can be defined within the design space. First select the appropriate design space (Atomic or Coupled) for the model by clicking on the appropriate tab.

Create Atomic Model

The basic steps required for creating an atomic model are as follows:

- 1) Choose the atomic design space by clicking once on the Atomic tab in the interface.
(The atomic design space consists of a central panel and two lateral panels.)
- 2) Select *File > New* from the main menubar.
- 3) Position the mouse within the design space, and right-click. The following popup menu will be displayed.

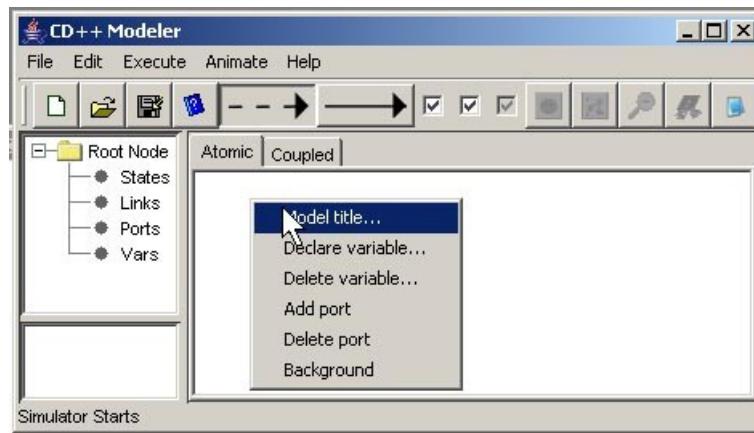


Figure 130: Creating an atomic model

Setting the Model Title

- 1) Left-click on *Model title*. The *Set Title* ("Model ClassName") dialog will be displayed.



Figure 131: Entering the model title

- 2) Type in the appropriate Model Title, and press Set.

Adding Units

- 1) Left double-click on the design space. A circular unit will be drawn in blue.

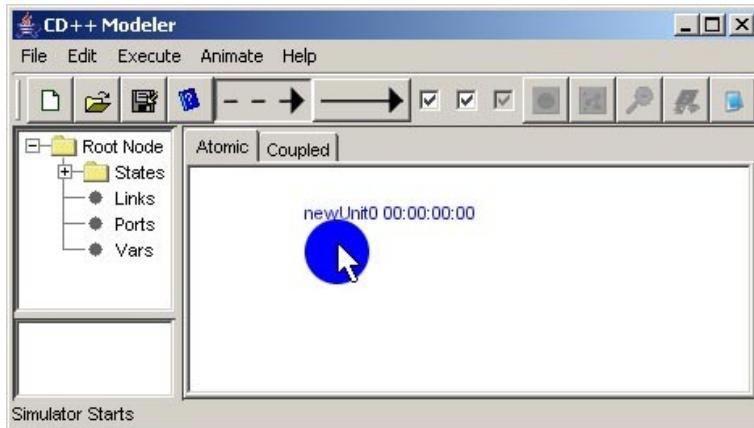


Figure 132: Adding Units to an Atomic Model

- 2) Left double-click on the circular unit. This will select the circular unit, as represented by the unit becoming red.

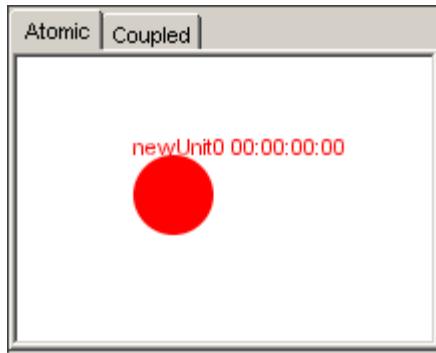


Figure 133: Selecting a unit

- 3) Left double-click on the selected unit. This will de-select the circular unit, as represented by the unit returning to blue.

Setting Properties of Units

- 4) Right-click on the circular unit. The following popup menu will be displayed.

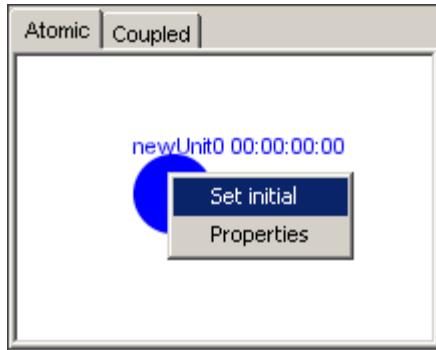


Figure 134: Setting initial property units

- 5) To set this unit as the initial unit of the model, left-click on *Set initial*.

When another unit is set as the initial unit of the model, the previous unit is automatically deselected as the initial unit.

- 6) To set the properties of this unit, left-click on *Properties*. The State Properties ("Atomic Unit properties") dialog will be displayed. Type in the appropriate state ID and state Time-to-Leave. Press *OK*.

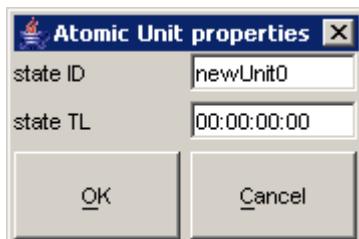


Figure 135: Entering State ID and state TL

Deleting Units

7) Select the circular unit (ie. left double-click on the circular unit). After the unit becomes red, press the Delete button on the keyboard, or select *Edit>Delete* from the main menubar.

When a unit is deleted, all links connected to the unit will also be deleted.

To delete multiple units, select the multiple units (ie. left double-click on each of the units), and press Delete or *Edit>Delete*.

Adding Ports

1) Right-click within the "canvas" (design space). The following popup menu will be displayed.

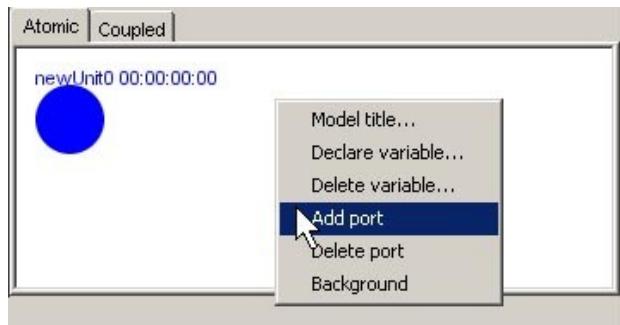


Figure 136: Adding ports

2) Left-click on *Add port*. The Add Port dialog will be displayed. Type in the appropriate PortID, select the direction of the port (In or Out), and select the port type (Integer or Float). Press *OK*.

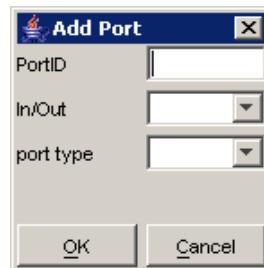


Figure 137: Entering Port ID

Repeat this procedure to add all the necessary ports for a unit.

Deleting Ports

3) Right-click within the "canvas" (design space). The following popup menu will be displayed.

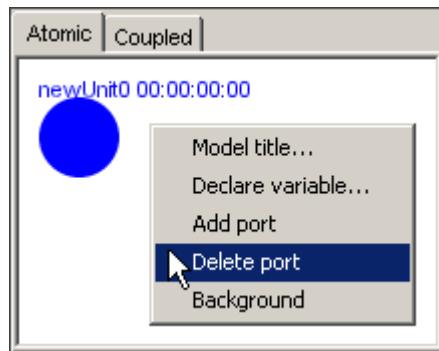


Figure 138: Deleting Ports

4) Left-click on Delete port. The Delete Port dialog will be displayed. Choose the appropriate port to be deleted from the Port ID dropdown menu. Press *Delete* to delete the selected port.



Figure 139: Delete Port Dialog box

Adding Links

Links represent transitions between different or similar units. After creating all the necessary atomic units, links can be made between the units or be created as a selflink.

Two types of links are available for atomic models:

Internal link (representing an internal transition): Transition happens automatically as a timer runs out.

External link (representing an external transition): Transition happens when a certain expression returns true.

Steps to creating a link:

1) Before drawing a link, the user should select the desired link type by clicking either the internal or external link button on the toolbar.

2) To draw a link between two units, position the mouse on one of the units, press and hold the left mouse button on one unit, drag the mouse to the other unit, and release the left mouse button. A link with the pre-selected type will be drawn.

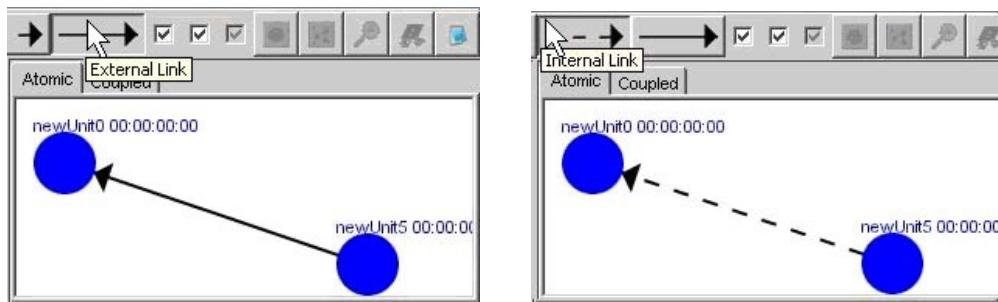


Figure 140: Adding links (external and internal)

3) To create a 'self link' (a link between an atomic unit and itself), position the mouse cursor on the unit. Make shure that the unit is not selected [unit is not coloured in red]. Press and hold the left mouse button. Drag the mouse away from and back to the unit with the left mouse button being held. Release the left mouse button to finish creating the self link.

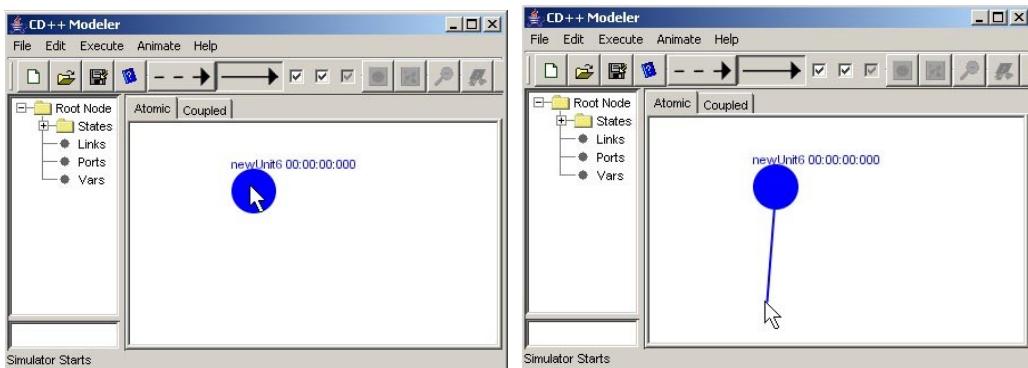


Figure 141: Creating a self link

A Warning Message dialog will be displayed, verifying the creation of a self link. Click OK.



Figure 142: Warning message for creating a self link

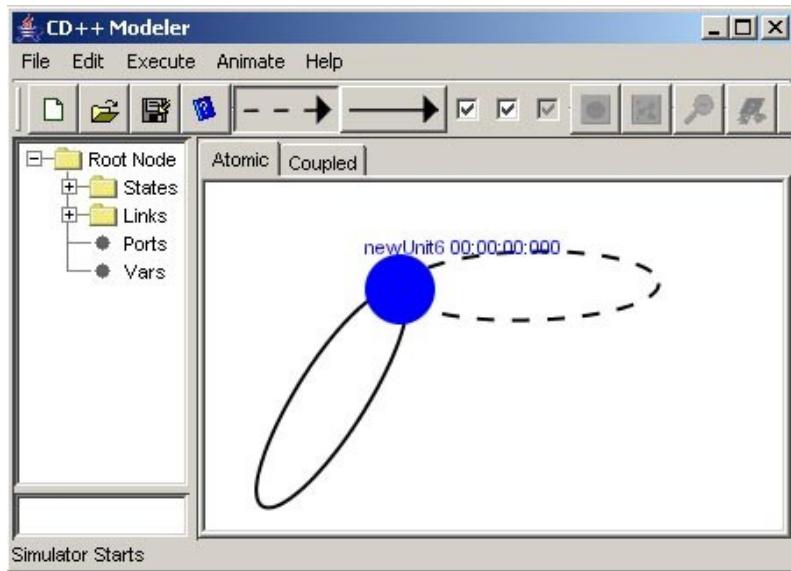


Figure 143: Example of internal and external selflinks attached to an atomic unit

Adding Transition Events to Internal and External Links

Internal Link Transition: To attach a port and value assignment to an internal link, right-click on the link. The following popup menu will be displayed.

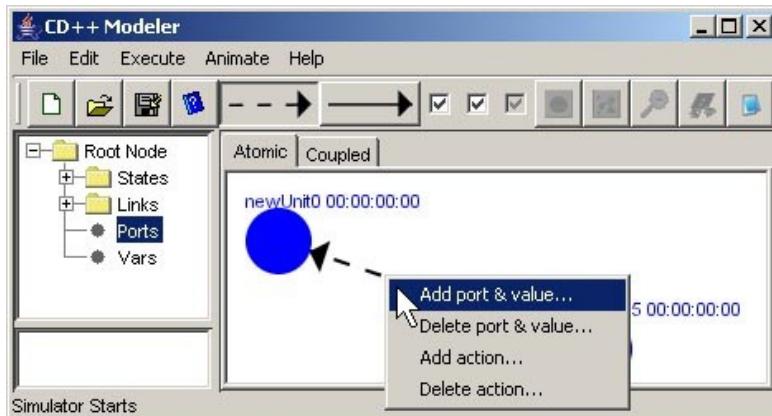


Figure 144: Attaching a link to a port

Left-click on *Add port & value*. The Add port & value dialog box will be displayed (See Figure 145). Select the appropriate port [output ports are for internal transition], and type in the appropriate value for the port. Press *OK*. When an internal transition occurs the selected output port will receive the assigned value which is set in the Add port & value dialog box.

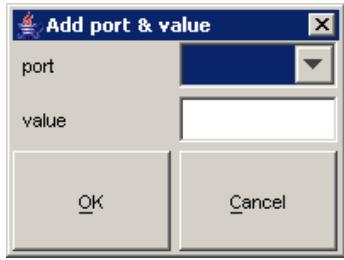


Figure 145: Adding port and value dialog box

External Link Transition: These transitions will allow transitions between different atomic units [states]. In general the transition happens when a user-defined value is received on a specific input port.

To add an expression to an external link [represented by a solid arrow] the following must be done:

- 1) Right click on the on the external link. And select *Edit Expression....*

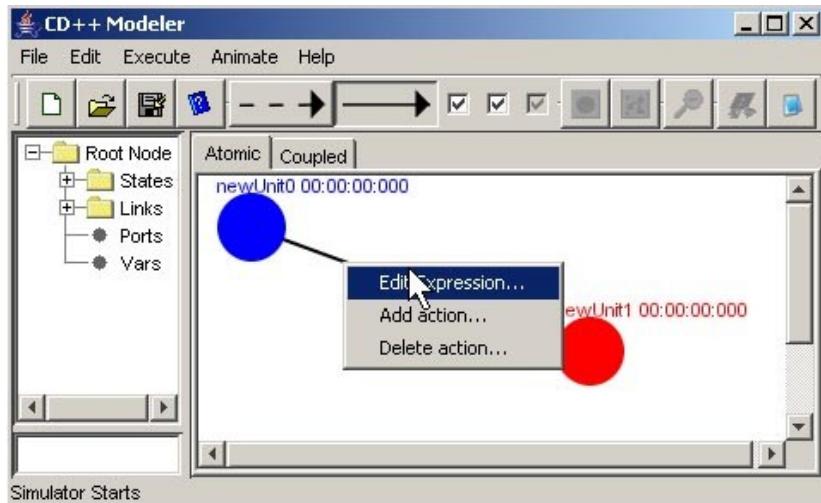


Figure 146: Adding an expression to an external link

- 2) Expressions can be either added manually [if you know what to type] or with the help of the *Add Function* button. Inorder to use the *Add Function* button first select a function to add from the bottom text field then click on the *Add Function* button. Now a dialog box will pop up asking for parameters required in the function. Fill in the parameters and click *ok*. Now that a function is added, user must MANUALLY configure the rest of the expression by typing "? value" [without the quotes] after the expression. The "value" represents any number. An example external transition expression would be: Value(in)?2 . This simply means that when the value at the port "in" [represented by the Value(port) function] is equal to "2" then make a transition to the other state.

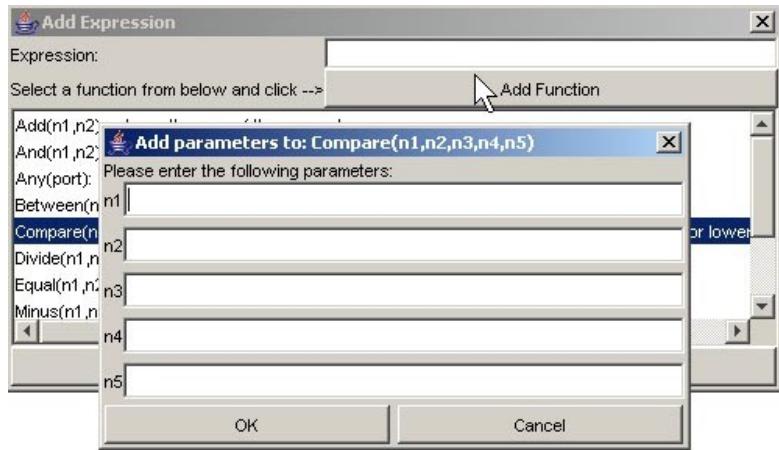


Figure 147: Showing a dialog window when the Add Function button is clicked.

Deleting Links

5) Select the link (ie. left double-click on the link). After the link becomes red, press the Delete button on the keyboard, or select *Edit>Delete* from the main menubar.

To delete multiple links, select the multiple links (ie. left double-click on each of the links), and press Delete or *Edit>Delete*.

Saving and exporting the model

1) To save the model to disk, select *File>Save* or *File>Save As* from the main menubar. The model will be saved as a .gam file.

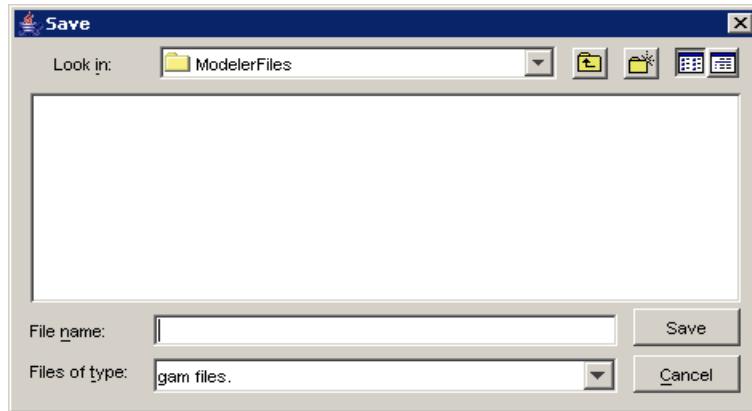


Figure 148: Save file Dialog box

2) In order to use the model in the other CD++ tools, the model must be exported to a standard .cdd file. To export the model, select *File>Export*. *.cdd must be the file name extension.

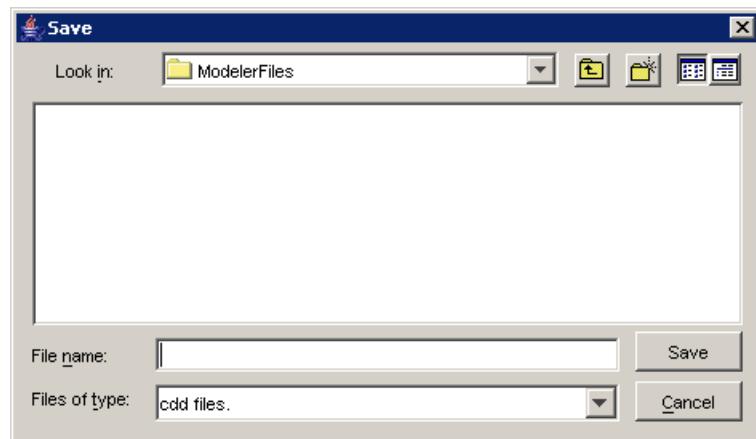


Figure 149: Choosing cdd files only

- 3) To save and export the model in one step, select File>Save and Export.

8.16.2.1 Create Coupled Model

The basic steps required for creating a coupled model are similar to those for creating an atomic model.

Open CD++ Modeler and choose the coupled design space by clicking once on the Coupled tab in the interface.

The coupled design space consists of a central panel and three lateral panels:

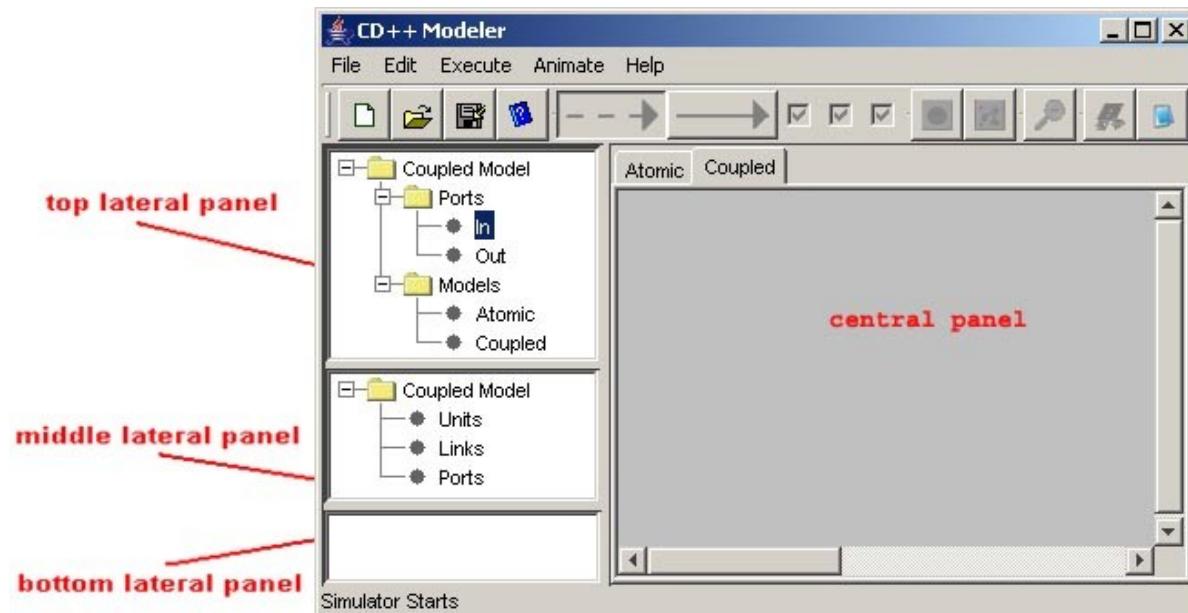


Figure 150: The coupled design space

The central panel displays the objects (in graphical form) of the coupled model. Objects (such as ports, atomic models, and coupled models) can be added, and links can be connected between the objects.

The top lateral panel displays the types of objects that are available for defining the model. It is divided into ports (input and output), predefined atomic models, and predefined coupled models.

The middle lateral panel displays the objects (in textual form) that define the model. (The textual form of the objects in the middle lateral panel correspond to the graphical form of the objects in the central panel.)

When an object/element is selected in the middle lateral panel, the description/details of the element will be displayed in the bottom lateral panel. (ie. The bottom lateral panel displays a description of the element currently selected in the middle lateral panel.)

The central panel is referred to as the 'Modeling Panel'.

The middle lateral panel is referred to as the 'Units Panel/Tree'.

Adding Atomic Units

1) To add a new atomic model that is not yet defined, click once on the *Add new Atomic Model Unit* button. The new atomic model unit will be drawn in the upper-left corner of the Modeling Panel, and will be given a default name (eg. newAtomicModel0).

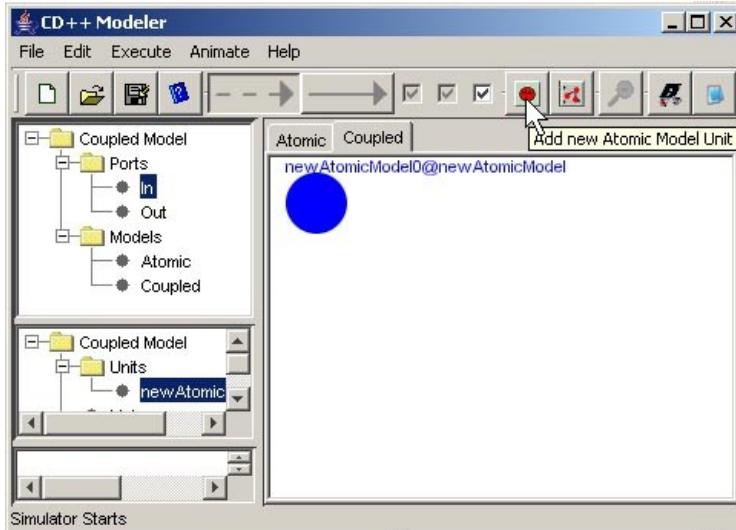


Figure 151: Adding atomic units

2) To edit the atomic model unit, right-click on the unit. The following popup menu will be displayed.

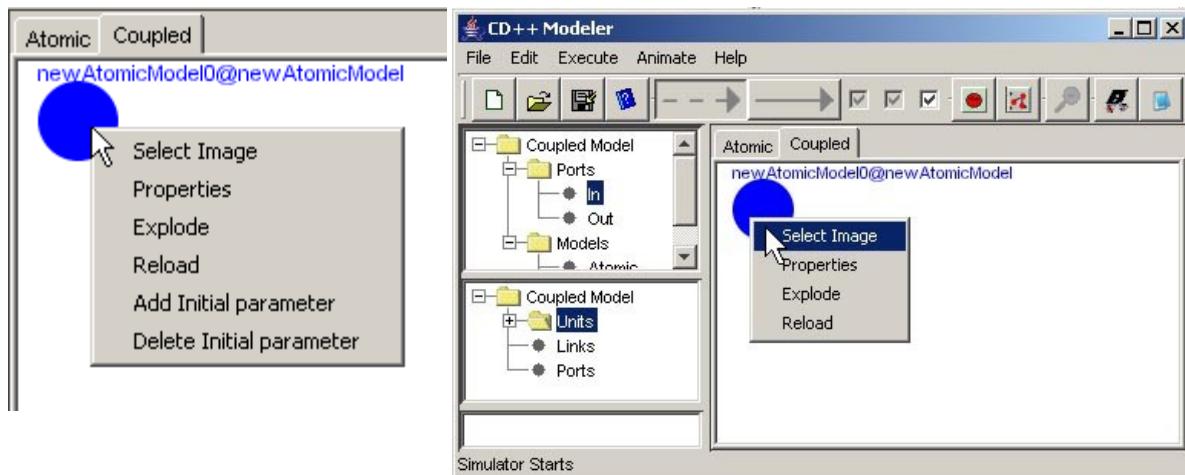


Figure 152: Right clicking on the image to edit

Select Image - change the icon for the unit

Properties - change the unit name

Explode - explode the unit for its definition

Reload - reload the unit if its definition has been modified

3) To change the unit name, left-click on Properties. The Model Properties dialog will be displayed. Type in the appropriate Unit ID. Press OK.

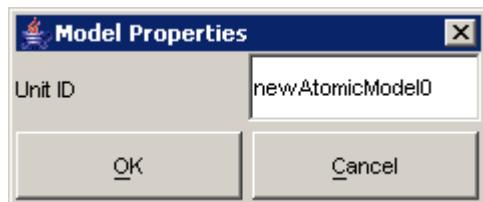


Figure 153: Changing unit name

The Explode and Reload options will be described in later sections.

Adding Coupled Models

- 1) To add a new coupled model that is not yet defined, click once on the Add new Coupled Model Unit button. The new coupled model unit will be drawn in the upper-left corner of the Modelling Panel, and will be given a default name (eg. newCoupledModel2).

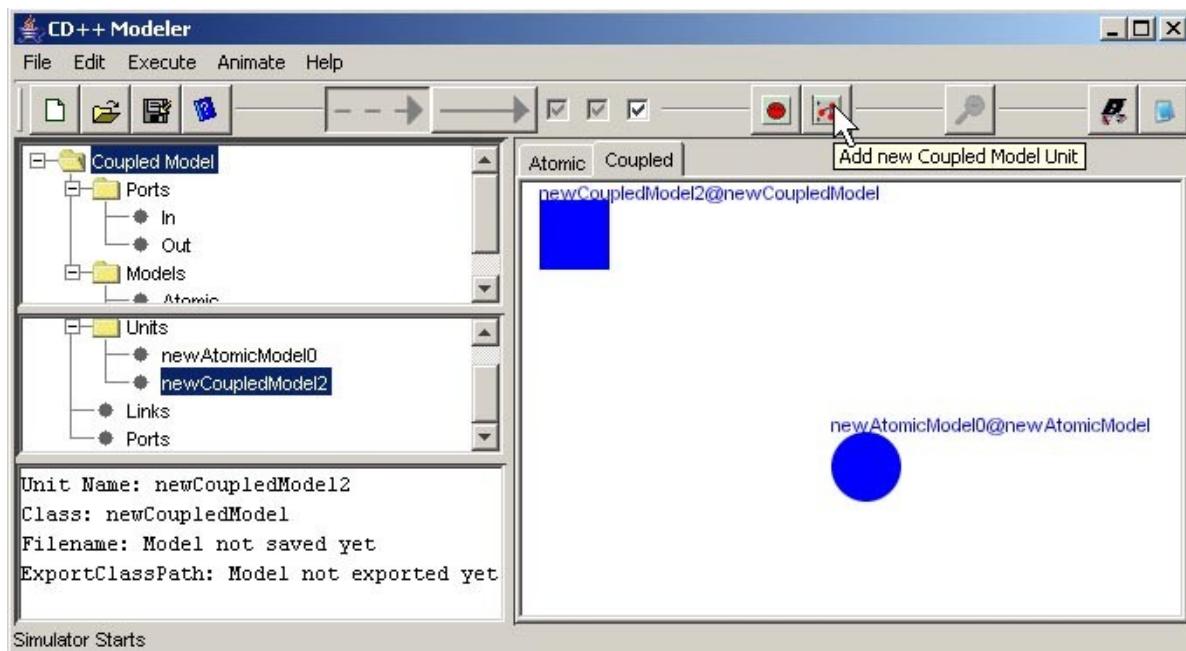


Figure 154: Adding coupled models

- 2) To edit the coupled model unit, right-click on the unit. The following popup menu will be displayed.

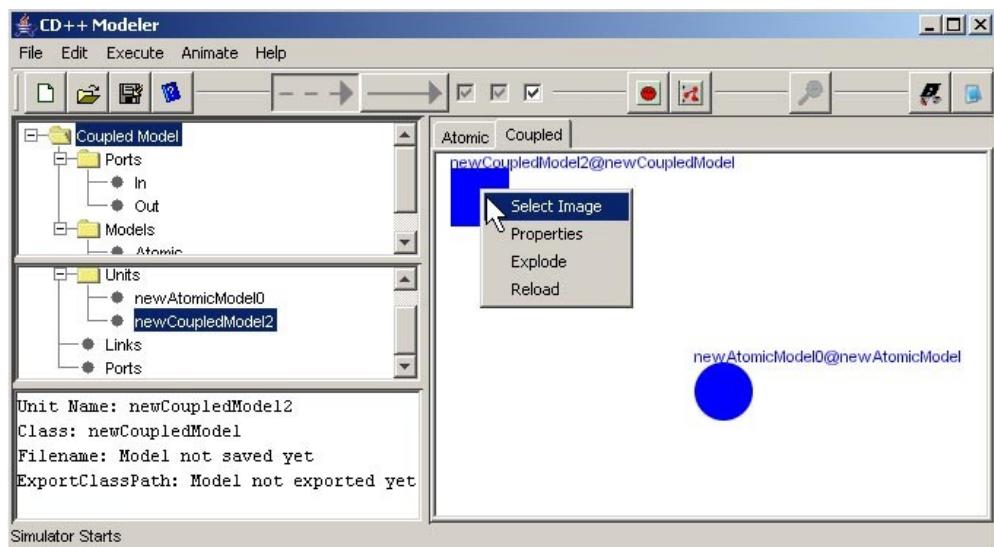


Figure 155: Editing a coupled model

- Select Image - change the icon for the unit
- Properties - change the unit name
- Explode - explode the unit for its definition
- Reload - reload the unit if its definition has been modified

3) To change the unit name, left-click on *Properties*. The Model Properties dialog will be displayed. Type in the appropriate Unit ID. Press *OK*.

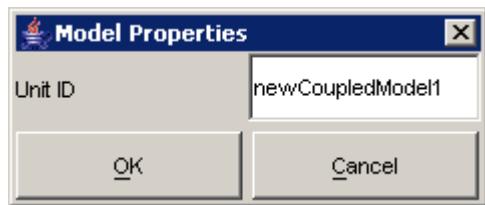


Figure 156: Changing the unit name

The Explode and Reload options will be described in later sections

Adding Ports

Two types of ports are available for coupled models: Input and Output.

- 1) To add a port, first select the type of port in the top lateral panel. Within the canvas (design space), position the mouse cursor at the desired location of the port. Left double-click. The selected port type will be drawn, and will be given a default name. For example, Port0 represents an input port, while Port1 represents an output port.

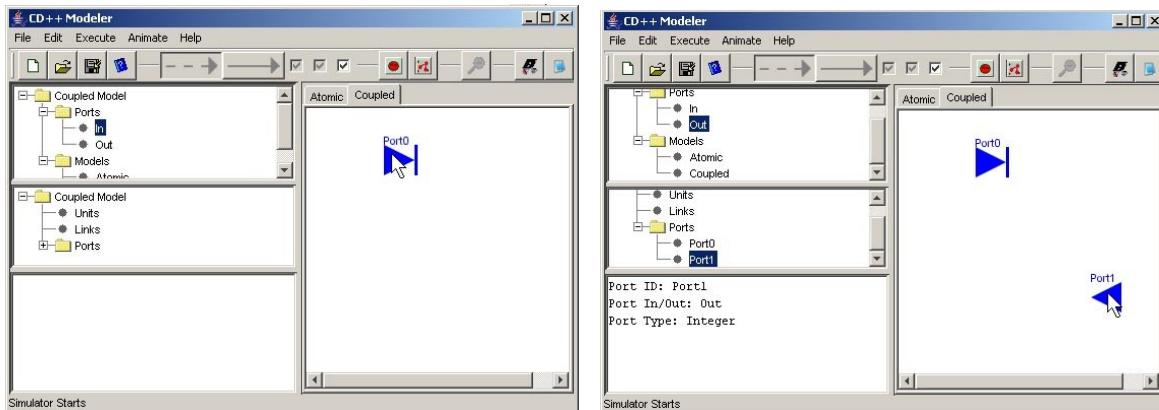


Figure 157: Adding ports

The name of the port can be changed in two ways:

- 2a) Right-click on the appropriate port. The following popup menu will be displayed. Left-click on *Properties*.

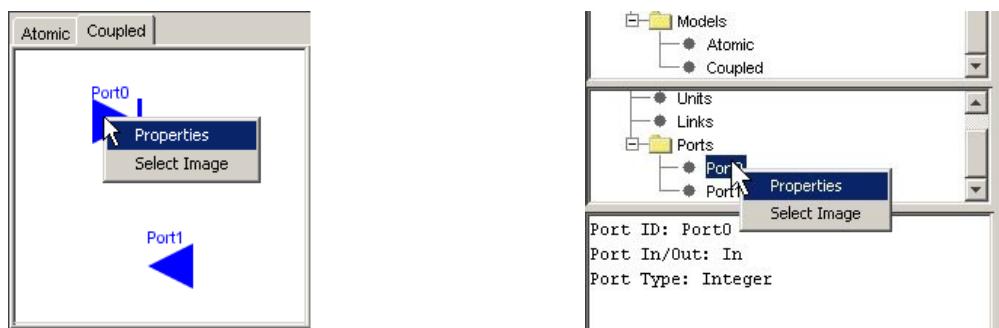


Figure 158: Changing the name of the port

- 2b) In the Units Tree (the middle lateral panel), expand the Ports folder. Left double-click on the appropriate port. An alternative method is to right click on the appropriate port and select properties as seen in Figure 158.

- 3) The Port Properties dialog will be displayed. Type in the appropriate PortID. Press OK.

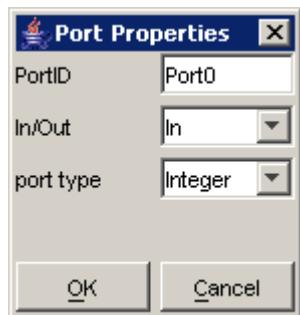


Figure 159: *Entering port ID*

Adding Links

To add a link between two objects, position the mouse cursor on the first object (the origin of the link). Press and hold the left mouse button. Drag the mouse to the second object (the destination of the link). Release the left mouse button. The link will be drawn from the origin to the destination. For example, the origin of the link is Port 2, while the destination of the link is newAtomicModel0.

The link will be created only if both the origin and destination are valid.

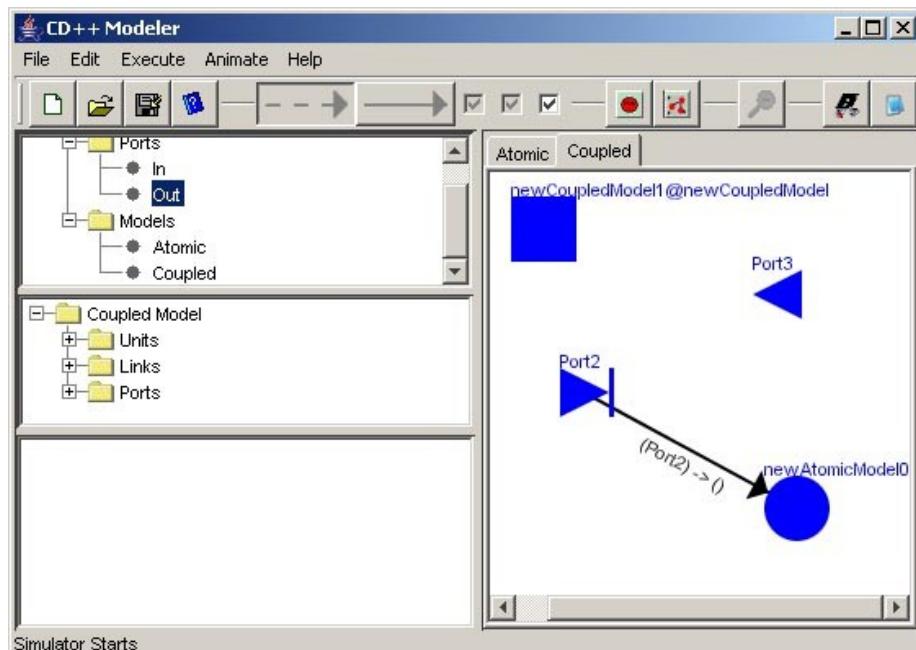


Figure 160: Adding links

NOTE: If you uncheck the *Show link port* checkbox then *(Port2)->()* will not be displayed.

Selflinks can also be created for coupled models. One can add selflinks to the coupled or atomic model unit as seen in Figure 161. Selflinks are added in the same way as one would add a selflink to an atomic model (See section).

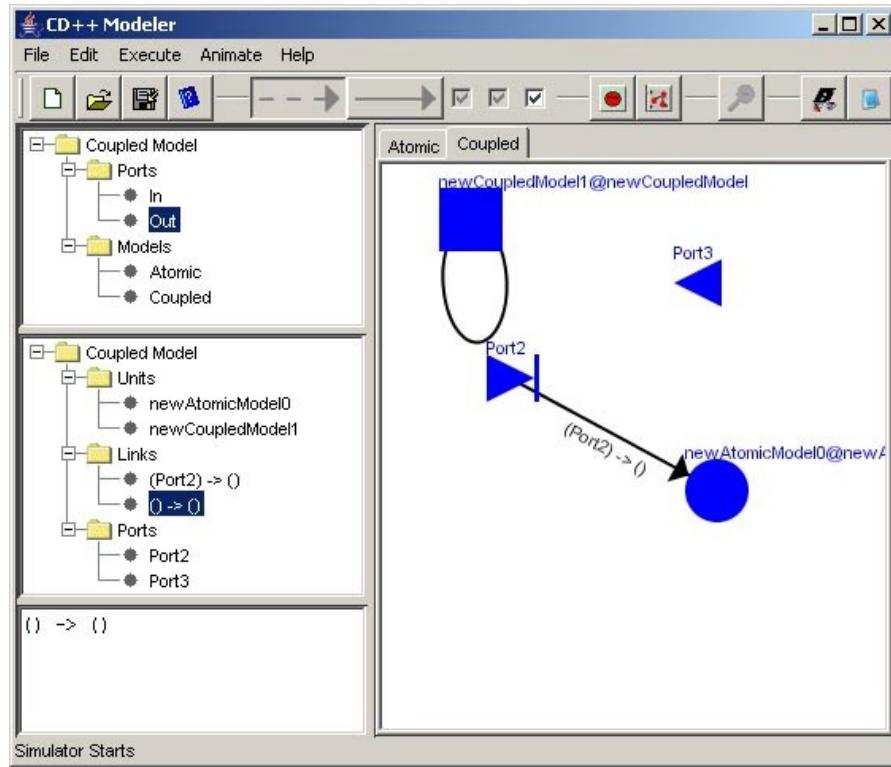


Figure 161: Example selflink starts and ends at newCoupledModel1

8.16.2.2.1 Importing Models

A coupled model is composed of other atomic and coupled models, which can be imported as predefined models. Three model types are valid for importing:

- predefined coupled models (.ma)
- predefined atomic models (.cdd)
- basic atomic models (from register.cpp)

For example, let's consider the ATM model.

You can download the necessary files for the ATM model here: <http://chat.carleton.ca/~jcao/blog/cd+files/ATM%20MODEL%20USING%20NEW%20CD%20MODELER.zip>

Continue reading once you have extracted the zip files.

Open CDModeler, click on the coupled tab, click *File>New* to start the creation of a new coupled model.

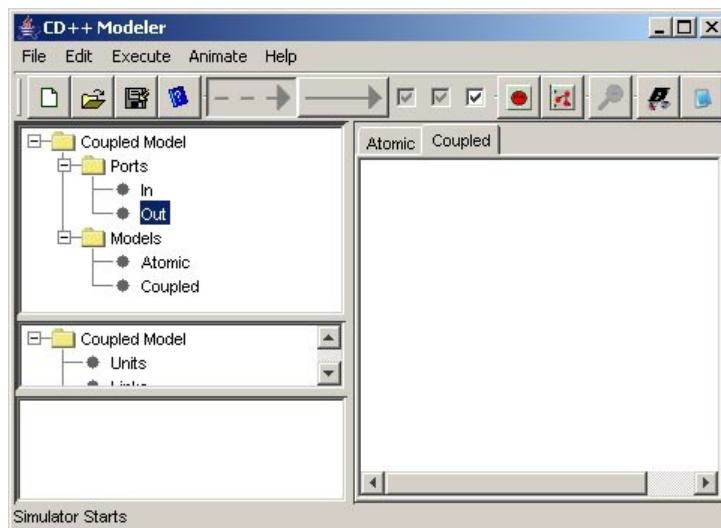


Figure 162: Fresh new coupled design space.

A predefined model can be imported by selecting *File>Import*. The Import dialog will be displayed. (Only .ma, .cdd, and .cpp files can be imported.)



Figure 163: Import dialog box

Browse to the extracted folder containing ATM files, select *register.cpp*, and press *Import*. Now, a dialog box will popup as a reminder that one must create ports to the model(s) before adding them to the design space. Click *OK*.

In the top lateral panel browse to *Models>Atomic*. The CDmodeler window should now look like the following:

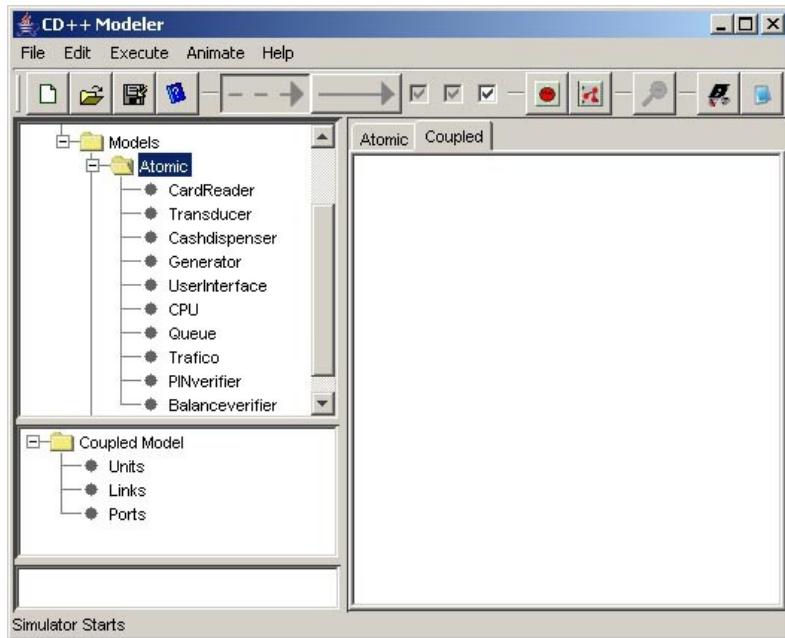


Figure 164: Showing imported models.

The name of the imported models will be displayed in the top lateral panel, within the appropriate folder (Atomic or Coupled). In this example the *Register.cpp* corresponds to a series of atomic models and thus expanding the atomic subfolder one can find all the models associated with *Register.cpp*.

For example: When a coupled model definition file (.ma) is imported, the name of the imported model is displayed in the top lateral panel under *Models>Coupled*. When an atomic model definition file (.cdd) is imported, the name of the imported model is displayed in the top lateral panel under

Models>Atomic.

To add a unit of the imported model to the definition of the current coupled model, first select the imported model name (PINVerifier) in the top lateral panel. Within the canvas (design space), at the desired location of the unit, left double-click. A unit of the imported model will be drawn.

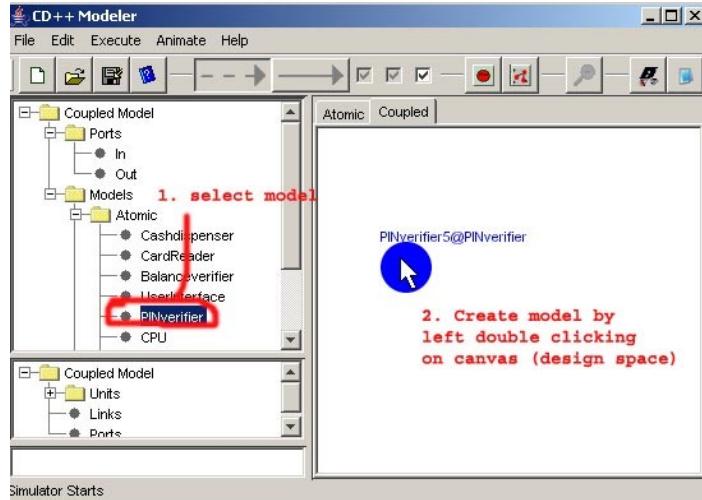


Figure 165: Creating a model on the canvas (design space)

*NOTE: Units of imported models **cannot** be modified.*

Importing an *.ma file:

Continuing from the previous example, click *File>Import*. Browse to the ATM folder again but this time select *atmNEW.ma* and click on *import*.

A coupled model definition file (.ma) named "ATM" is imported:

- (1) The name "ATM" is displayed in the top lateral panel under Models>Coupled.
- (2) "ATM9" (a unit of "ATM") is added to the definition of the current coupled model [This is done by selecting ATM in the top lateral panel and double clicking on the canvas (design space)].
- (3) The predefined ports of "ATM9" are displayed in the middle lateral panel. (For "ATM", there are two output ports: 'cash_out' and 'card_out'. There is also one input port: 'in').

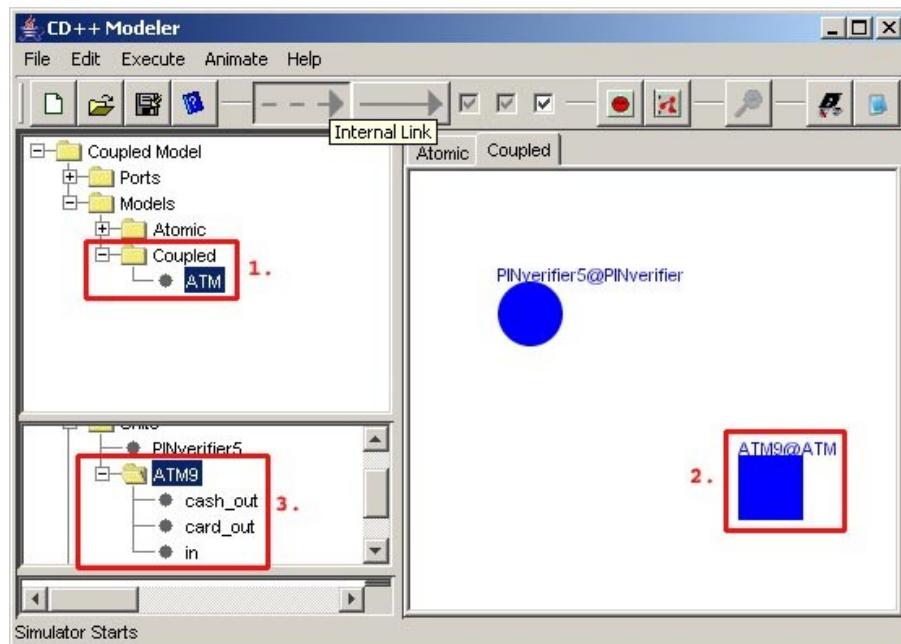


Figure 166: Importing and creating a coupled model.

NOTE: Importing a *.cdd file undergoes the same process as importing a *.cpp file the only difference is that one will need to select a *.cdd file and the model imported will still be found in the *Atomic* subfolder.

Adding ports to imported atomic models:

Ports can be added to any unit of the imported atomic models.

To do so, follow the steps:

1. Open CDModeler
2. Click on the *Coupled* tab
3. Click on *File>New*
4. Import Register.cpp
5. Expand the Atomic Subfolder by clicking on the '+' beside the Atomic folder
6. Left click to select the model one wishes to add ports to. In this case select "Transducer".
7. Right click on "Transducer" and select *Add port* from the popup menu to add a port to "Transducer". (See Figure 167)

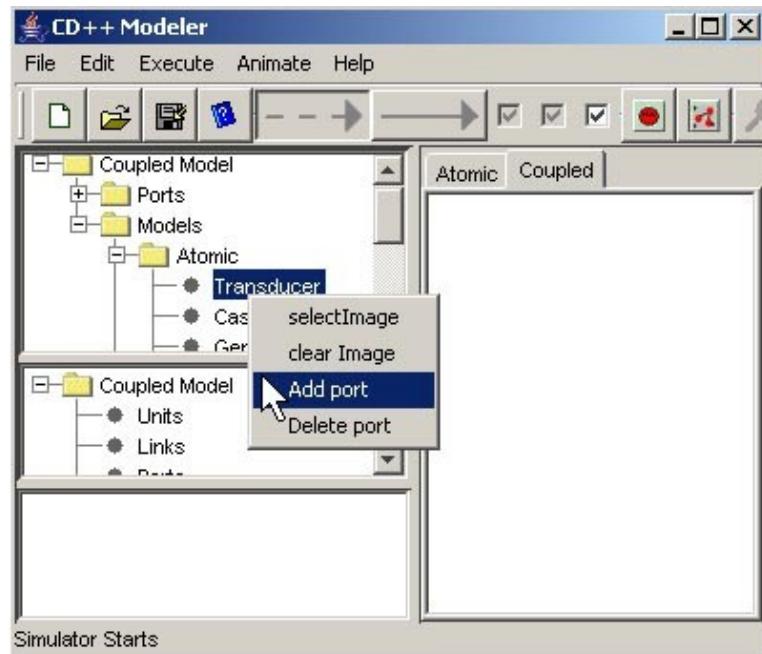


Figure 167: Adding a port to imported model.

8.16.2.2 Exploding Models

A coupled model (henceforth referred to as "current coupled model") is composed of other atomic and coupled models (henceforth referred to as "component models"), which (as demonstrated in previous sections) can be added as blank models from the button bar or imported as predefined models.

These component models (within the current coupled model) can be inspected, or "exploded", for the purposes of definition and modification.

- 1) To explode a component model, right-click on the model. The following popup menu will be displayed. Left-click on Explode. A new model editor (henceforth referred to as "exploded editor") will be displayed, in which the exploded model can be edited. (The original model editor will be temporarily hidden, and will become visible again after the exploded editor has been closed.)

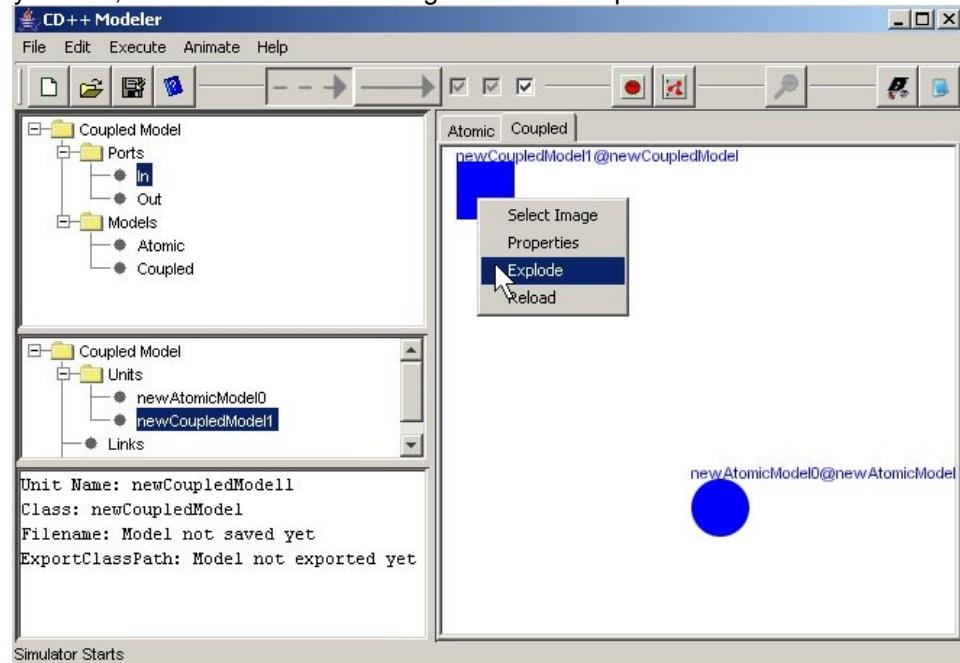


Figure 168: Explode model menu

If an atomic model is exploded, an atomic model editor will be displayed. If a coupled model is exploded, a coupled model editor will be displayed.

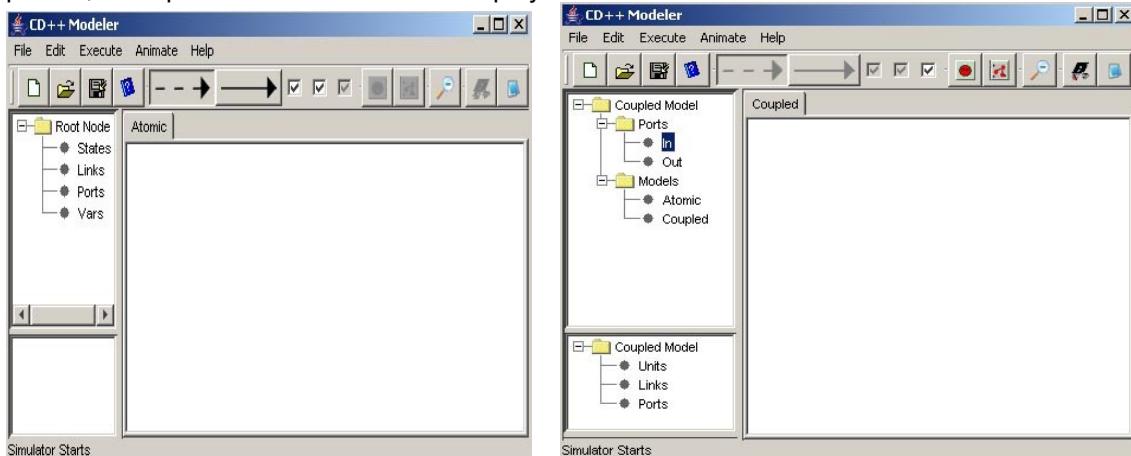


Figure 169: Atomic and Coupled editor corresponding to exploded model

Note that within the exploded editor, the exploded model type (ie. atomic or coupled) cannot be changed.

2) The exploded model can be edited using the same procedures as for non-exploded models. For example, consider the following exploded coupled model, which was created for newCoupledModel1 (seen in step 1).

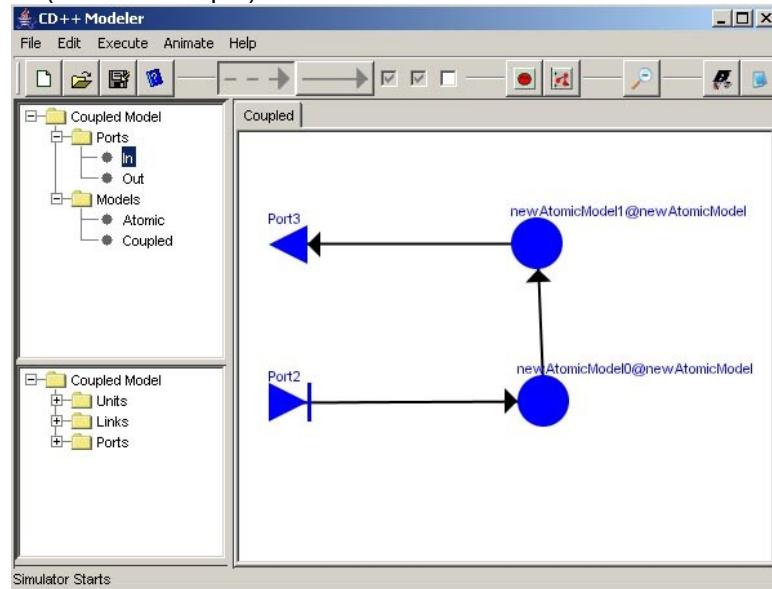


Figure 170: Editing exploded models

3) When the definition/modification of the exploded model is complete one must close the exploded model and choose to save/cancel the changes made in order to return to the original model editor.

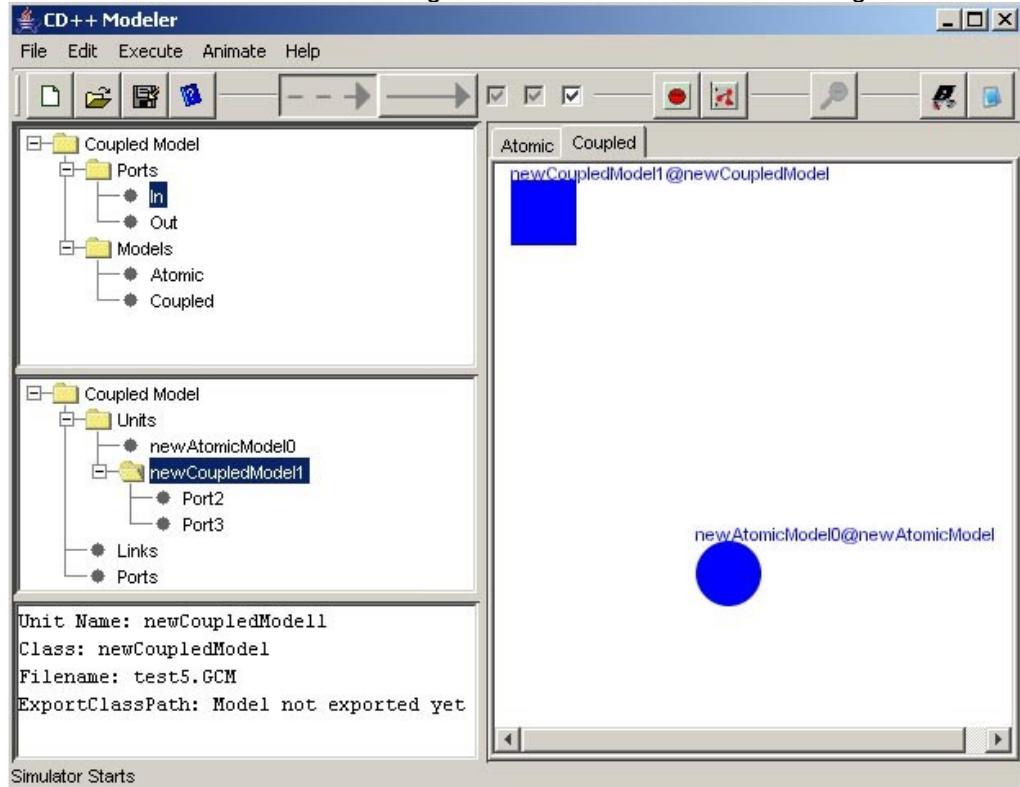


Figure 171: Original Model Editor

The ports from the exploded model (shown in step 2) are now available in the original model editor (after saving). For example, the ports for newCoupledModel1 (ie. input Port2 and output Port3) are

displayed in the middle lateral panel, under Coupled Model>Units>newCoupledModel1 (See Figure 171).

4) Once all remaining component models are defined, links can be added between the ports of the component models.

For example, once newAtomicModel0 has been exploded and defined/modified, the ports for newAtomicModel0 (ie. input portA and PortB, and output PortC) will be displayed in the middle lateral panel, under CoupledModel>Units>newAtomicModel0.

Additional ports (ie. input Port3 and output Port4) can be added to the current coupled model, and will be displayed in the middle lateral panel, under CoupledModel>Ports.

Links can be added between the ports of the current coupled model and the ports of the component models. These links will be displayed in the middle lateral panel, under CoupledModel>Links.

In the current coupled model seen below, there are three links:

- (1) output PortC of newAtomicModel0 is connected to input Port2 of newCoupledModel1
- (2) input Port3 of the current coupled model is connected to input Port2 of newCoupledModel1
- (3) output Port3 of newCoupledModel1 is connected to output Port4 of the current coupled model

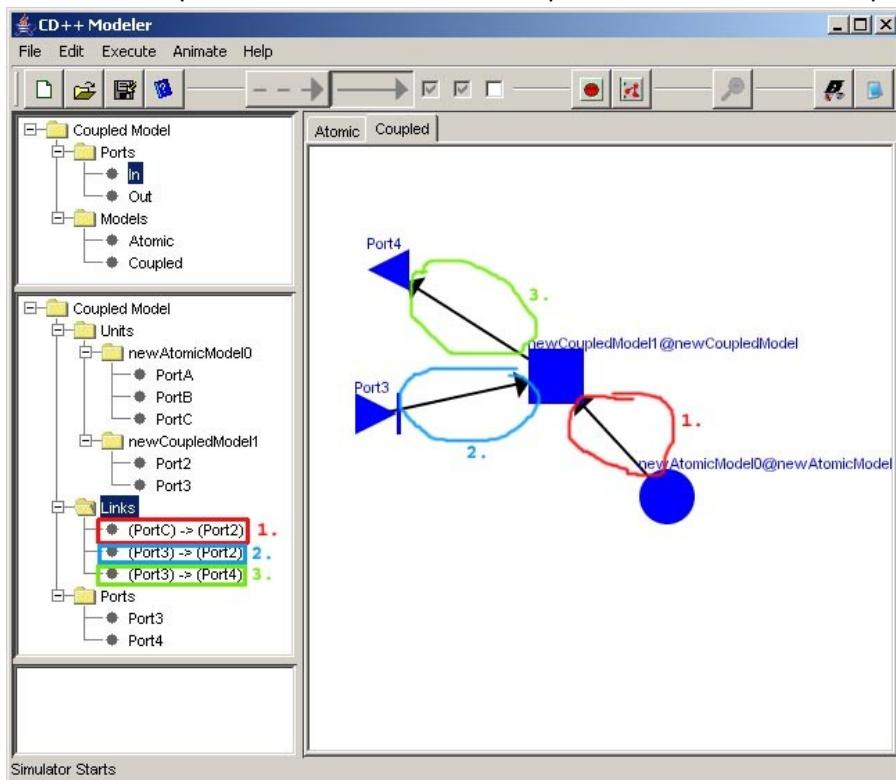


Figure 172: Coupled model with links

5) If a component model is exploded for modification after links have been connected, the links connected to the ports of the component model will be disengaged. This will prevent unpredictable changes from occurring.

For example, if newCoupledModel1 is exploded, the links connected to the ports of newCoupledModel1 (ie. input Port2 and output Port3) will be disengaged.

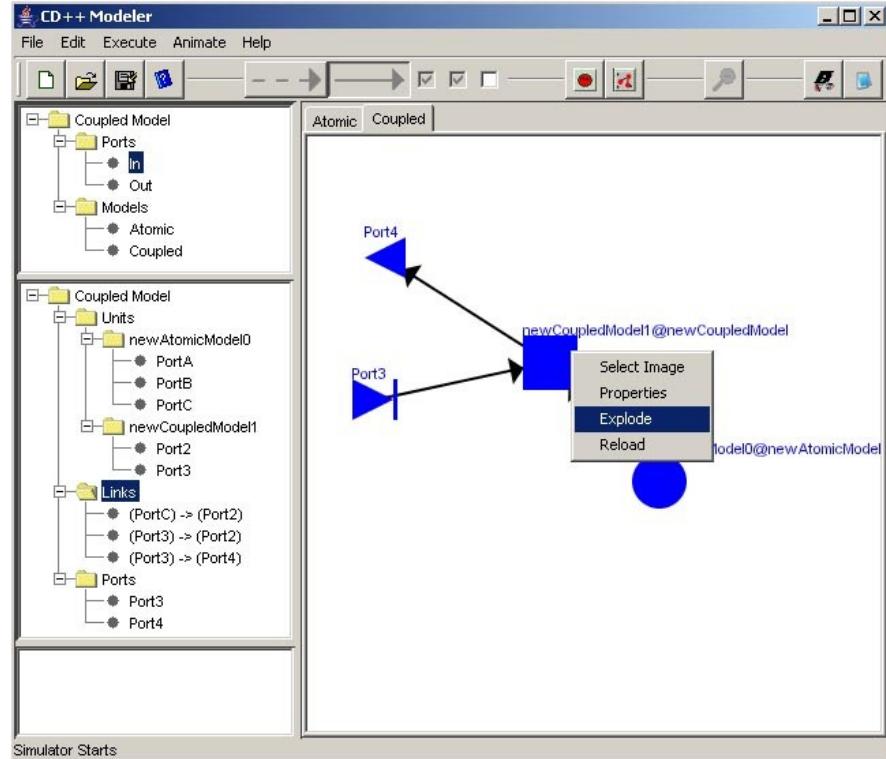


Figure 173: Disengaged links on an exploded coupled model

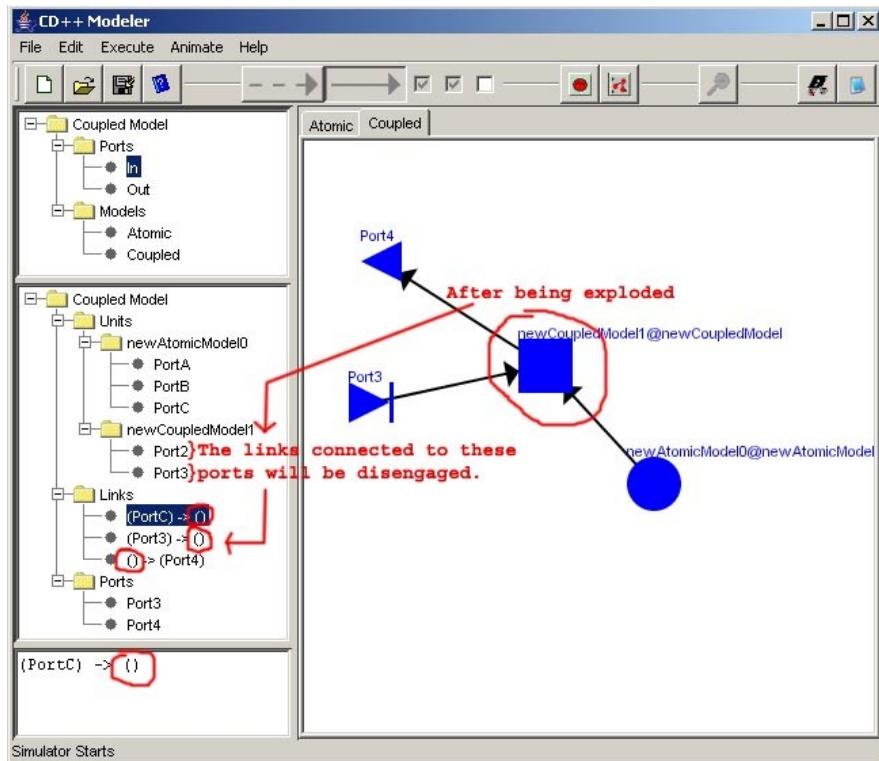


Figure 174: Detailed visualization of disengaged links

Adding Images to Models

Images can be added to a model in two ways. One way is to set an image as a background, the second way is to set an image to represent a unit within the model.

Adding Background

To add a background to the canvas; right click on the canvas [white space] and select *background* (see Figure 175). When the file browser window pops up select an image file to load and click *Set*.

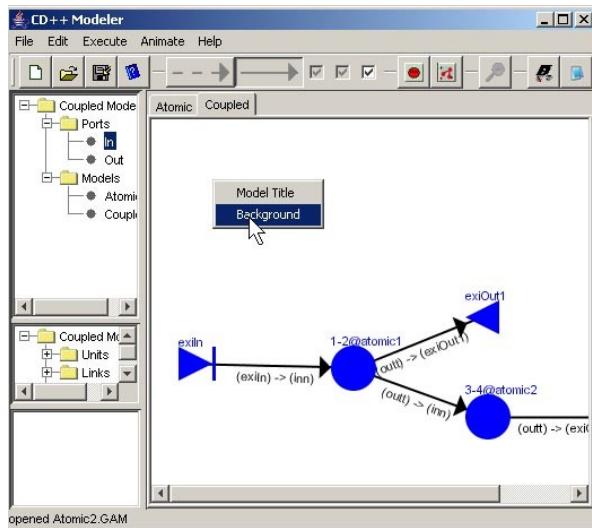


Figure 175: Loading Background

Note: The menu you get when right clicking on the canvas may be different than the one shown in Figure 175. This is because you are adding the background to the atomic canvas and not the coupled canvas. However, the steps to add the image is all the same. [Hint: click on *Background*]

After an image has been set as background you should see the background image stretched to cover the whole canvas. (See Figure 176)

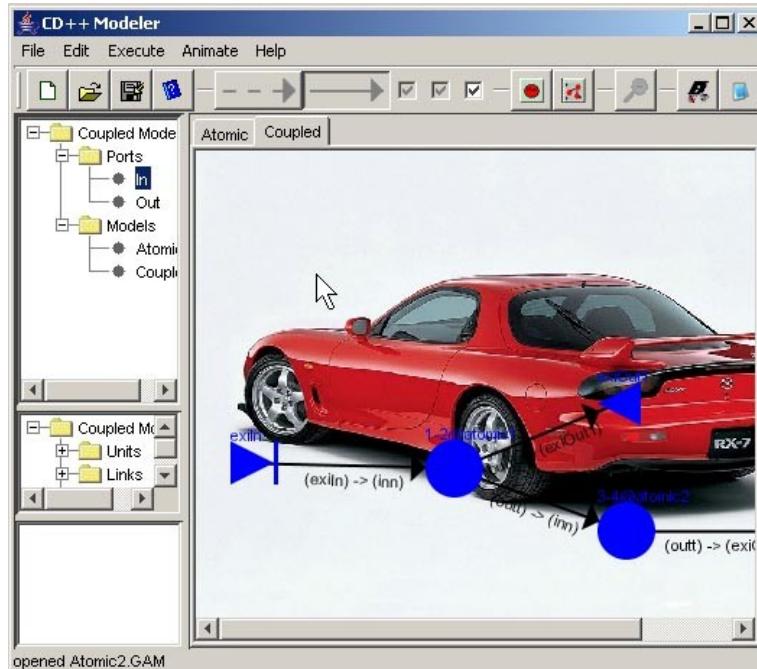


Figure 176: Image of a car set as background

Adding Image to Units on Coupled Canvas

Note: Images can only be added to units [any object that is not a link] on the coupled canvas.

Images can be added to atomic units, coupled units, input ports and output ports.

To add an image right click on the unit and click *Select Image* (see *Figure 177*).

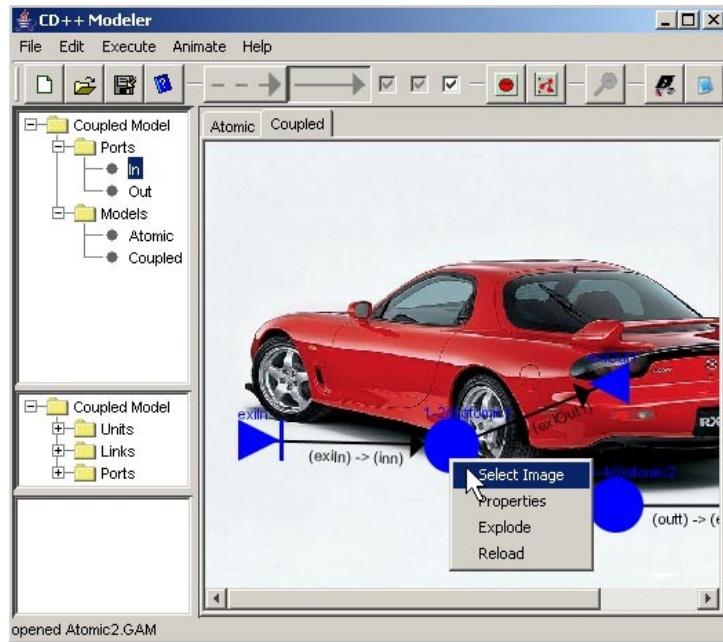


Figure 177: Adding image to unit

In the the file browser window select an image you wish to set as the image for the unit and click *Set*.



Figure 178: Choosing an image

The following is an example of having an image representing a unit [note the yellow car].

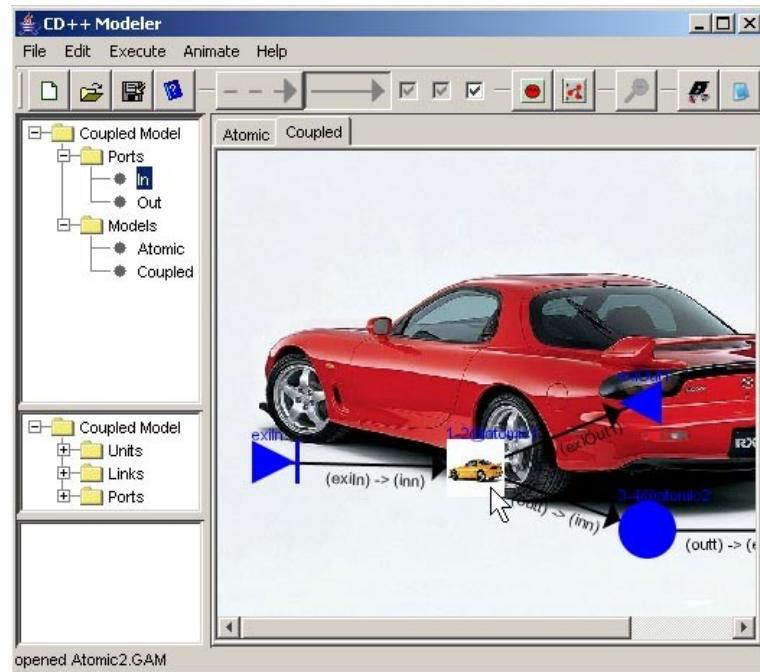


Figure 179: Image added to atomic unit

8.16.3 Visualization of DEVS models (using the Animate menu commands)

This section describes how to visualize the result files of atomic Cell-DEVS models, atomic-DEVS models, and coupled-DEVS models.

Model type selection

From the main menubar, left-click on Animate. The following menu will be displayed. Select the appropriate option, depending on the model type to be visualized:

- | | |
|---------------------|---------------------------|
| Cell-DEVS animation | - atomic Cell-DEVS models |
| AtomicAnimate | - atomic-DEVS models |
| CoupledAnimate | - coupled-DEVS models |

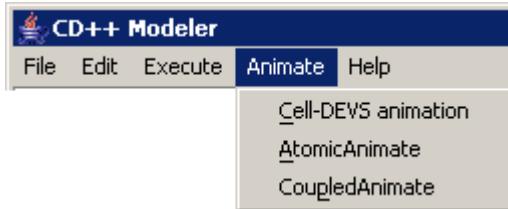


Figure 180: Animate menu

The following sections describe the commands of the Animate menu for the version of CD++ Modeler included in the CD++ Builder plugin for Eclipse, not the stand-alone version of CD++ Modeler.

AtomicAnimate using: Atomic-DEVS, Coupled-DEVS, Atomic Cell-DEVS

A DEVS model must include at least one atomic model. After simulating the DEVS model, a .log file is generated. The .log file records all the messages sent between DEVS components. This includes all messages sent/received by all atomic models in the coupled model. The message values sent/received by a specific atomic model can be extracted from the .log file and visualized.

The steps required for visualizing the message values sent/received by a specific atomic model are as follows:

- 1) In the Animate menu, left-click on AtomicAnimate. An atomic animate dialog box will be displayed.



Figure 181: Atomic Aniamte Options

- 2) Left-click on the browse button. A Set dialog will be displayed. Select the .log file that contains the messages sent/received by the atomic-DEVS model to be visualized. (The atomic-DEVS models that send/receive messages within the selected .log file will be available for visualization.) After choosing the appropriate .log file, left-click on Set.

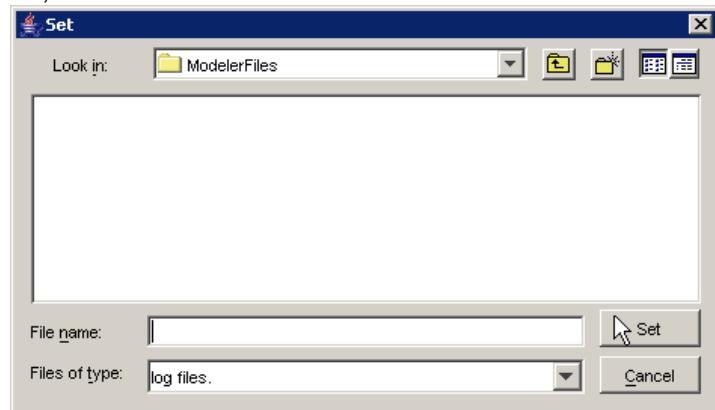


Figure 182: Set dialog box

If an invalid file type is chosen, the (blank) visualization window will be displayed as follows:

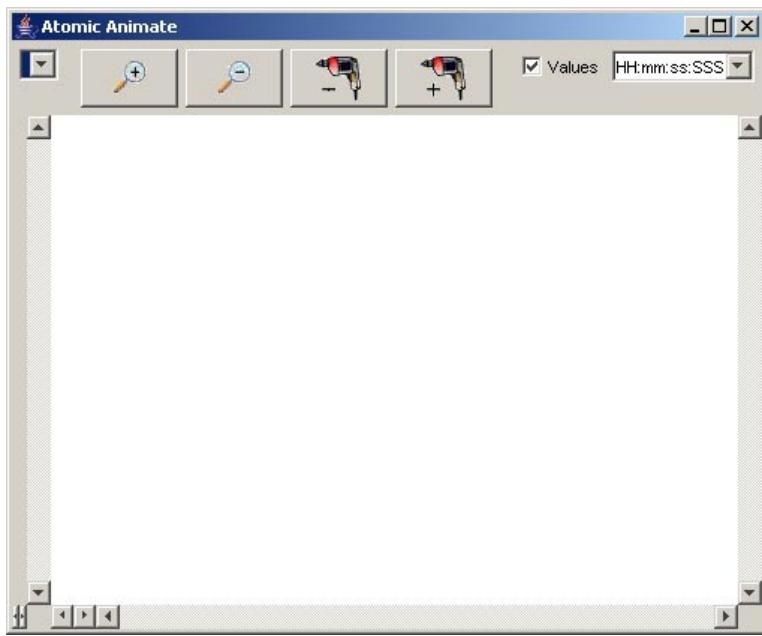


Figure 183: Empty visualization window due to invalid file type

The following example demonstrates the functionality of the Atomic Animate visualization window, using the atomic-DEVS model *FunctionEval* (which was extracted from the *Hybrid* model). Additional examples for a coupled-DEVS model (*4BitCounter*) follows.

AtomicAnimate Example of Atomic-DEVS: FunctionEval

To run this example, download the FunctionEval project. [Where would one download this?](#) FunctionEval is an atomic-DEVS model that contains one output signal.

In the Set and atomic animate dialogs, open tester_caso1.log.

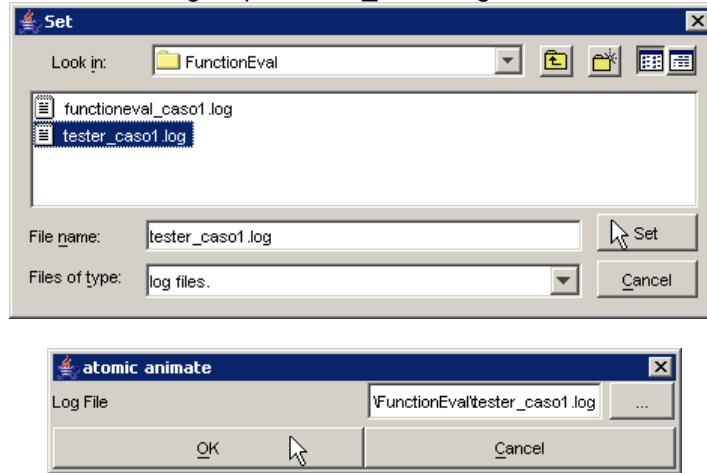


Figure 184: Opening tester_caso1.log

The atomic-DEVS models that send/receive messages within tester_caso1.log will be available for visualization. Since tester_caso1.log only contains the messages sent/received by the function@FunctionEvaluator model, the only atomic-DEVS model available for this example is the function model.

The following Atomic Animate visualization window will be displayed.

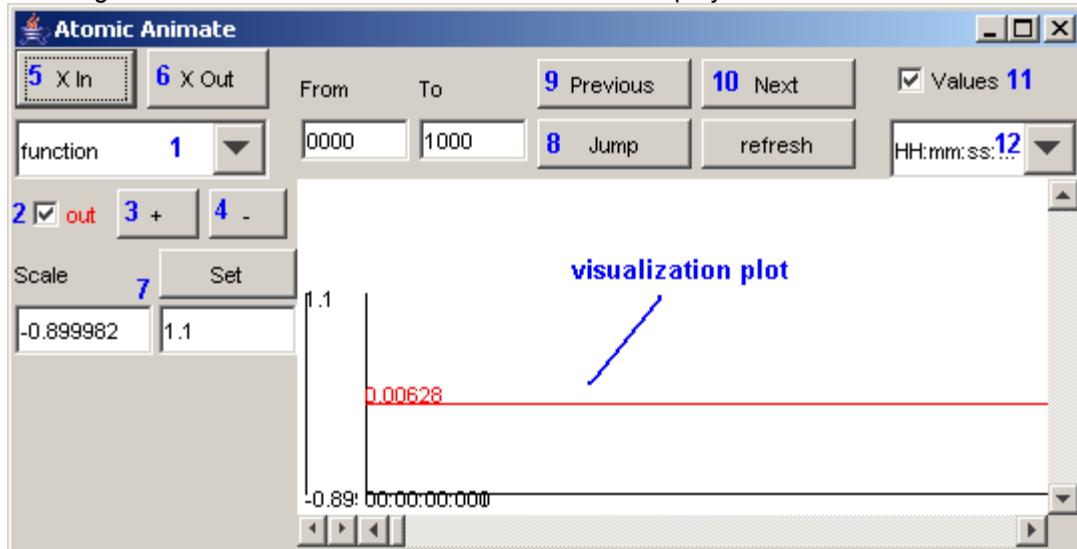


Figure 185: Atomic Animate visualization window

The Atomic Animate visualization window has several functions for modifying the appearance of the visualization, such as:

- select the atomic-DEVS model to be visualized (1)
- select the output signals (of the selected atomic-DEVS model) to be plotted (2)
- zoom in/out on the vertical axis (3, 4) and the horizontal axis (5, 6) of the plot

- set the upper and lower bounds of the vertical scale **(7)**
- jump to the specified time interval **(8)**
- go to the previous/next time interval **(9, 10)**
- show the values of the output signal on the plot **(11)**
- select the time format to be displayed on horizontal axis **(12)**

Select the atomic-DEVS model to be visualized

- 1) Left-click on the drop-down list. From the drop-down list, select or left-click the atomic-DEVS model to be visualized. The output signals of the selected atomic-DEVS model will be displayed. In this example, the only atomic-DEVS model available for visualization is the function model.

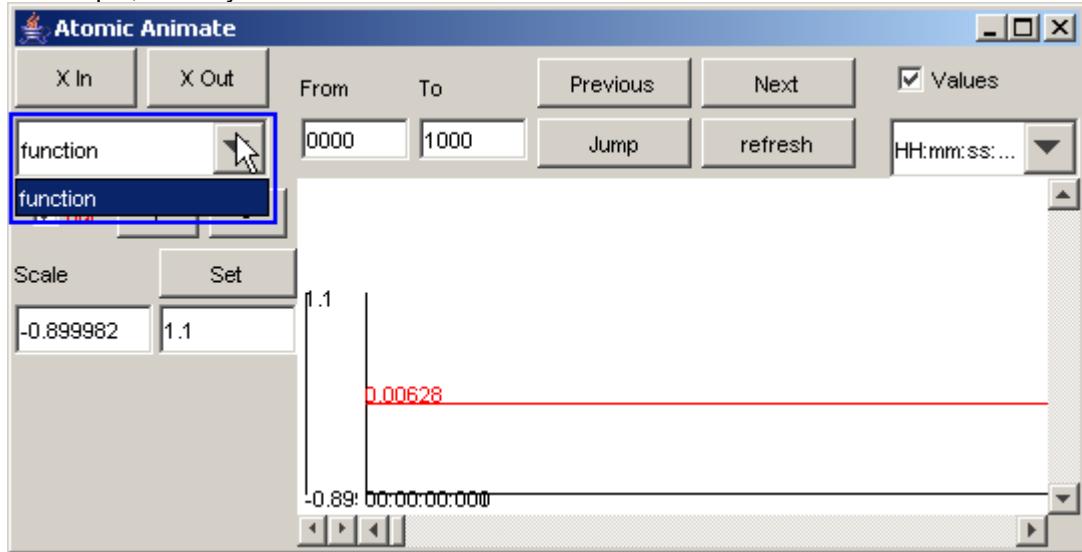


Figure 186:Atomic DEVS model visualization

Select the output signals to be plotted

- 1) To remove a signal from the visualization, uncheck or left-click the box beside the signal name. The plot of the output signal will be removed from the visualization window.
In this example, the function model has one signal, **out**.

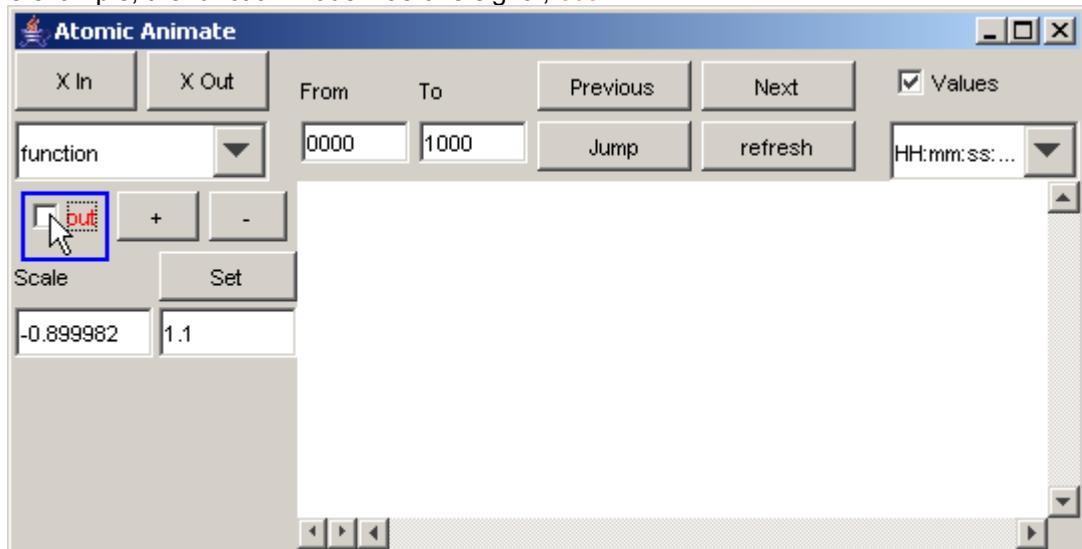


Figure 187: Removing output signal from visualization window

- 2) To include a signal in the visualization, check or left-click the box beside the signal name.

8.16.3.2.1 Select the time format for the horizontal axis

- 1) Left-click on the drop-down list located below the Values checkbox. From the drop-down list, select or left-click the appropriate time format to be displayed on the horizontal axis.

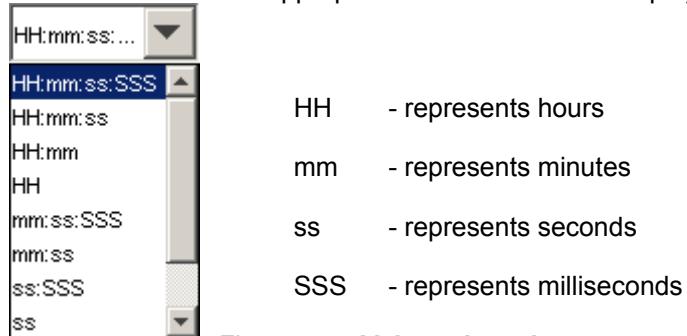


Figure 188: Values drop down menu

In this example, the time selected time format is SSS (which represents milliseconds).

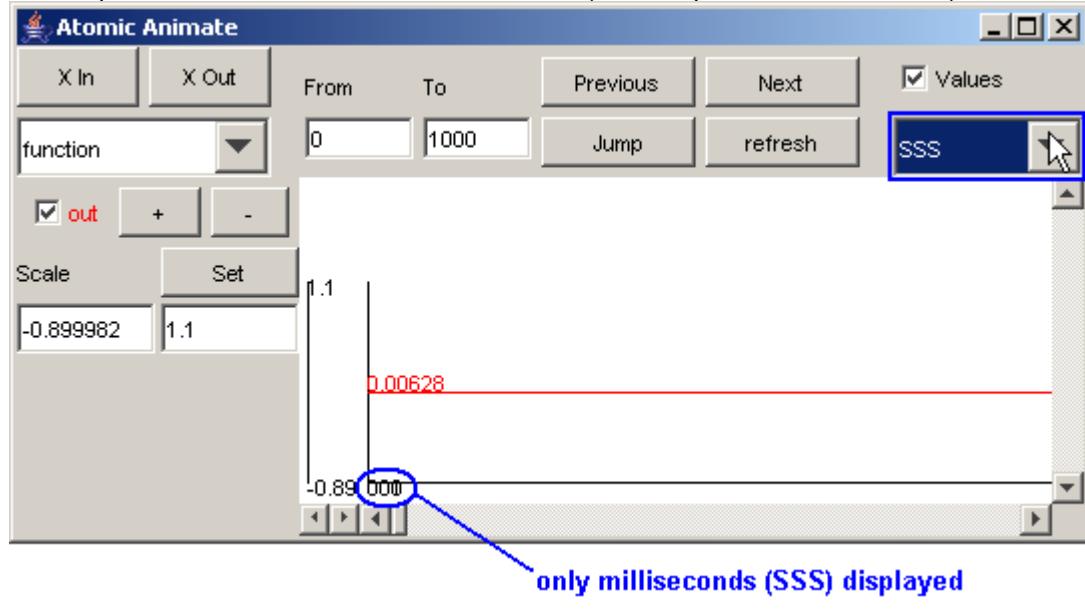


Figure 189: Specifying time

The entire visualization plot for the **out** signal of function model appears as follows:

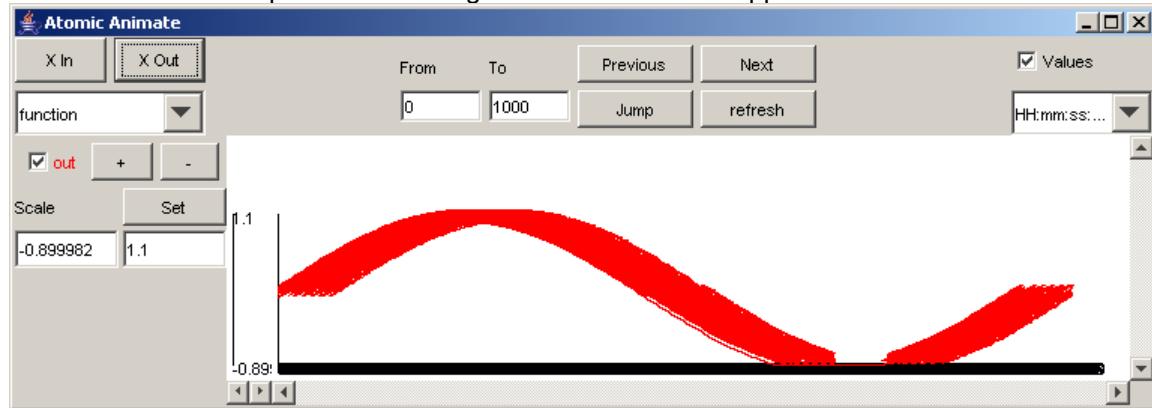


Figure 190: Visualization of the out signal

For upcoming sections, the following plot will be modified:

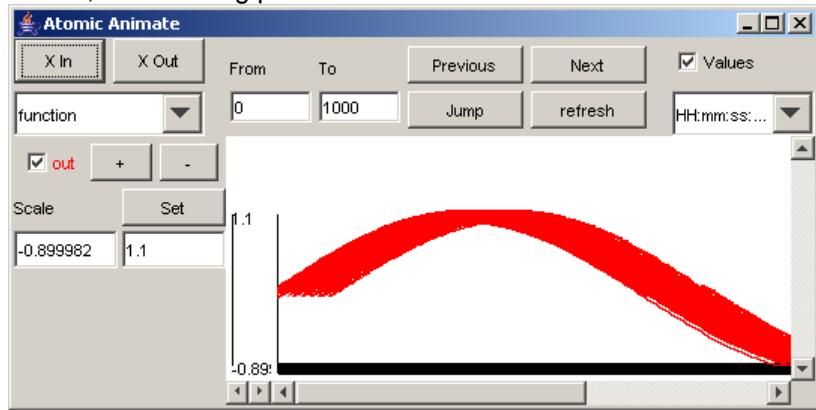


Figure 191: Plot to be modified

8.16.3.2.2 Show the values of the output signal on the plot

1) To remove the output signal values from the plot, uncheck or left-click the box beside Values. The values of the output signal(s) will be removed from the visualization.

In this example, the values of the signal **out** are removed, and so are no longer visible.

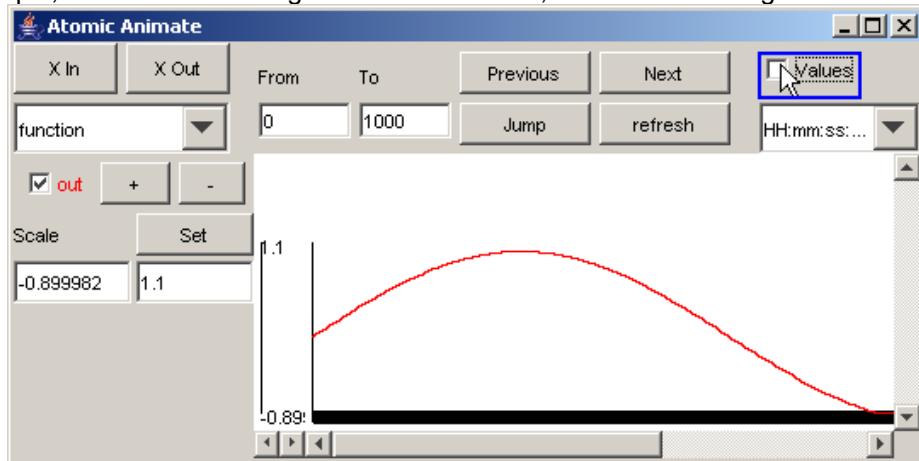


Figure 192: Removing values of the out signal

2) To include the output signal values in the plot, check or left-click the box beside Values. The values of the output signal(s) will be added to the visualization.

In this example, the values of the signal **out** are added, and so are visible. (The values are not legible due to their proximity.)

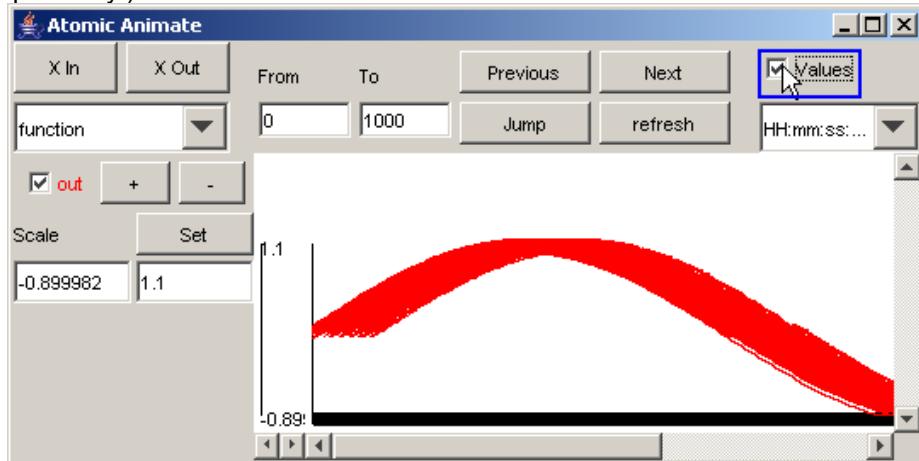


Figure 193: Adding the values of the out signal (making the signal visible)

Note: When the plot is zoomed in (and the time format changed), the values can be made legible:

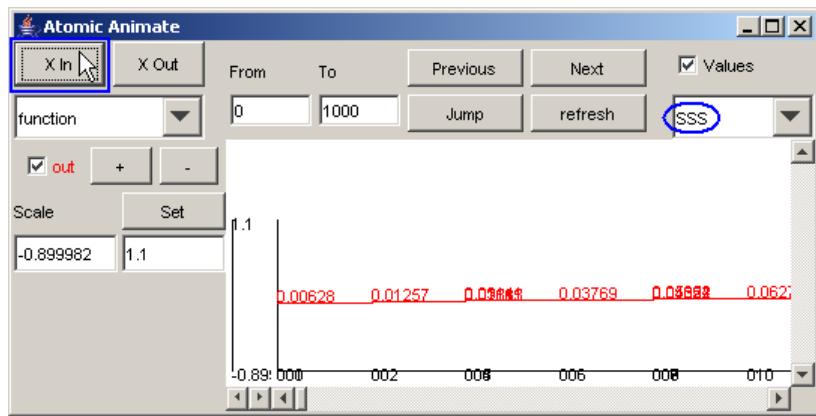


Figure 194: Viewing the values only

(Zooming in and out of the horizontal axis is described in the following section.)

Zoom in or out of the horizontal axis of the plot

- 1) To 'zoom in' or stretch the horizontal scale, press the X In button. The length of the horizontal scale will increase.

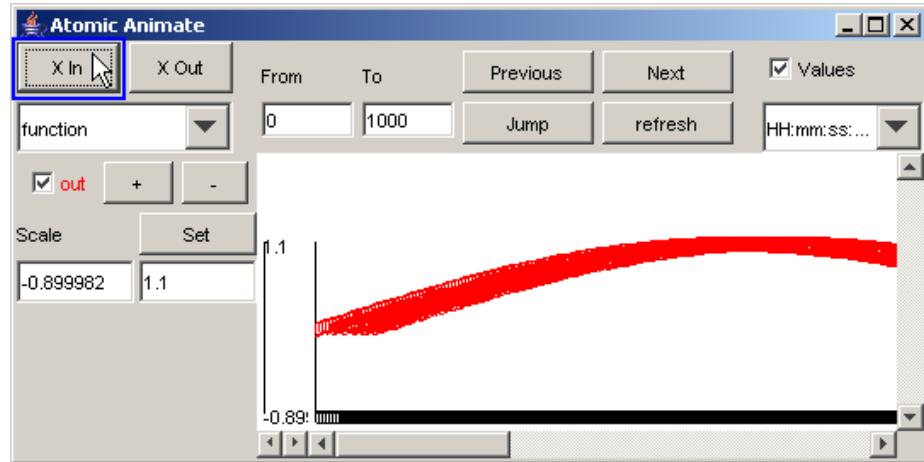


Figure 195: Zooming in on the horizontal scale

- 2) To 'zoom out' or compress the horizontal scale, press the X Out button. The length of the horizontal scale will decrease.

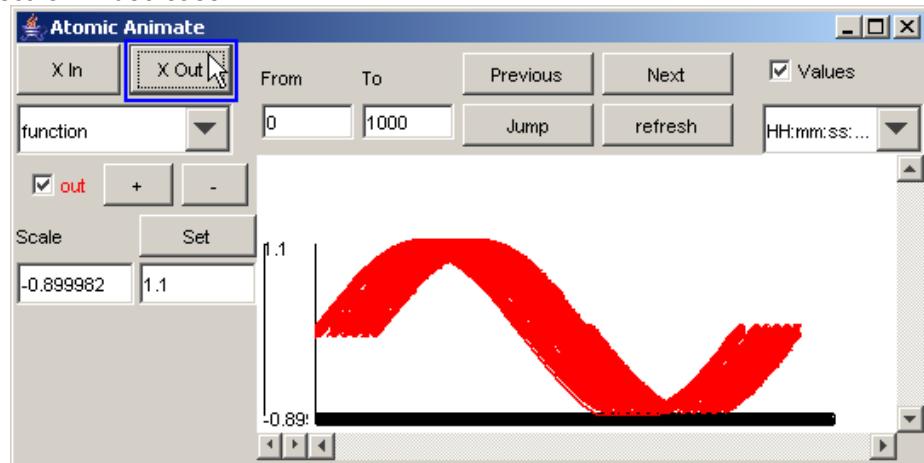


Figure 196: Zooming out on the horizontal scale

Zoom in or out of the vertical axis of the plot

1) To 'zoom in' or stretch the vertical scale, press the + button. The length of the vertical scale will increase.

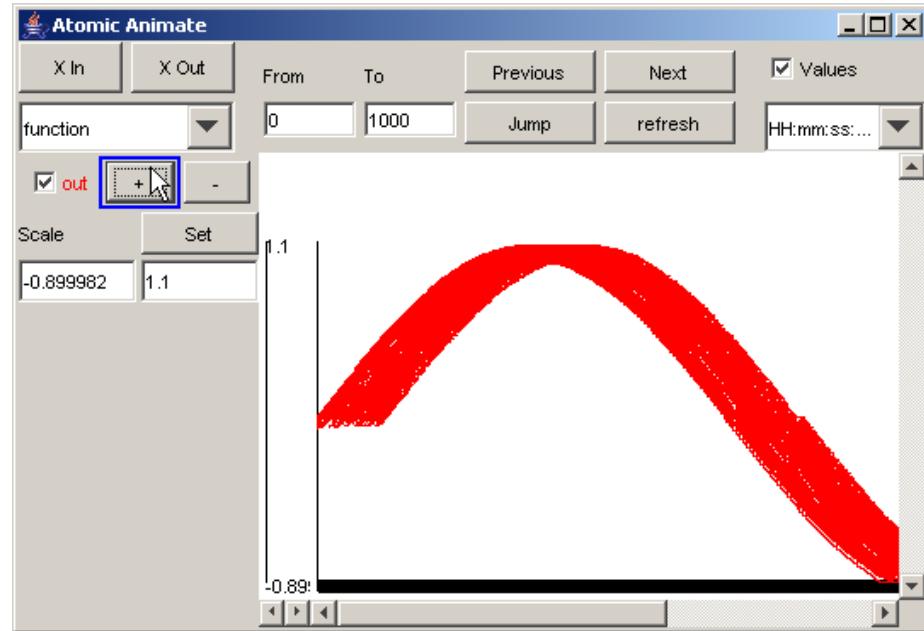


Figure 197: Zooming in on the vertical scale

2) To 'zoom out' or compress the vertical scale, press the - button. The length of the vertical scale will decrease.

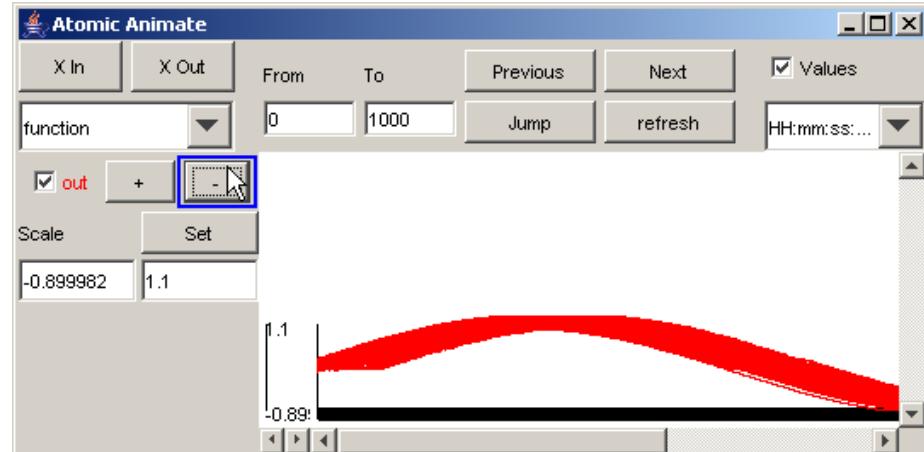


Figure 198: Zooming out on the vertical scale

8.16.3.2.3 Set the lower and upper bounds of the vertical scale

1) To set the lower and upper bounds of the vertical scale, first specify the desired lower and upper bounds in the Scale fields.

In this example, the upper bound is specified as 2.0, while the lower bound remains unchanged.

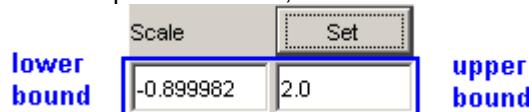


Figure 199: Setting lower and Upper bounds

2) Press the Set button. The parts of the output signal that fall within the lower and upper bounds will be displayed. (The values for the lower and upper bounds are visible on the vertical axis.)

In this example, the parts of the output signal that fall within -0.899982 and 2.0 on the vertical scale are displayed.

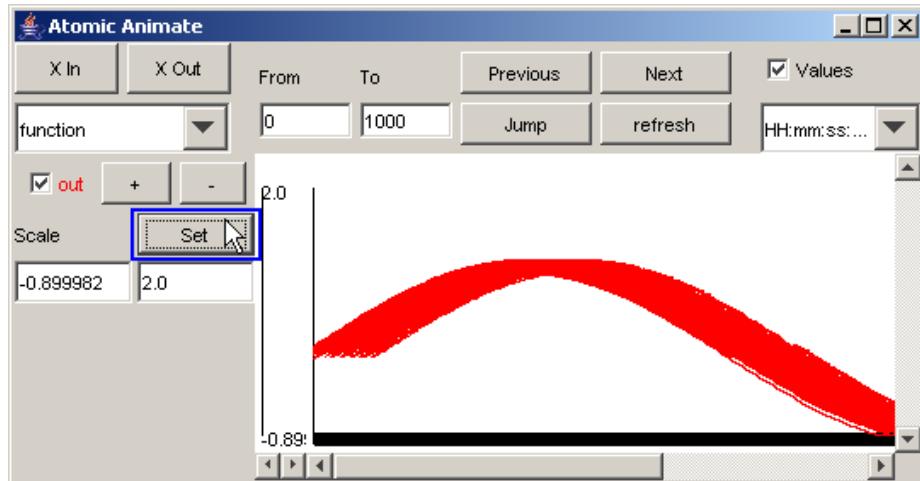
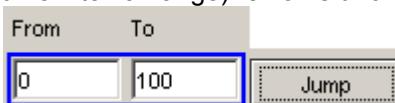


Figure 200: Displaying the signal between the lower and upper bounds

8.16.3.2.4 Jump to the specified time interval

1) To go directly to a particular time interval of the visualization, first specify the desired range of the time interval in the From and To fields.

In this example, the To field (or upper bound of the time interval range) is specified as 100, while the From field (or lower bound of the time interval range) remains unchanged.



The screenshot shows a software interface with two input fields labeled 'From' and 'To'. The 'From' field contains '0' and the 'To' field contains '100'. A blue rectangular box highlights the 'To' field, indicating it is selected or active.

Figure 201: Specifying desired range of time

2) Press the Jump button. The parts of the output signal that occur at times that fall within the From and To range will be displayed. (The values for the time interval range are visible on the horizontal axis.)

In this example, the parts of the output signal that occur between 0 and 100 (milliseconds) are displayed.

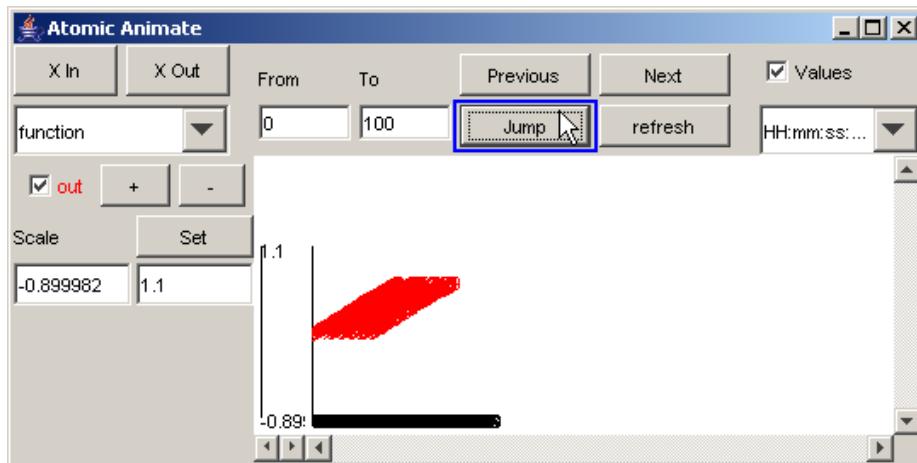


Figure 202: Viewing only the specified time range

Note: When zoomed in (ie. by pressing the X In button several times), the plot from the previous figure will appear as follows:

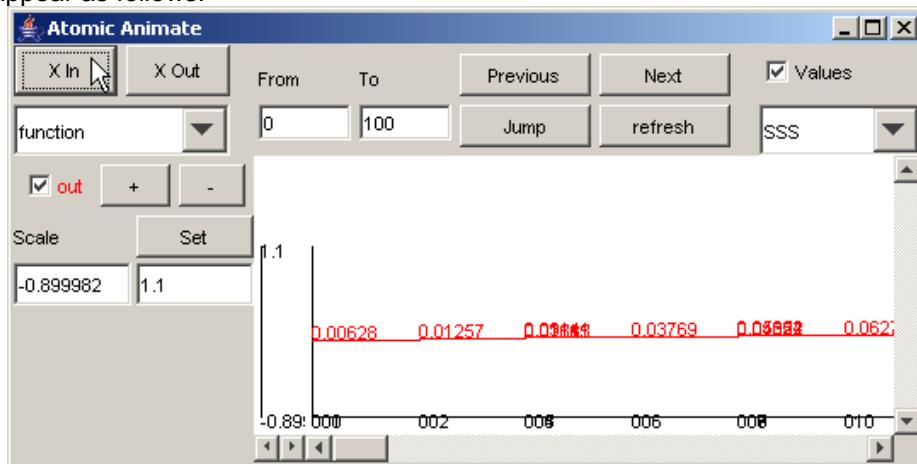


Figure 203: Zooming in to the previous plot

Go to the previous/next time interval

1) To go to the next increment of the specified interval, press the Next button.

In this example, using the same time interval as previously specified, when the Next button is pressed, the parts of the output signal that occur between 100 and 200 (milliseconds) are displayed.

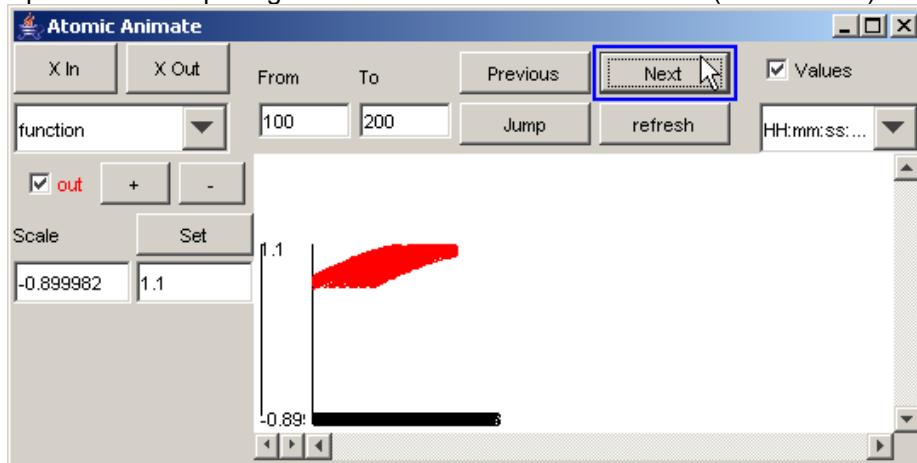


Figure 204: Going to the next time interval

In this example, when the Next button is pressed again, the parts of the output signal that occur between 200 and 300 (milliseconds) are displayed.

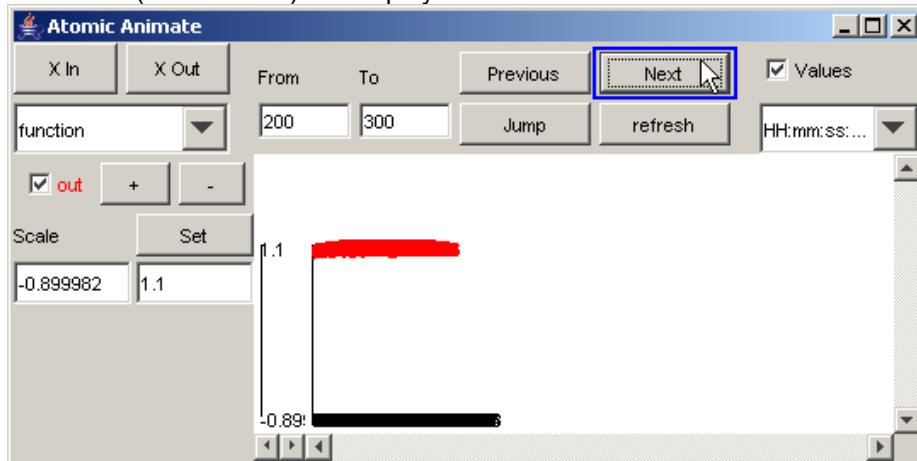


Figure 205: Signal between 200 and 300 milliseconds

2) To go to the previous increment of the specified interval, press the Previous button.

In this example, when the Previous button is pressed, the parts of the output signal that occur between 100 and 200 (milliseconds) are displayed. This is the same plot as the first plot seen in step 1.

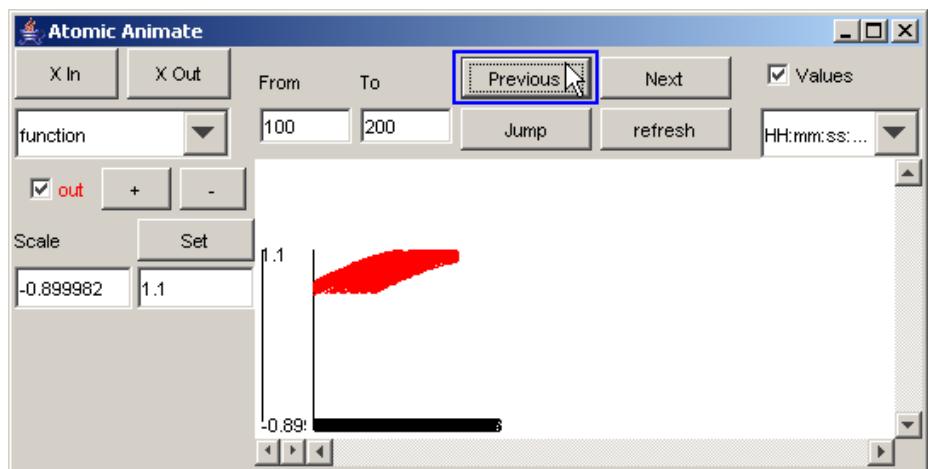


Figure 206: Going to the previous increment

Jump to a specific time

1) To go directly to a specific instance of time of the visualization, first specify the instance in both the From and To fields.

In this example, the specific instance to be plotted is 200.

From	To
200	200
Jump	

Figure 207: Jumping to a specific instance

2) Press the Jump button. The part of the output signal that occurs at the specified instance will be displayed. The value for the specified instance of time will be visible on the horizontal axis. The output value for the signal at the specified instance will be visible on the plot.

In this example, the part of the output signal that occurs at the instance of 200 (with a corresponding time of 196 milliseconds) is displayed. The output value of the signal at 196 milliseconds is 0.94299.

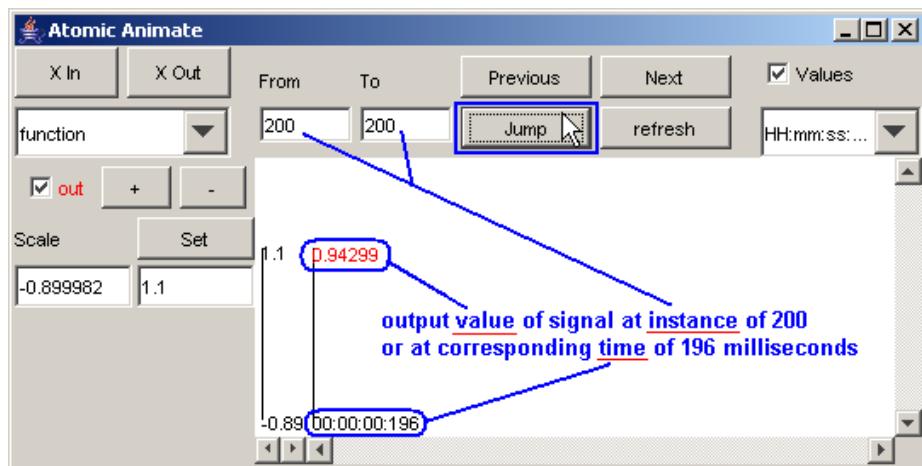


Figure 208: Viewing a specific instance

Note: When the From and To fields contain the same value, the Previous and Next buttons are inoperable.

Note: As demonstrated earlier, when output signal values are removed from the plot (by unchecking the Values checkbox), no output values will be displayed. A red line representing the range of values of the output signal will be visible.

In this example, when the single output signal value is removed from the plot, the red line is barely visible, since the "range" of the output signal is limited to one single value.

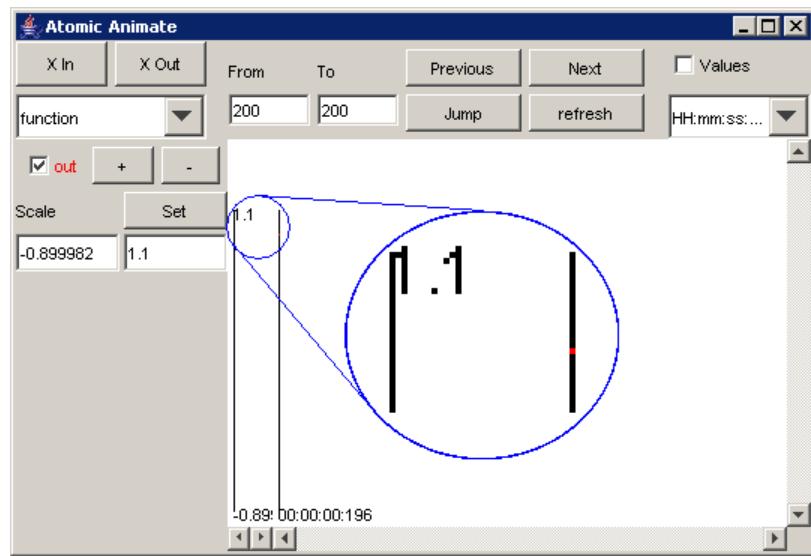


Figure 209 : Range of output signal limited to one value

AtomicAnimate Example of Coupled-DEVS: 4BitCounter

To run this example, download the 4BitCounter project.

4BitCounter is a coupled-DEVS model composed of multiple atomic-DEVS models that each contain multiple output signals.

The main functionality of the Atomic Animate visualization window described in the previous example (FunctionEval) also applies for this example. However, while the previous example (FunctionEval) illustrated the visualization of a single atomic-DEVS model, this example (4BitCounter) illustrates the visualization of (a coupled-DEVS model composed of) multiple atomic-DEVS models.

Using the same procedure as in the previous example, in the Set and atomic animate dialogs, open 4Counter.log.

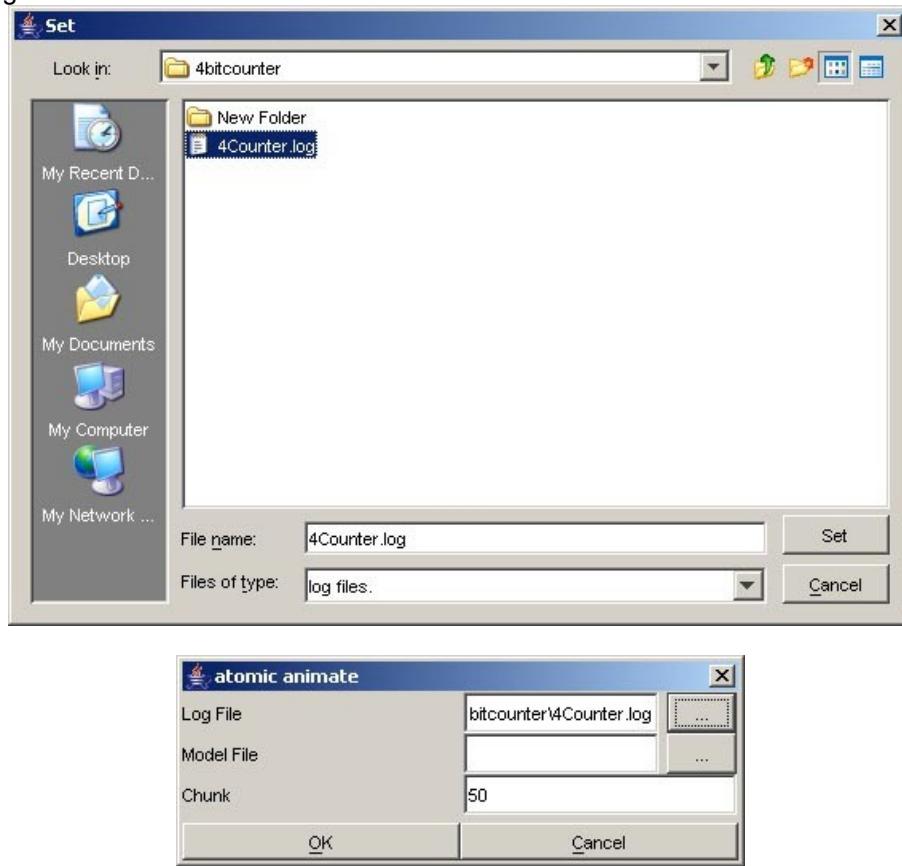


Figure 210: Opening 4Counter.log

The models that send/receive messages within 4Counter.log will be available for visualization. The file 4Counter.log contains the messages sent/received by the top model, which is composed of the following models: 4count@Process_4_Counter, b1@Signal, b2@Signal, b3@Signal, b4@Signal, clock. Whereas the first four models are atomic-DEVS models, the fifth model, clock, is a coupled-DEVS model composed of the atomic-DEVS models: inv@Process_Inv, sig1@Signal. So, for this example, the following models/components are available for visualization: clock, 4count, inv, b4, b3, top, b2, b1, sig1.

The following Atomic Animate visualization window will be displayed.

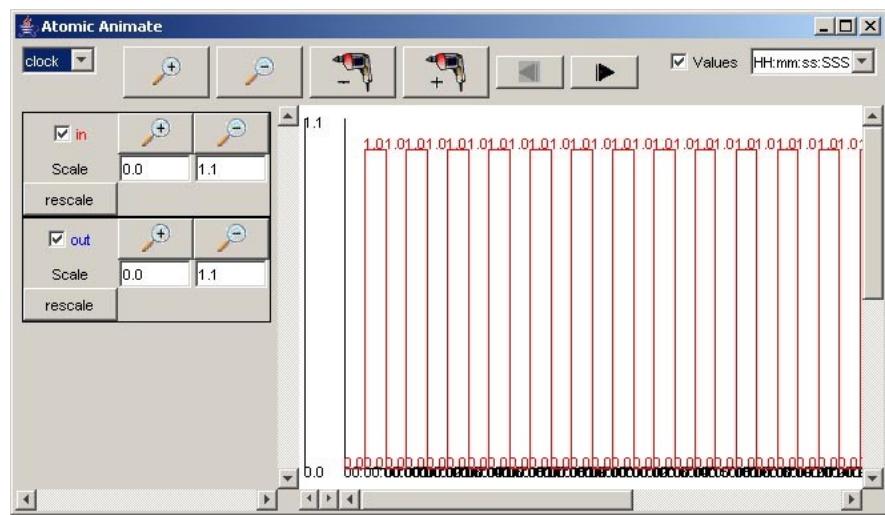


Figure 211: Atomic Animate Visualization Window

By default, the signals of the first model/component in the list - in this example, clock - will be displayed.

Also by default the visualization plot will display the range of output values of the selected signals for the time interval spanning the entire simulation. In this example, the time interval of [00:00:00:000, 00:00:00:400] corresponds to the entire simulation, and the in and out signals of clock each have a range of [0.0, 1.0] for this interval.

Also by default, the lower/upper bounds of the vertical Scale will initially correspond to the minimum/maximum of the output value range. So, the vertical scale for an output signal will automatically be set such that the entire range of output values for the signal (for the initial time interval) will be visible in the visualization plot. In this example, the in and out signals of clock each initially have a vertical scale with bounds [0.0, 1.1].

The differences in the functions of the Atomic Animate visualization window, when used for displaying multiple atomic-DEVS models, will be described:

- select the model/component to be visualized
- *availability of signals for the selected model/component (explanation)
- select the (available) signals of the selected model/component to be plotted

Note: The plot can be more clearly viewed by zooming out (ie. pressing the “-“ Magnification button), and changing the time format appropriately. The format of the time (horizontal axis) values was changed to SSS, since the simulation ends at 00:00:00:400 (according to the last message time in the .log file).

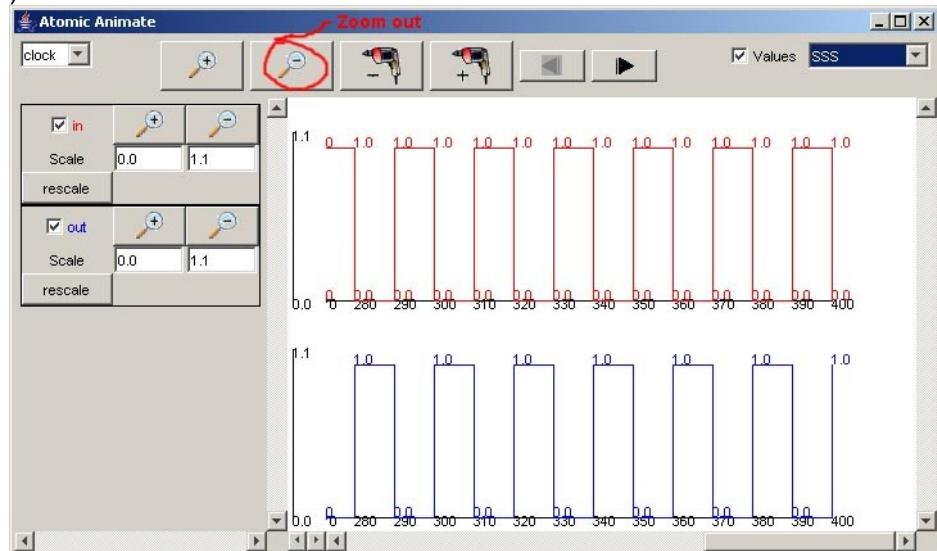


Figure 212: Formatted output

Select the model/component to be visualized

1) The models/components available for visualization can be seen in the drop-down list.

To select a model/component to be visualized, first left-click on the drop-down list.

Note: The order of the models/components in the drop-down list is arbitrary.

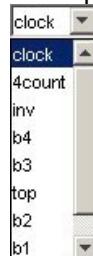


Figure 213: Selecting a model component

2) From the drop-down list, select or left-click the model/component to be visualized. The signals of the selected model/component will be displayed.

Note: *For each component, the order of the available signals (in the column below the drop-down list) is arbitrary.*

Also for each component, the colors of the available signals are automatically assigned based on their order. The first available signal will be red, the second blue, the third green, etc.

The order of the visible visualization plots will correspond to the order of available signals that are selected for visualization.

In this example, 4count is selected, and the signals available are: q1 (green), q4 (brown), q3 (pink), and q2 (light blue). Since all the available signals are checked, all the signals are plotted.

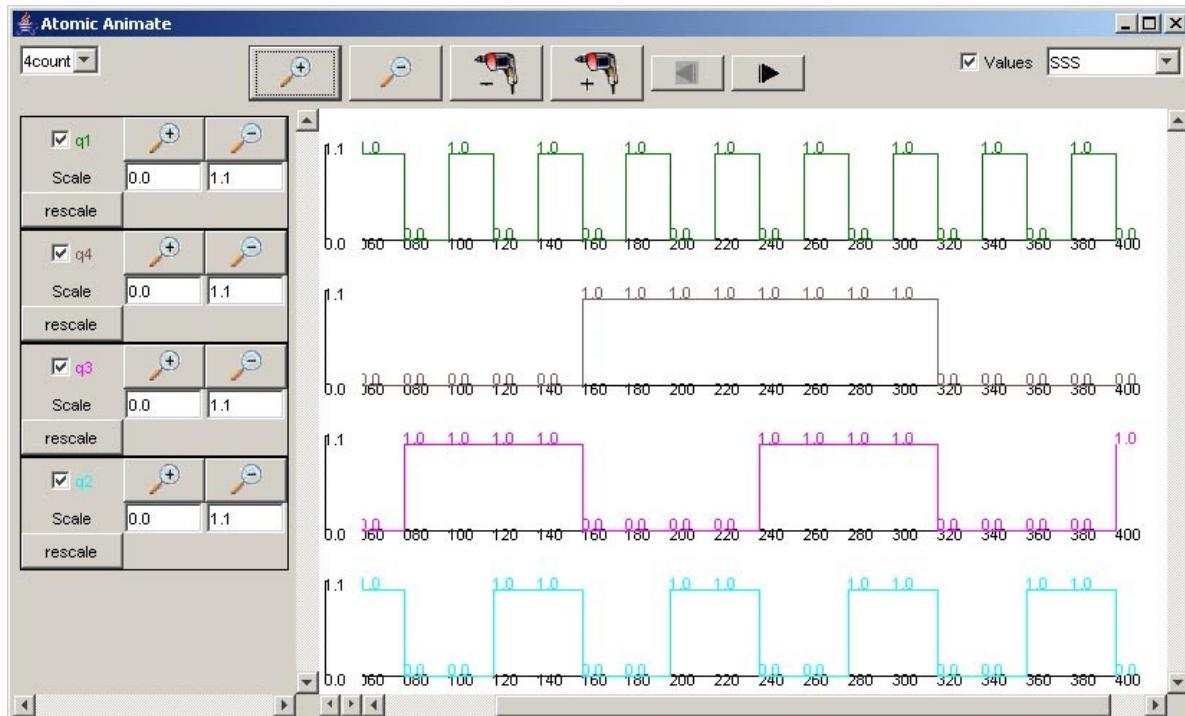
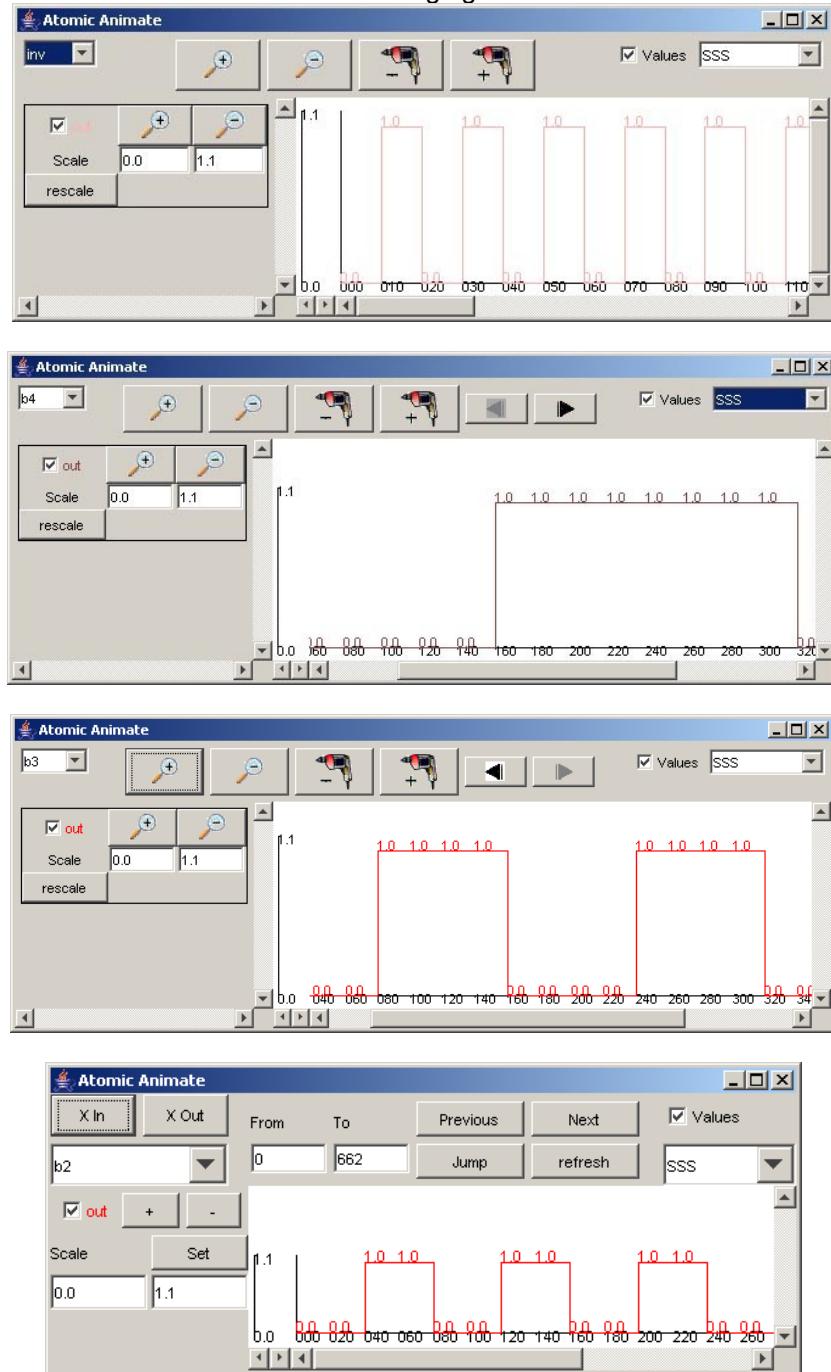


Figure 214: Viewing all signals

For the rest of the components in the drop-down list, the available signals and range of output values for the entire simulation can be seen in the following figures.



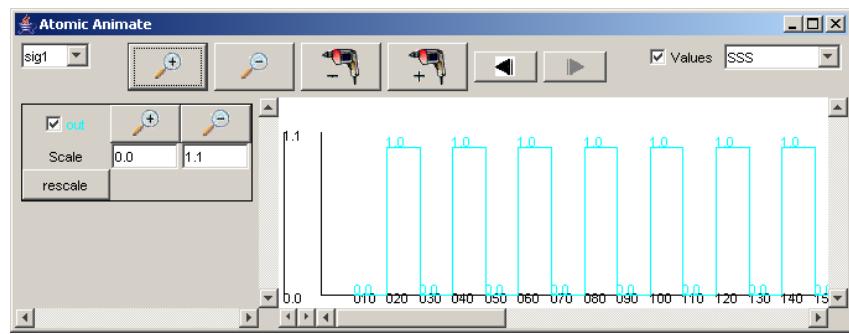
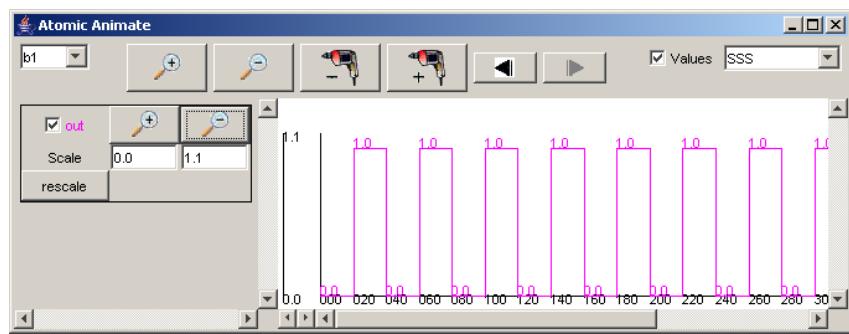
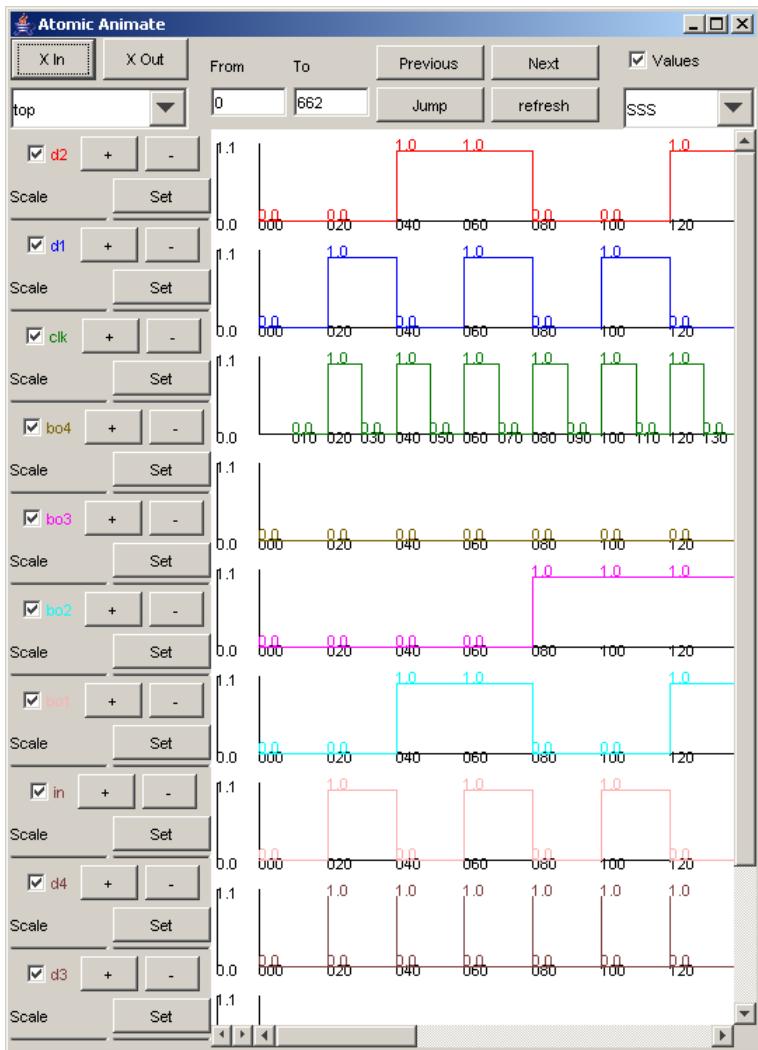


Figure 215: Available signals and range of output values



When the *top* component is selected, the signals available are:
in, clk, d1, d2, d3, d4, bo1, bo2,
bo3, bo4.

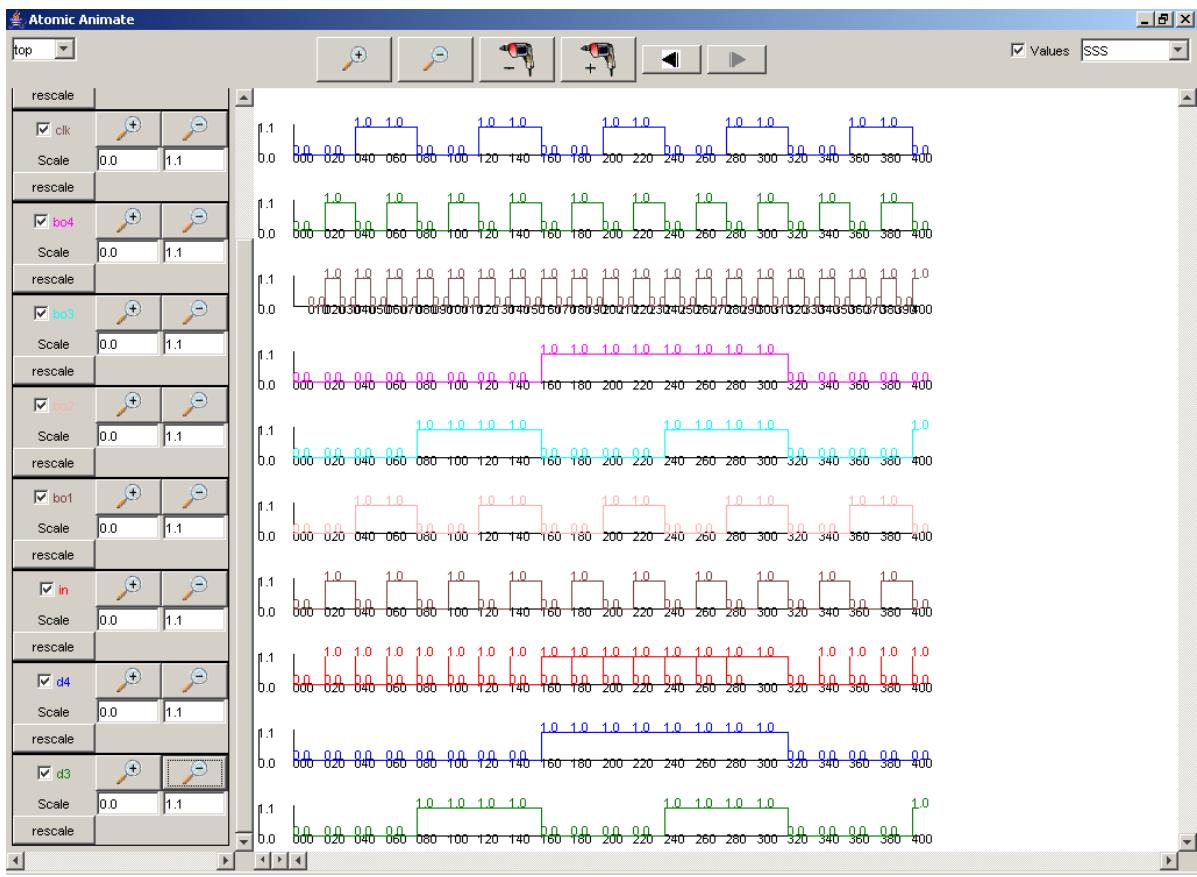


Figure 216: Viewing all signals in one window

Note: When the top component is selected, since the time interval is initially set to correspond with the entire simulation, all the available signals of the top component will be checked. Due to the number of available signals in the top component, the space required to display the Scale fields of the signals exceeds the height of the Atomic Animate visualization window. The Scale fields can be made visible if the height of the visualization window is increased.

It may not always be possible to increase the height of the visualization window to accommodate the Scale fields. (Please see Appendix B - 4BitCounter for more details.)

The visualization plots of the checked signals also exceed the height of the Atomic Animate visualization window. The vertical scrollbar can be used to view the visualization plots that are located beyond the bottom edge of the window.

Availability of signals for the selected model/component

A selected component can have multiple signals (usually called ports).

When a component is selected, the signal associated with the component can be selected to be displayed as a plot in the visualization.

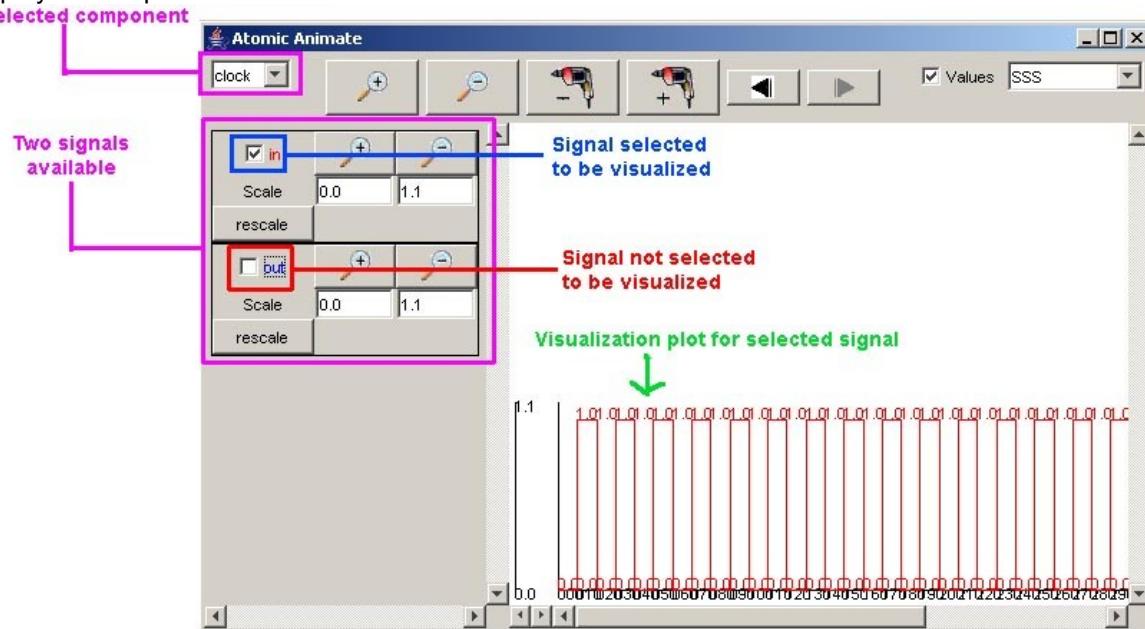


Figure 217: Selecting a signal to be displayed

A component will be displayed in the drop-down list if values are sent to at least one signal of the component. (ie. if at least one signal is available for the component)

The following behaviour still applies: *For a selected component, the order of the available signals is arbitrary. The colours of the available signals are automatically assigned based on their order. The first available signal will be red, the second blue, the third green, etc. The order of the visible visualization plots will correspond to the order of available signals that are selected for visualization.*

8.16.4 CoupledAnimate using: Coupled-DEVS

The message values sent/received via ports within a coupled model can be visualized using the graph (.gcm file) of the coupled model. The visualization displays the output values with the corresponding output ports, superimposed on the coupled model's graph.

The steps required for visualizing the message values sent/received via the ports of a coupled model are as follows:

- 1) In the Animate menu, left-click on CoupledAnimate. A coupled animate dialog box will be displayed.



Figure 218: Choosing coupled animate

- 2) Specify the Log file that contains the messages sent/received by the coupled-DEVS model to be visualized (using the same procedure as for AtomicAnimate).
- 3) Left-click on the browse button for the Coupled Model Definition. A Set dialog will be displayed. Select the .gcm file that corresponds to the coupled-DEVS model to be visualized. After choosing the appropriate .gcm file, left-click on Set.

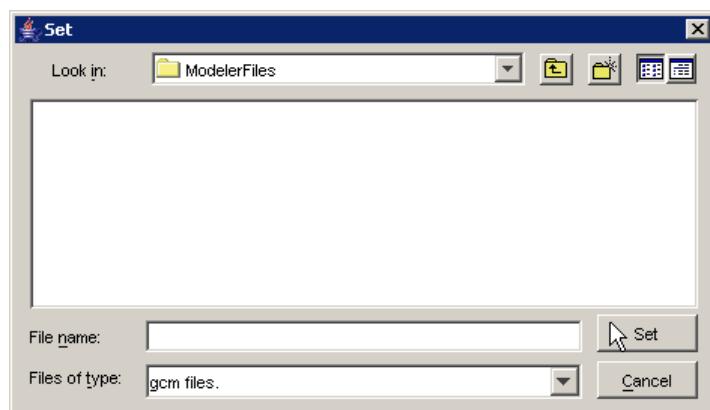


Figure 219: Set dialog box

- 4) Type in the appropriate Delay between displays. The Delay value (milliseconds) will be the amount of time the visualization waits between each update of the display. (The speed of the visualization is determined by the Delay value.) By default, the Delay is 1000 milliseconds (or 1 second).

If an invalid file type is chosen, the (blank) visualization window will be displayed as follows:

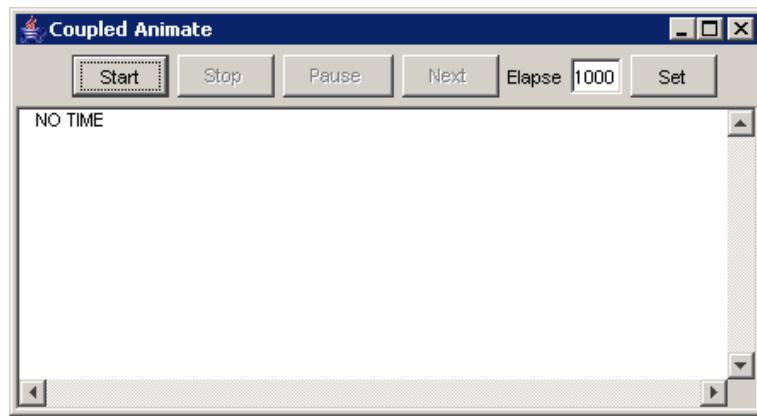
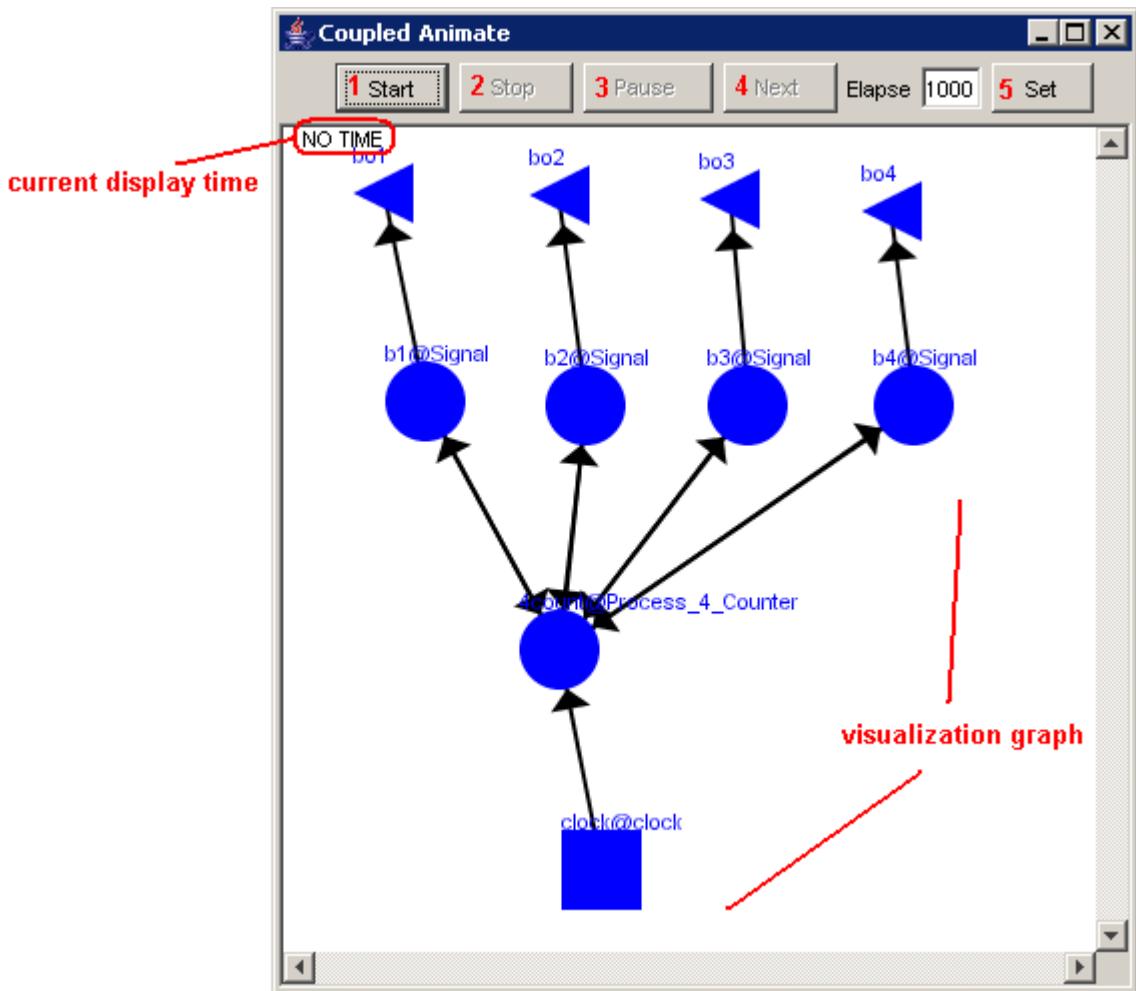


Figure 220: **Blank view for invalid file type**



The following example demonstrates the functionality of the Coupled Animate visualization window, using the coupled model 4BitCounter.

CoupledAnimate Example of Coupled-DEVS: 4BitCounter

To run this example, download the 4BitCounter project from:

<http://www.angelfire.com/sc3/schao2/projects/4BitCounter.zip>

4BitCounter is a coupled-DEVS model composed of multiple atomic-DEVS models that each contain multiple output signals.

In the Set and coupled animate dialogs, open 4Counter.log and 4counter.gcm as the Log File and Coupled Model Definition, respectively.

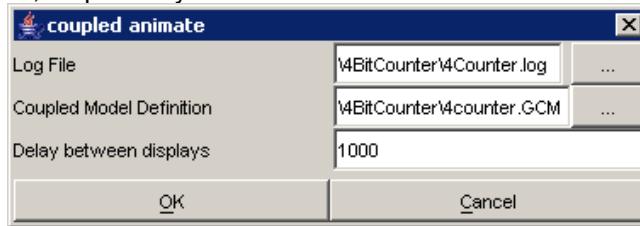


Figure 221: Opening the coupled model files

The coupled-DEVS model that sends/receives messages within 4Counter.log, and that is represented in graphical form in 4counter.gcm, will be available for visualization.

The following Coupled Animate visualization window will be displayed.

Figure 222: Coupled model visualization for 4bitcounter

The Coupled Animate visualization window has several commands for the visualization, such as:

- start the visualization (1)
- stop the visualization (2)
- pause the visualization (3)
- go to the next display of the visualization (4)
- set the delay between displays (5)

Start the visualization

1) Left-click the Start button. The visualization will start and continue updating the graphical display, along with the corresponding display time (according to the message times of the .log file).

In this example, the visualization has been started, and has the current display time of 00:00:00:070, with the output values visible alongside their corresponding ports.

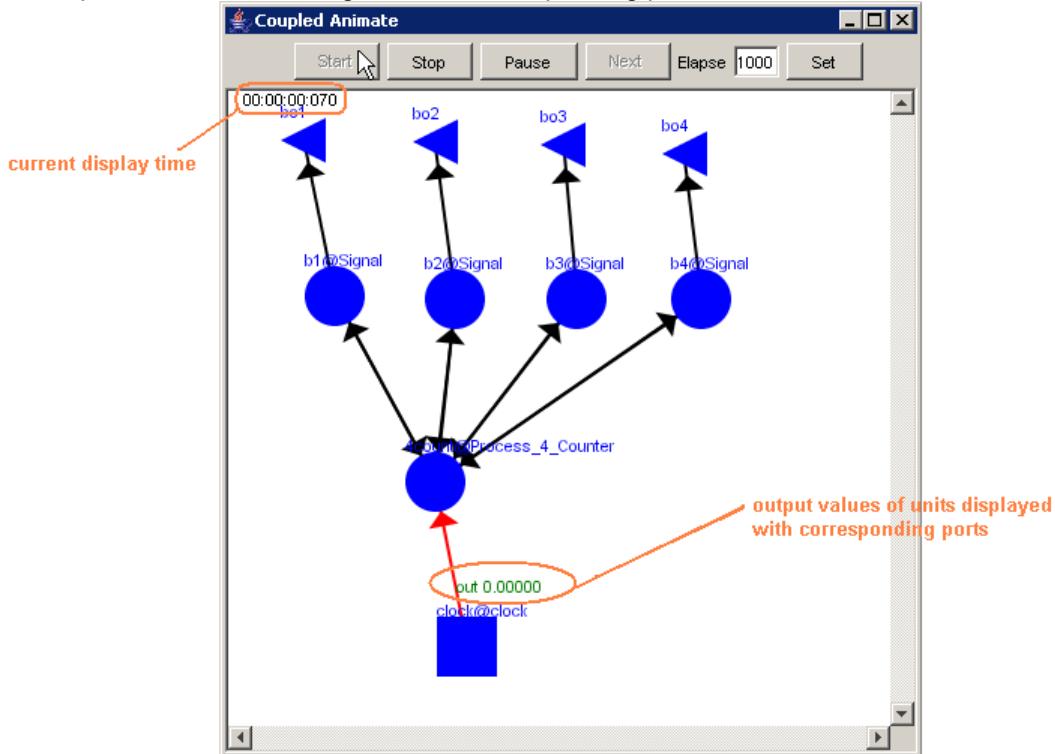


Figure 223: Viewing the current display time and output values of units

2) After the visualization has started, it can be run to the end of the simulation time, stopped, or paused.

- After the visualization has started, if no other buttons are pressed, the visualization will continue until it reaches the end of the simulation (ie. the last message time in the .log file).

In this example, after pressing the Start button, if no other buttons are pressed, the visualization will continue until it reaches 00:00:00:400, which is when the simulation ends in the .log file.

- The Stop and Pause buttons are described in the following sections.

Stop the visualization

- 1) Left-click the Stop button. The visualization will stop at the current display time. The output values occurring at the current display time will remain visible on the graph.

In this example, the visualization has been stopped at the current display time of 00:00:00:260, with the output values visible alongside their corresponding ports.

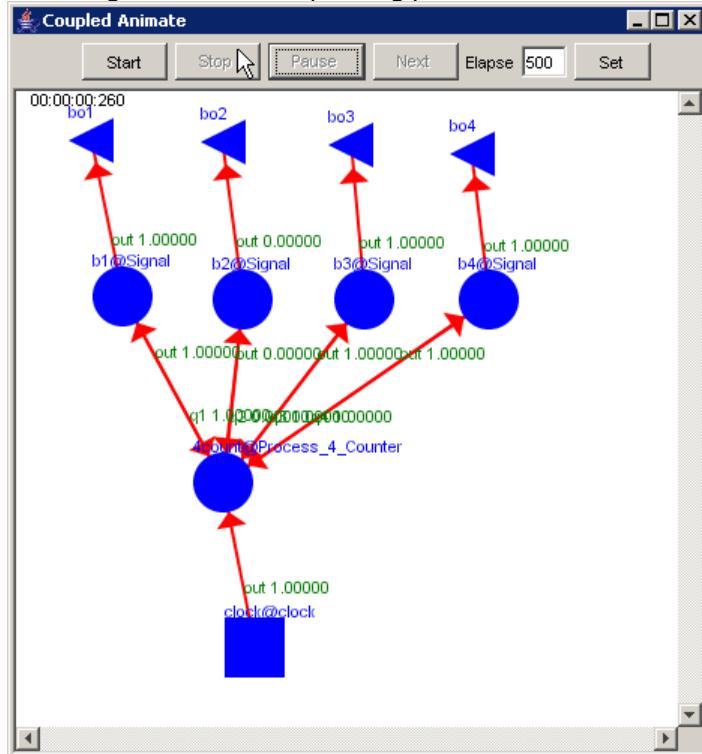


Figure 224: Stopped visualization

- 2) After the visualization has stopped, it can be restarted (at the beginning of the simulation).

- After the visualization has stopped, if the Start button is pressed, the visualization will restart only at the beginning of the simulation, ie. at 00:00:00:000. The visualization will not restart at the current display time.
- The Start button is described in the preceding section.

8.16.4.1 Pause the visualization

- 1) Left-click the Pause button. The visualization will pause at the current display time. The output values occurring at the current display time will remain visible on the graph.

In this example, the Pause button is pressed after the current display time of 00:00:00:380, pausing the visualization at 00:00:00:390, with the output values visible alongside their corresponding ports.

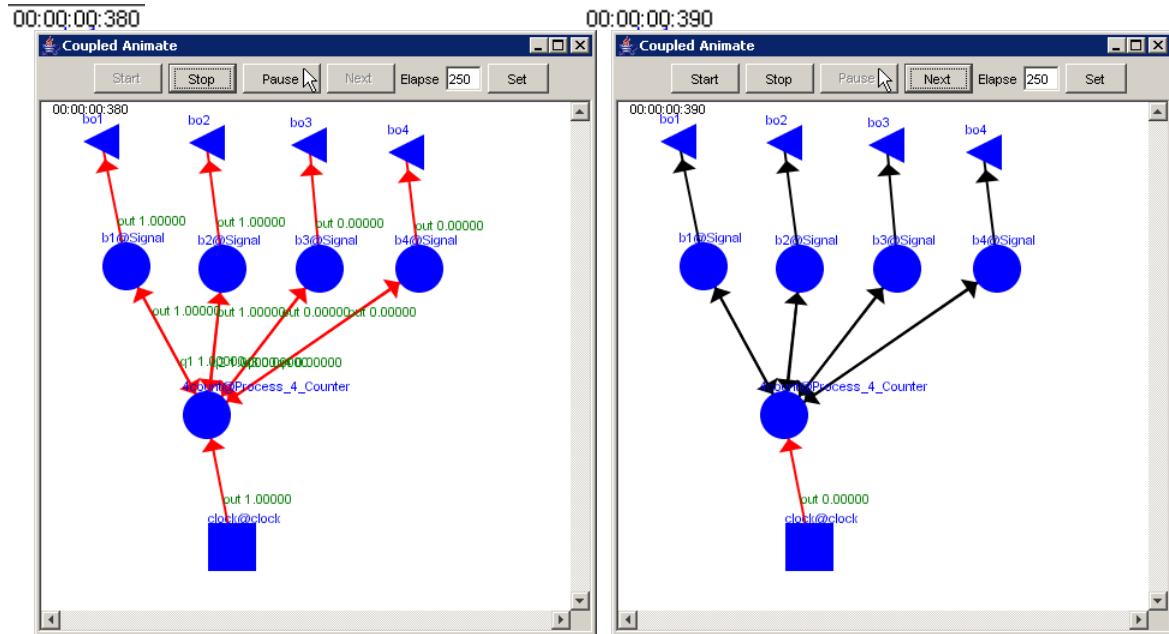


Figure 225: Pausing the visualization

- 2) After the visualization has paused, it can be restarted (at the current display time), stopped, or incremented to the next display.

- After the visualization has paused, if the Start button is pressed, the visualization will restart at the current display time.
- The Start and Stop buttons are described in the preceding sections. The Next button is described in the following section.

8.16.4.2 Go to the next display of the visualization

1) Left-click the Next button. The visualization will increment to the next display, updating the current display time and the output values visible on the graph.

This example continues from the second graph in the previous section - 'Pause the visualization'. In this example, with the visualization paused at 00:00:00:390, the Next button has been pressed. The visualization is incremented to the next display at 00:00:00:400, updating the current display time and the output values displayed on the graph.

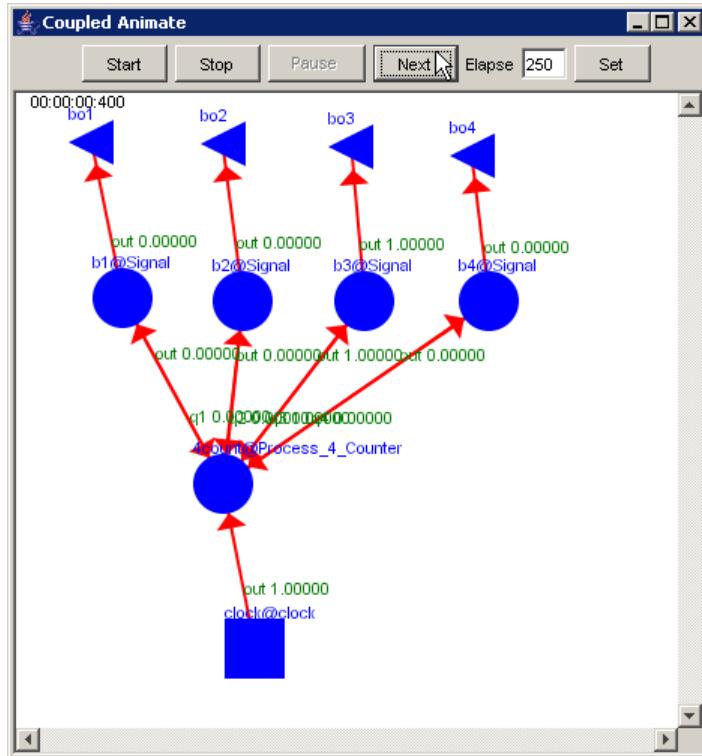


Figure 226: Resumed visualization

2) After the visualization has incremented to the next display, it can be restarted (at the current display time), stopped, or incremented again to the next display.

- After the visualization has incremented to the next display, if the Start button is pressed, the visualization will restart at the current display time.
- The Start and Stop buttons are described in the preceding sections. The Next button is described in this section.

Set the delay between displays

1) Type the appropriate delay length (milliseconds) in the Elapse field. Press Set.



The delay can be set while the visualization is running (ie. after the visualization has been started), after the visualization has been stopped, or while the visualization is paused (ie. after the visualization has been paused).

8.16.5 Cell-DEVS animation using: Atomic Cell-DEVS, Coupled-DEVS

The steps required for visualizing the cell values of an atomic cellular model are as follows:

- 1) In the Animate menu, left-click on Cell-DEVS animation.

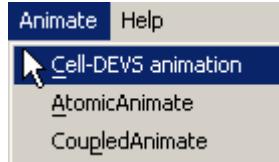


Figure 227: Cell-DEVS animation menu

The following Cell-DEVS animation visualization window will be displayed.

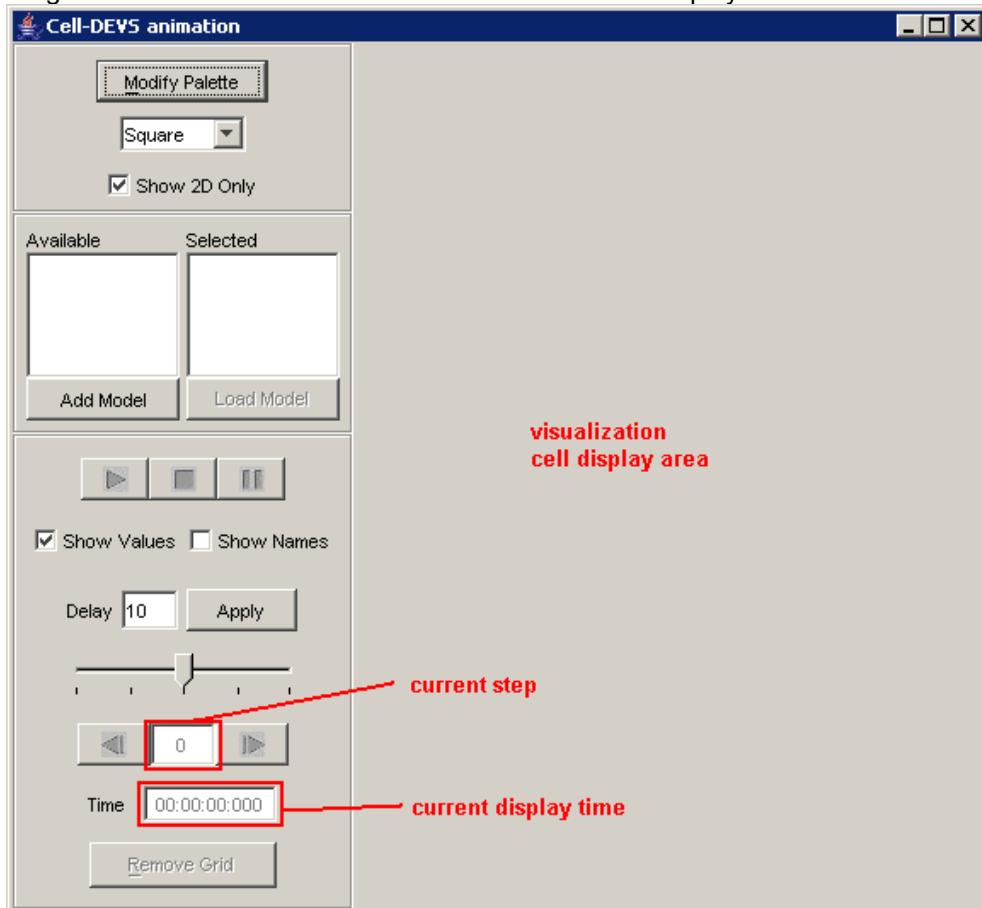
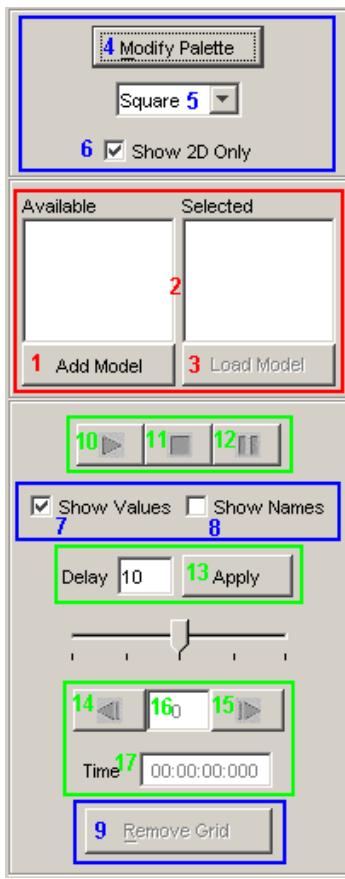


Figure 228: Cell-DEVS Animation window

- 2) A model must first be loaded before it can be visualized. The loading of models will be described in upcoming sections.
- 3) The appearance of the visualization can be modified before the visualization starts. (The appearance of the visualization can also be modified while the visualization is running.) The options available for changing the appearance of the visualization will be described in upcoming sections.
- 4) The individual steps/displays in the visualization can be navigated to directly or indirectly. Navigating through the visualization will be described in upcoming sections.

Thus, steps 2, 3, and 4 are the main steps for visualizing an atomic Cell-DEVS model. Each of the steps can be further broken down, as shown in the following figure and list.



The Cell-DEVS animation visualization window has functions that can be grouped according to the step in which the function is involved.
(See Figure 229)

- loading models to be visualized
- modifying the appearance of the visualization
- running/navigating the visualization

For each of the main steps, there are several functions.

For loading models to be visualized, functions include:

- adding models to be made available (1)
- selecting available models to be loaded (2)
- loading models to be visualized (3)

For modifying the appearance of the visualization, functions include:

- modifying the palette of the loaded models (4)
- selecting the lattice type (5)
- toggle between 2D and multidimensional display (6)
- showing the cell values of the loaded models (7)
- showing the names of the loaded models (8)
- toggle the visibility of the grid of the lattice (9)

For running/navigating the visualization, functions include:

- playing/starting the visualization (10)
- stopping the visualization (11)
- pausing the visualization (12)
- set the delay between steps of the visualization (13)
- go to the previous/next step in the visualization (14, 15)
- go directly to a particular step in the visualization (16)
- go directly to a particular time in the visualization (17)

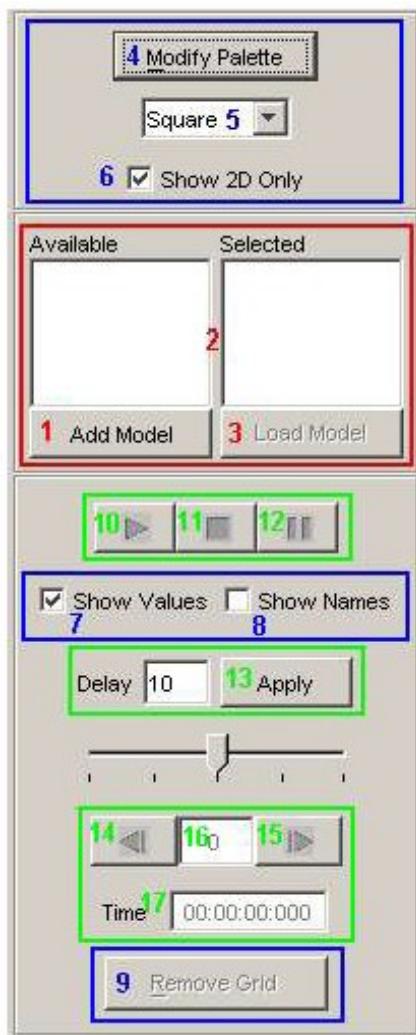


Figure 229: Options for cell DEVS animation.

The following example demonstrates the functionality of the Cell-DEVS animation visualization window, using the atomic Cell-DEVS model Fire. Additional examples for a 3D atomic Cell-DEVS model (SatelliteClouds) and a coupled-DEVS model (RiceField) follow.

Cell-DEVS animation Example of Atomic Cell-DEVS: Fire

To run this example, download the Fire project.

Fire is a 2D (two-dimensional) atomic Cell-DEVS model.

Loading models to be visualized

A model must first be loaded before it can be visualized.

To load a model, the appropriate model must first be added to the Available list. From the Available list, the appropriate model(s) to be visualized must be selected (ie. to the Selected list). Once the models to be visualized are in the Selected list, the models can then be loaded. These functions are described in the following sections.

Adding models to the Available list

- 1) To add an atomic cellular model to the Available list, left-click the Add Model button. The Choose a CD++ DRW or Model File dialog will be displayed.

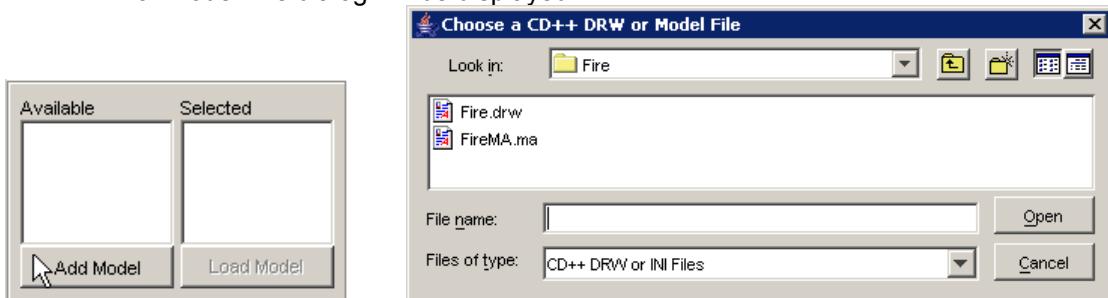


Figure 230: Adding an atomic cellular model

- 2) Two options are available: (a) choose a .ma file and a .log file, or (b) choose only a .drw file.

(a) Choose the appropriate .ma file, and left-click the Open button.

The Choose a CD++ LOG File dialog will be displayed. Choose the appropriate .log file (corresponding to the .ma file), and left-click the Open button.

In this example, FireMA.ma is chosen in conjunction with FireLOG.log.

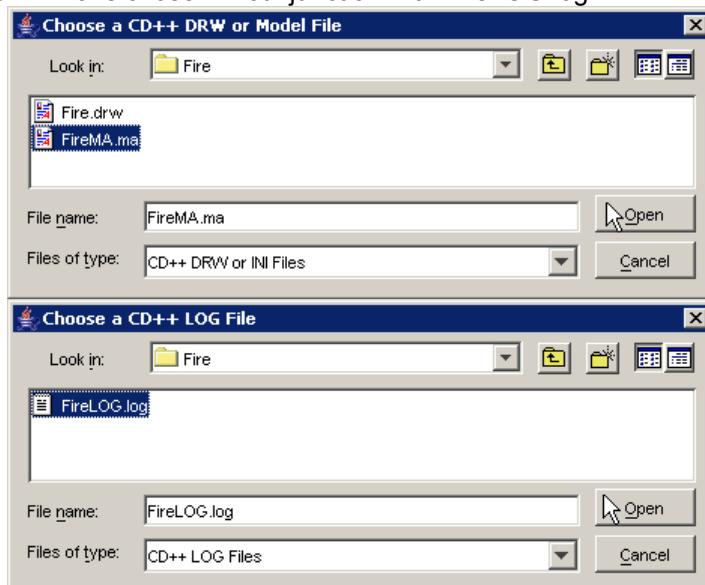


Figure 231: opening the Fire model

(b) Choose the appropriate .drw file, and left-click the Open button.
In this example, Fire.drw is chosen.

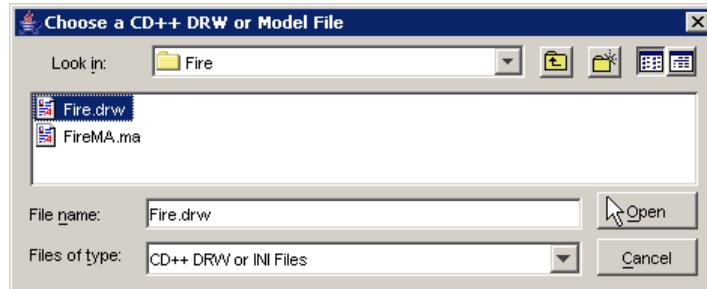


Figure 232: Choosing the appropriate drw file

3) The atomic cellular model name will appear in the Available list. The format of the model name (displayed in the Available list) depends on the option chosen:

- (a) <componentName>@<modelLog.log>
where:
- <componentName> is an atomic cellular model listed included in the .ma file (ie. components :) *
- <modelLog.log> is the actual file name of the .log file

In this example, the format of the model name will be forestfire@FireLOG.log.



Figure 233: forestfire@FireLOG.log
(b) <modelDrw.drw>

where:
- <modelDrw.drw> is the actual file name of the .drw file **

In this example, the format of the model name will be Fire.drw.

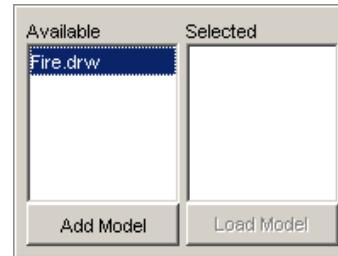


Figure 234: Fire.drw

* Note: (a) If the model in the .ma file contains multiple atomic cellular models in the components: parameter, each of the atomic cellular models will appear in the Available list. All of these atomic cellular models will use the same .log file. (See the RiceField example for more details.)

** Note: (b) Each .drw file corresponds to only one atomic cellular model. (This differs from option (a), where a single .ma file may contain multiple atomic cellular models.)

- 4) To remove an atomic cellular model from the Available list, left-click on the model name in the Available list, and press Delete on the keyboard.

8.16.5.1 Selecting available models to be loaded

1) To select an atomic cellular model (ie. to the Selected list), left double-click on the model name in the Available list.

In this example, the model name (a) forestfire@FireLOG.log, and (b) Fire.drw, are selected.

2) The atomic cellular model name will appear in the Selected list. The format of the model name displayed in the Selected list will be the same as in the Available list.

(a)

In this example, the format of the model name remains as forestfire@FireLOG.log.



Figure 235: selecting forestfire@FireLOG.log

(b)

In this example, the format of the model name remains as Fire.drw.

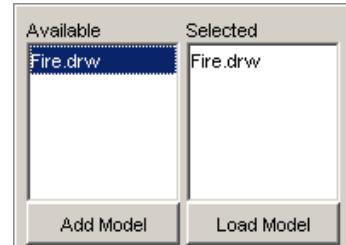


Figure 236: Selecting Fire.drw

Once an atomic cellular model is in the Selected list, the Load Model button will be enabled. (If the Selected list is empty, ie. does not contain any atomic cellular models, the Load Model button will be disabled.)

3) To remove an atomic cellular model from the Selected list, double left-click on the model name in the Selected list. Alternatively, left-click on the model name in the Selected list, and press Delete on the keyboard.

Note: All atomic cellular models in the Selected list will be loaded (and visualized) concurrently when the Load Model button is pressed.

Note: After a cellular model is selected from the Available list, the model will be placed in the Selected list. If the model name in the Available list is deleted, the model in the Selected list will not be affected (ie. the model in the Selected list can still be loaded and visualized as usual).

For the following section, this example will be considered, where forestfire@FireLOG.log and Fire.drw have been added to the Available list, and Fire.drw has been selected (ie. to the Selected list).



Figure 237:Selecting an from an available list

8.16.5.2 Loading models to be visualized

- 1) To load the atomic cellular models in the Selected list, left-click the Load Model button. In this example, only Fire.drw is in the Selected list, so only Fire.drw will be loaded.



Figure 238: Loading an atomic cellular model

Note: After pressing the Load Model button, please wait until all selected models have been loaded to the visualization cell display area. The number of models in the Selected list corresponds to the number of models to be loaded and visualized. In general, the waiting time required to load the models will increase with the number of models in the Selected list.

- 2) The atomic cellular models in the Selected list will be loaded into the visualization cell display area.

In this example, after Fire.drw is loaded, the cell values of the Fire model are displayed, as seen below.

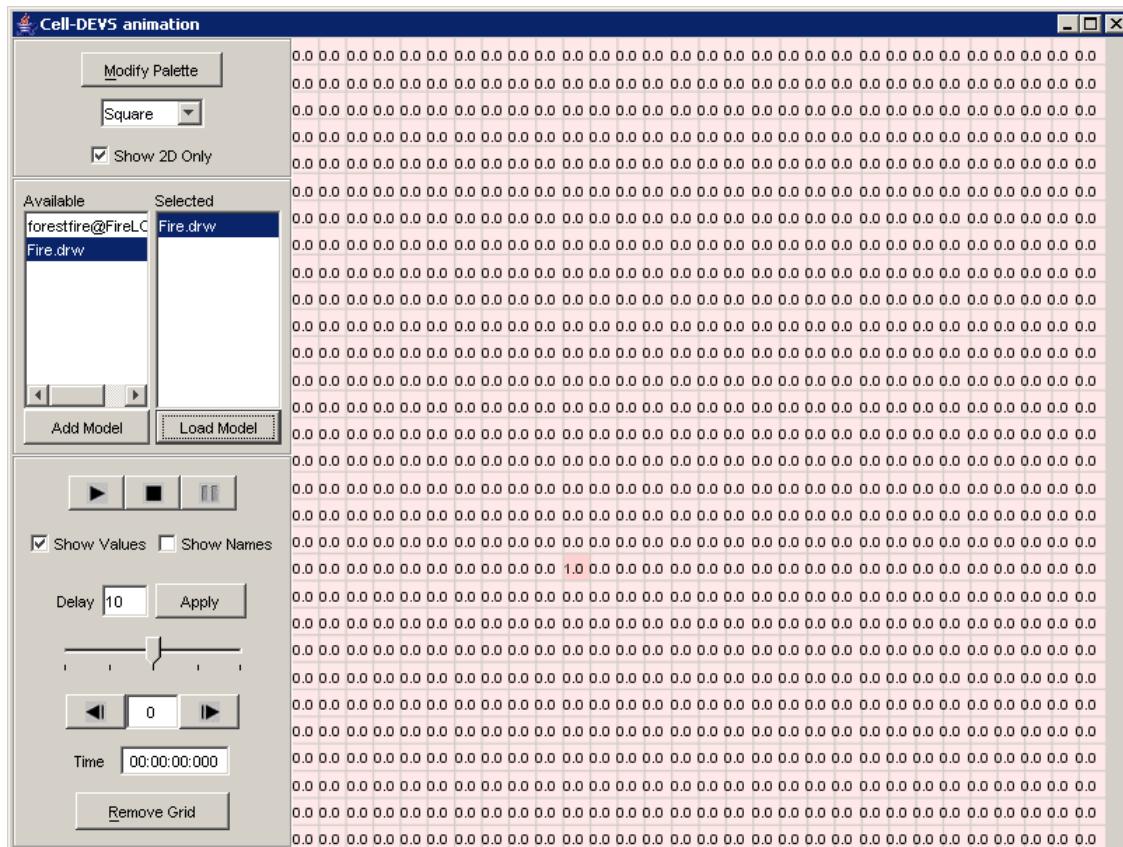


Figure 239:Displayed cell values for fire.drw

The size of the Cell-DEVS animation visualization window can be changed to fit the dimensions of the visualization cell display area.

Some of the options for modifying the appearance of the visualization are automatically set by default.

If a palette (.pal) file exists with the same name as the first model in the Selected list, then the .pal file will automatically be applied to the visualization. In this example, Fire.pal is automatically used for the visualization of Fire.drw. Also, by default: a square lattice is selected; only two dimensions are shown; the cell values are shown; the names of the loaded models are not shown; and the grid is visible.

Modifications to the appearance of the visualization will be described in the following sections.

8.16.5.3 Modifying the appearance of the visualization

After the appropriate atomic cellular models have been loaded, the appearance of the visualization can be modified before the visualization starts. (The appearance of the visualization can also be modified while the visualization is running.) Several options are available for changing the appearance of the visualization:

The palette (or colour scheme) of the loaded model(s) can be modified.

The lattice type can be selected as square, hexagonal, or triangular.

The display of the loaded model(s) can be multi-dimensional or restricted to two dimensions.

The cell values of the loaded model(s) can be shown.

The names of the loaded model(s) can be shown.

The grid of the lattice can be removed/added.

These options are described in the following sections.

Modify the palette of the loaded model(s)

A palette file contains preset colour settings for the visualization of a model. Each setting contains a colour that corresponds to a range of values. When a cell value falls within the range of values of a setting, the colour of the setting will be applied to the cell.

1) Left-click the Modify Palette button. The Modify Palette dialog will be displayed.

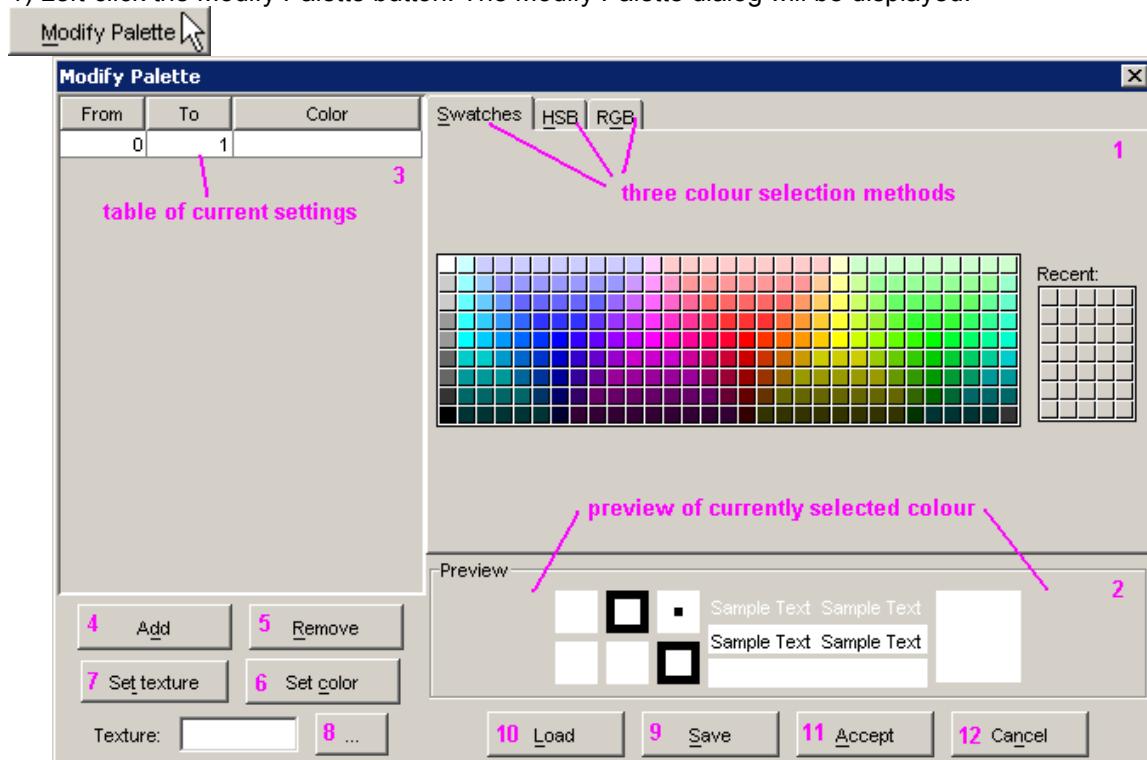


Figure 240: modifying the palette

For modifying the settings of the palette (.pal) file used for the visualization, functions include:

- selecting a colour (1)
- previewing the selected colour (2)
- changing the ranges for the current settings (3)
- adding a setting (4)
- removing a setting (5)
- setting the colour of a setting (6)

- setting the texture of a setting **(7)**
- selecting the texture to be set **(8)**
- saving the current settings in a .pal file **(9)**
- loading settings from an existing .pal file **(10)**
- accepting the current settings for the visualization **(11)**
- cancelling the modify palette operation **(12)**

Select and preview a colour

- 1) On the upper-right panel, left-click on one of the three tabs: Swatches, HSB, RGB.

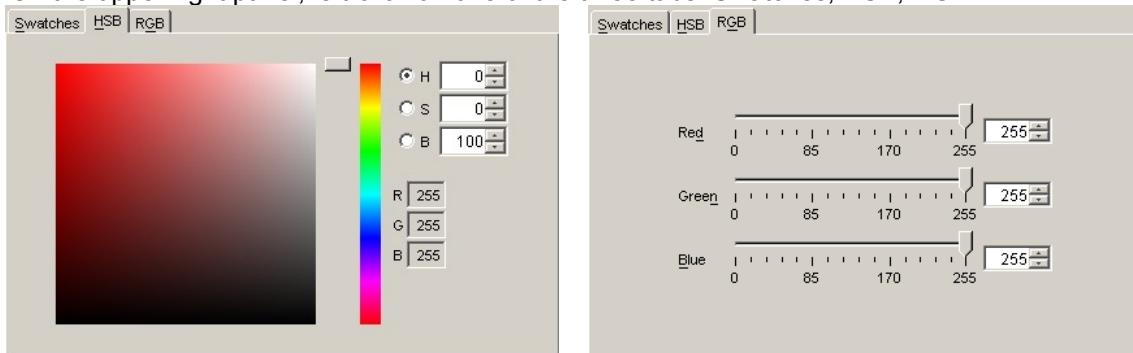


Figure 241: Previewing color

- 2) From one of the three tab panels, select a colour. The selected colour will be displayed in the lower-right Preview panel.

Load settings from an existing .pal file

- 1) To load settings from an existing .pal file, first left-click on the Load button. The Open dialog will be displayed.

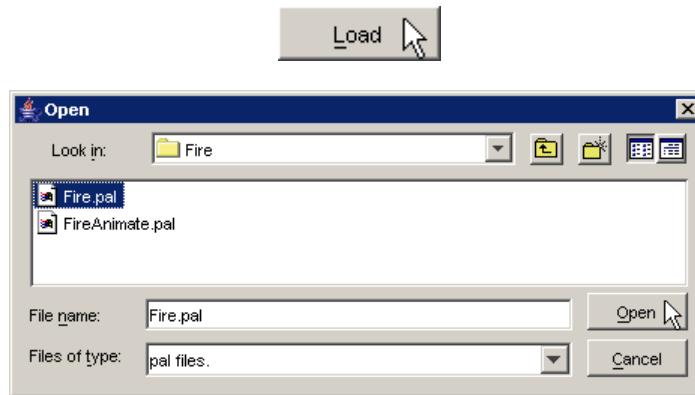


Figure 242: Loading settings from an existing file

- 2) Choose the appropriate .pal file, and left-click the Open button. The settings from the chosen .pal file will be displayed in the left panel, in the table of current settings.

In this example, Fire.pal is chosen for the visualization of Fire.drw. The settings range from -100 to 180.

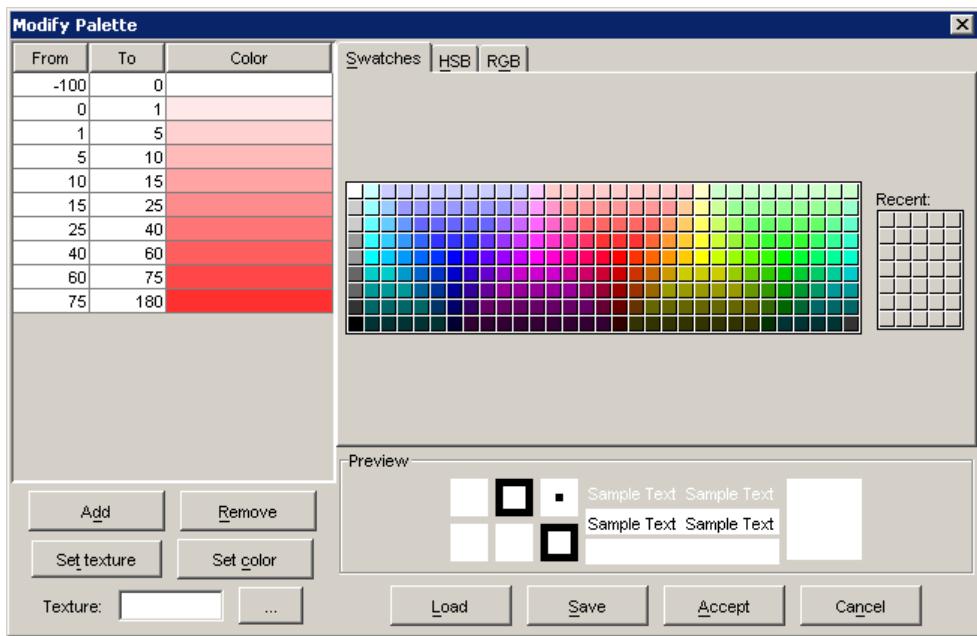


Figure 243: Existing palette

Add a setting

- 1) To add a new setting to the table of current settings, first left-click the Add button.



A new setting will appear at the bottom of the table of current settings.

By default, the new setting has a range [0,0], with white set as the colour corresponding to the range.

In this example, a new setting is added to the bottom of the table of settings loaded from Fire.pal.

From	To	Color
-100	0	
0	1	
1	5	
5	10	
10	15	
15	25	
25	40	
40	60	
60	75	
75	180	
0	0	

Figure 244: Adding a new setting

Change the range of a setting

- 1) To change the range of values for a particular setting, left-click on the From/To field.
2) Type in the appropriate value(s).

In this example, for the most recently added setting, the From field has been changed to 180, while the To field is being changed to 200.

From	To	Color
-100	0	
0	1	
1	5	
5	10	
10	15	
15	25	
25	40	
40	60	
60	75	
75	180	
180	200	

Figure 245: Changing the range of values

Set or change the colour of a setting

- 1) To set or change the colour of a particular setting, first select the appropriate colour from one of the three right tab panels. The selected colour will be placed in the Recent colours list.
In this example, the colour (204,255,255) is selected from the Swatches tab, as seen in the figure below.

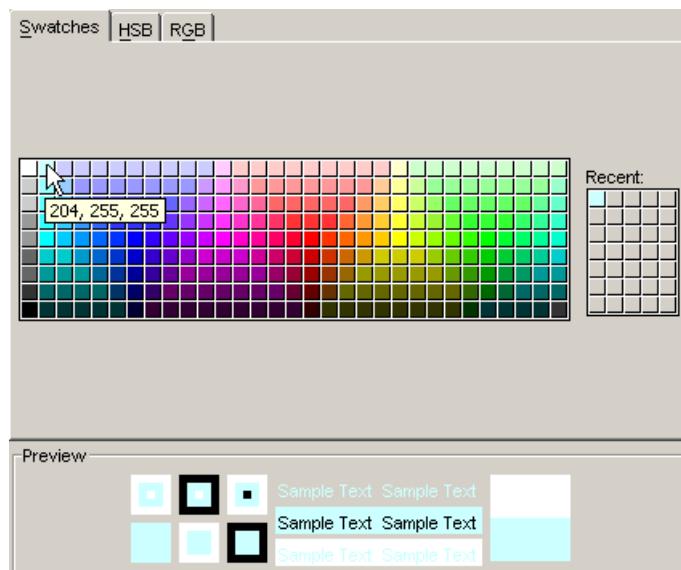


Figure 246: Setting or changing the color of a setting

2) Left-click on the particular setting.

From	To	Color
75	180	Red
180	200	Blue

3) Left-click on the Set color button.



The selected color will be displayed in the setting, alongside the range of values.

In this example, for the most recently added setting, the colour (204,255,255) has been set to correspond with the range [180,200].

From	To	Color
-100	0	
0	1	
1	5	
5	10	
10	15	
15	25	
25	40	
40	60	
60	75	
75	180	Red
180	200	Blue

Figure 247:Setting the color

Set the texture of a setting

1) To set the texture of a particular setting, first left-click the browse button beside the Texture field.



The Open dialog will be displayed.

2) Select the file containing the appropriate texture. Left-click Open. The selected file name will be displayed in the Texture field.

3) Left-click on the particular setting.

4) Left-click the Set texture button.



The selected file name will be displayed in the setting, alongside the range of values.

Remove a setting

1) To remove a particular setting from the table of current settings, first left-click on the particular setting.

From	To	Color
75	180	Red
180	200	Blue

2) Left-click the Remove button.



The selected setting will be removed from the table of current settings.

In this example, for the most recently added setting, with range [180,200] and colour (204,255,255), was removed from the table of current settings.

From	To	Color
-100	0	
0	1	
1	5	
5	10	
10	15	
15	25	
25	40	
40	60	
60	75	
75	180	Red

Figure 248: Removing a setting

Save the current settings in a .pal file

- 1) To save the current settings (displayed in the table in the left panel) in a .pal file, first left-click the Save button. The Save dialog will be displayed.

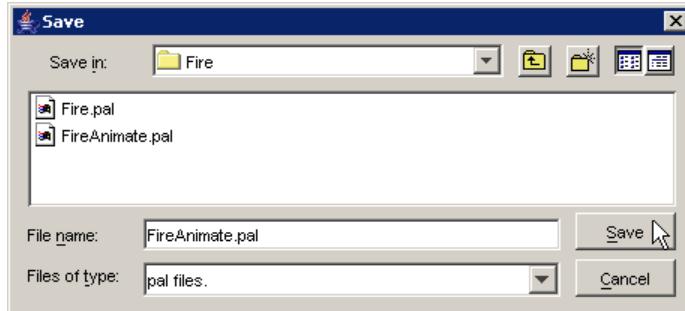


Figure 249: ***Saving the current settings in a .pal file***

- 2) Specify the appropriate existing or new .pal file name, and left-click the Save button. The current settings will be saved to the specified .pal file. (The specified .pal file can be loaded using the same procedure as any existing .pal file.) The current (saved) settings will remain displayed in the left panel, in the table of current settings.

In this example, the current settings will be saved in FireAnimate.pal.

Accept the current settings for the visualization

- 1) To accept and apply the current settings (displayed in the table in the left panel) to the visualization, left-click the Accept button. The Modify Palette dialog will close, and the current settings will be applied to the visualization of the loaded model(s).



In this example, the settings for FireAnimate.pal are accepted and applied to the visualization of Fire.drw.

Cancel the modify palette operation

- 1) To cancel the modify palette operation, left-click on the Cancel Button. The Modify Palette dialog will close, and no changes will be applied to the visualization of the loaded model(s).



8.16.5.4 Select the lattice type

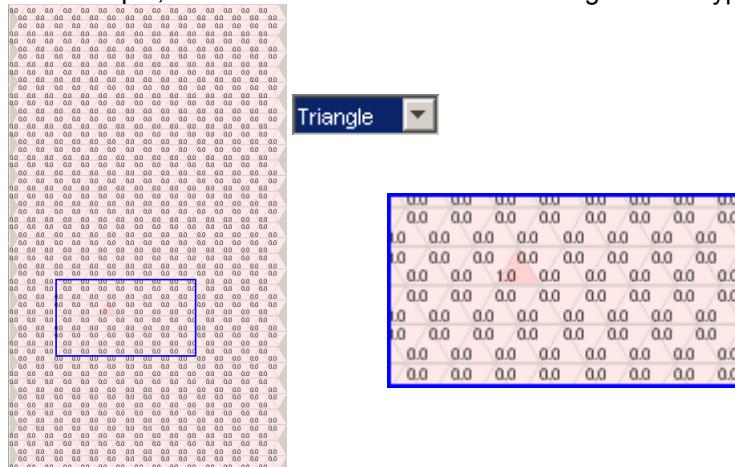
Three lattice types are available: Square, Triangular, and Hexagonal.
By default, a Square lattice is selected.

- 1) Left-click the lattice drop-down list.

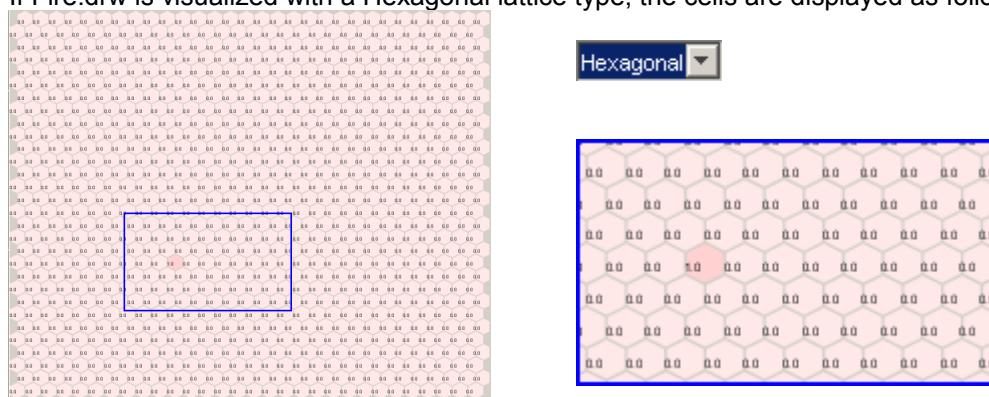


- 2) Select the appropriate lattice type, depending on the lattice type of the loaded model(s).

For example, if Fire.drw is visualized with a Triangle lattice type, the cells are displayed as follows.



If Fire.drw is visualized with a Hexagonal lattice type, the cells are displayed as follows.



For this example, Fire.drw should be visualized with a Square lattice type.

Note: If the selected lattice type does not match the lattice type of the loaded model(s) (eg. if a hexagonal or triangular lattice is used for a square-latticed model), the results will not be visualized as intended.

8.16.5.5 Show two-dimensional or multi-dimensional display

- For visualization of 1D or 2D models, this option is ignored.
- For visualization of 3D models, if Show 2D Only is checked, only cells with the coordinates $(x,y,0)$ are displayed (ie. only cells in the first plane). If Show 2D Only is not checked, the cells in the planes are displayed from left to right, with the sequence: $(x,y,0)$, $(x,y,1)$, $(x,y,2)$, etc.
- For visualization of multi-dimensional models, if Show 2D Only is checked, only cells in the first plane are displayed, ie. cells with coordinates $(x,y,0,0,\dots,0)$. If Show 2D Only is not checked, the cells in the planes are displayed from left to right, with the sequence: $(x,y,0,0,\dots,0)$, $(x,y,1,0,\dots,0)$, ..., $(x,y,D2,0,\dots,0)$, $(x,y,0,1,\dots,0)$, ..., $(x,y,D2,D3,\dots,DN)$.

By default, Show 2D Only is checked.

1) To restrict the visualization of the loaded model(s) to only two dimensions, check Show 2D only.

 Show 2D Only

2) To show all the dimensions of the loaded model(s) in the visualization, uncheck Show 2D Only.

 Show 2D Only

In this example, Fire.drw represents a two-dimensional model, so the Show 2D Only option is ignored. (See the example for a 3D atomic Cell-DEVS model, SatelliteClouds, for more details about the Show 2D Only option.)

Show cell values of the loaded model(s)

By default, Show Values is checked.

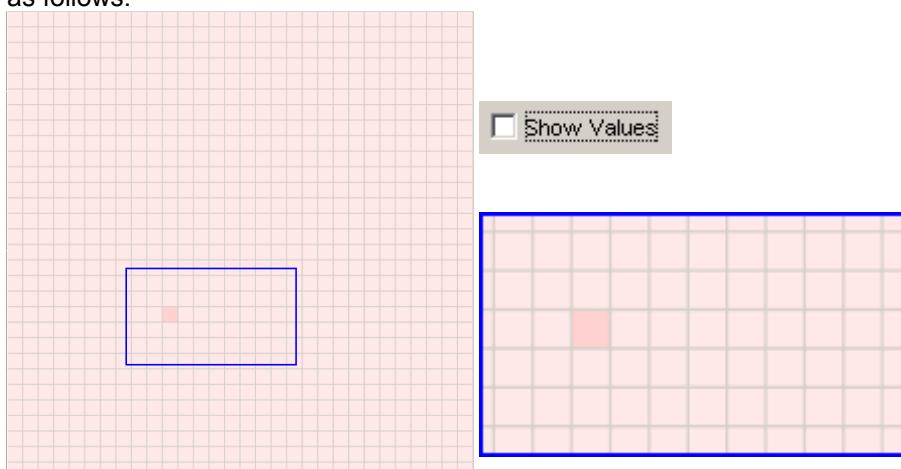
1) To show the cell values of the loaded model(s) in the visualization, check Show Values.

 Show Values

2) To remove the cell values of the loaded model(s) in the visualization, uncheck Show Values.

 Show Values

In this example, if Show Values is unchecked for the visualization of Fire.drw, the cells are displayed as follows:

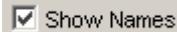


8.16.5.6 Show names of the loaded model(s)

For models of the form <componentName>@<modelLog.log>, this option is ignored.
This option is only available for models of the form <modelDrw.drw>.

By default, Show Names is not checked.

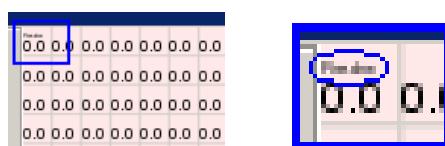
- 1) To show the names of the loaded model(s) in the visualization, check Show Names.



- 2) To remove the names of the loaded model(s) in the visualization, uncheck Show Names.



In this example, if Show Names is checked for the visualization of Fire.drw, the name 'Fire.drw' is displayed in the upper-left corner of the display, as follows:



Note: If Show Names is checked for a model of the form <componentName>@<modelLog.log>, the name of the model will not be displayed.

Remove/add the grid of the lattice

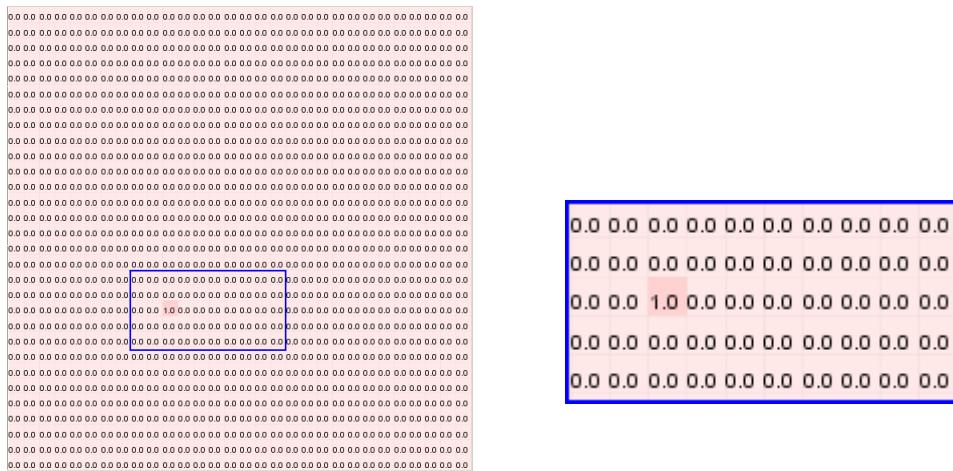
By default, the grid of the lattice is visible.

- 1) To remove the grid of the lattice from the visualization, left-click Remove Grid. The Remove Grid button will be replaced by the Add Grid button.



- 2) To add the grid of the lattice to the visualization, left-click Add Grid. The Add Grid button will be replaced by the Remove Grid button.

In this example, if Remove Grid is pressed, the visualization for Fire.drw will be displayed as follows:



8.16.5.7 Running/Navigating the visualization

Once the appearance of the visualization has been modified as desired, the visualization can be run/navigated. The individual steps in the visualization can be navigated to directly or indirectly. Navigating indirectly involves playing/starting, stopping, and pausing the visualization. Navigating directly includes going to the previous/next step in the visualization, and going directly to a particular step or time in the visualization. Running the visualization also requires setting the delay between the steps of the visualization. These functions are described in the following sections.

Set the delay between steps

In general, the larger the delay value, the longer the amount of time between each step during the visualization. By default, the delay is set at 10. ([units?](#))

- 1) Type the appropriate delay length in the Delay field. Left-click Apply.

In this example, the delay has been set to 250.



The delay can be set while the visualization is running (ie. after the visualization has been started), after the visualization has been stopped, or while the visualization is paused (ie. after the visualization has been paused).

Play/start the visualization

- 1) To start the visualization, left-click the start button. The visualization will start at the beginning of the simulation (ie. the beginning of the .log file).



The graphical results will be updated and displayed sequentially, along with the corresponding display step and time (according to the message times of the .log file).

In this example, the visualization of Fire.drw has been started, and has the current display time of 00:04:47:892 and current step of 6, with the corresponding cell values visible in the display area.

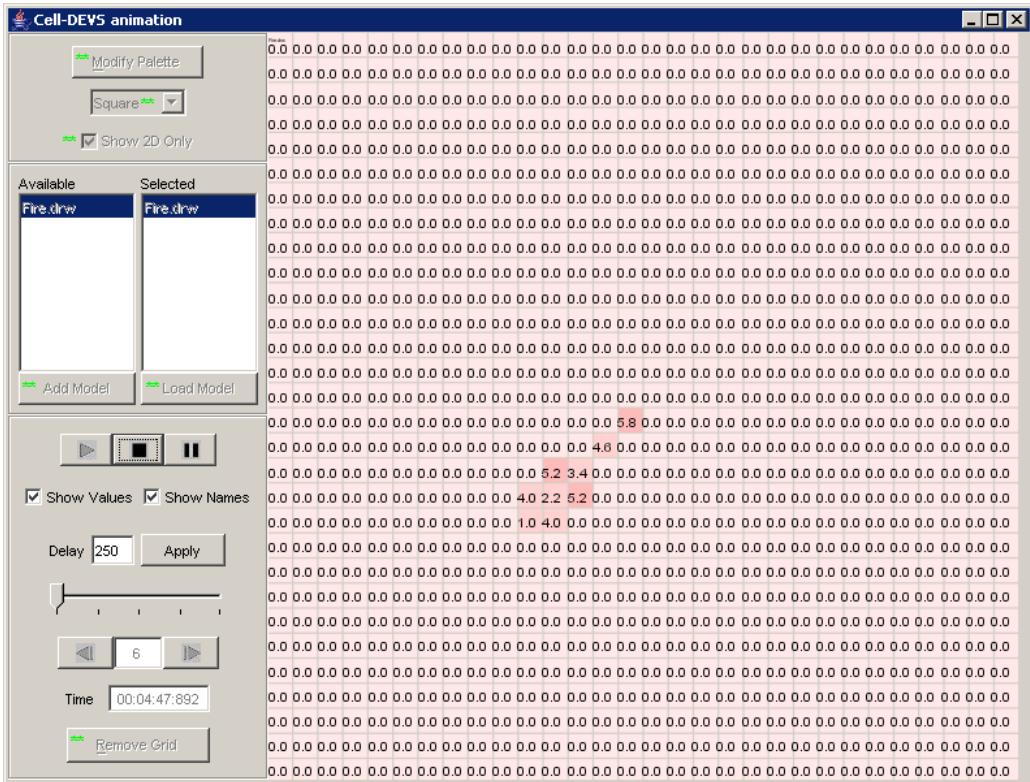


Figure 250: Play/Start Visualization

** Note: While the visualization is running (ie. After the visualization has been started), the following appearance options cannot be changed: Modify Palette, Select lattice type, Show 2D Only, Remove/Add Grid. Also, models cannot be added-loaded while the visualization is running.

Note: In general, to allow the visualization to update the cell display area more quickly, uncheck Show Values.

2) After the visualization has started, it can be run to the end of the simulation time, stopped, or paused.

 After the visualization has started, if no other buttons are pressed, the visualization will continue until it reaches the end of the simulation (ie. The last message time in the .log file).

Note: Once the end of the simulation has been reached, in order to replay the visualization, the stop button must be pressed.

In this example, after pressing the Start button, if no other buttons are pressed, the visualization will continue until it reaches 01:59:40:578 (ie. Step 386), which is when the simulation ends in the .log file.

 The stop and pause buttons are described in the following sections.

Stop the visualization

- 1) To stop the visualization, left-click the stop button. The visualization will stop at the current display time.



The cell values occurring at the current display time will not be visible; instead, the cell values occurring at the beginning of the simulation will be visible.

In this example, the visualization of Fire.drw is about to be stopped at the current step of 7 and current display time of 00:05:22:995. The cell values visible in the display area correspond to the current display time.

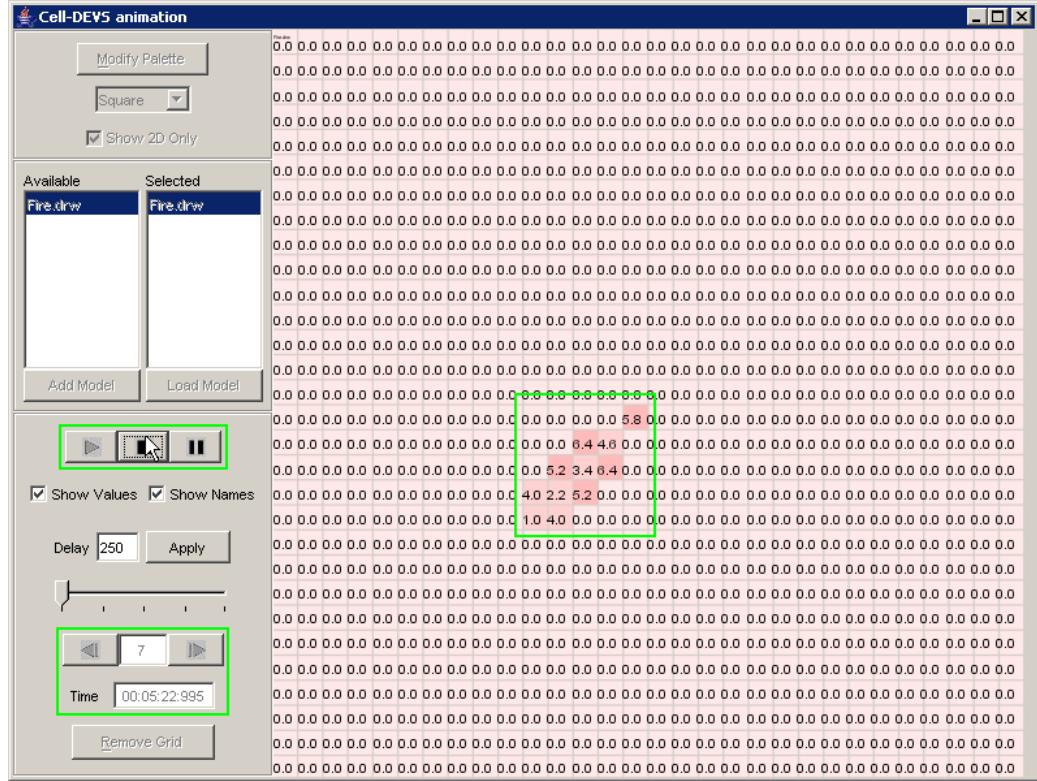


Figure 251: Visualization Stopped

In the following figure, the visualization has been stopped. The display time and step, as well as the cell values visible in the display area, correspond to the beginning of the simulation (ie. at 00:00:00:000).

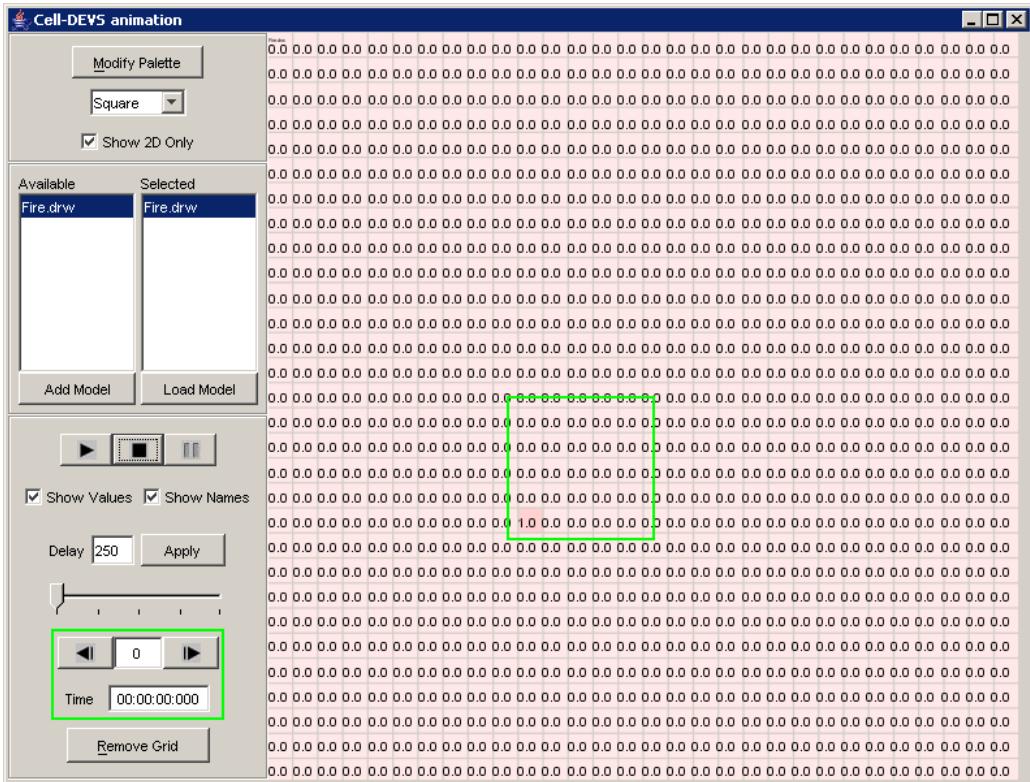


Figure 252: Values displayed on stopped visualization

2) After the visualization has stopped, it can be restarted (at the beginning of the simulation).

- After the visualization has stopped, if the start button is pressed, the visualization will restart only at the beginning of the simulation, ie. at 00:00:00:000. The visualization will not restart at time at which the visualization was stopped.
- The start button is described in the preceding section.

8.16.5.8 Pause the visualization

- 1) To pause the visualization, left-click the pause button. The visualization will pause at the current display time. The cell values occurring at the current display time will remain visible on the graph.



In this example, the visualization of Fire.drw has been paused at the current step of 16 and current display time of 00:08:57:147. The cell values visible in the display area correspond to the current display time.

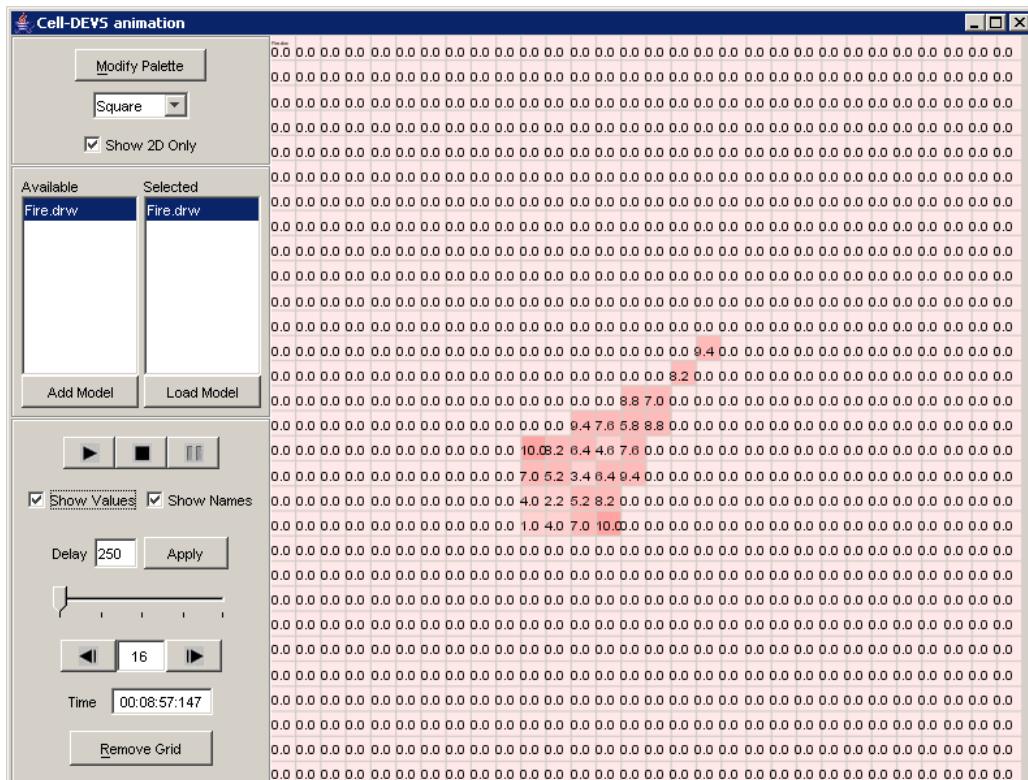


Figure 253: : Cell values corresponding to the current display time

- 2) After the visualization has paused, it can be restarted (at the current display time), stopped, or decremented/incremented to the previous/next step.

- After the visualization has paused, if the start button is pressed, the visualization will restart at the current display time.
- The start and stop buttons are described in the preceding sections.
- The previous and next step buttons are described in the following sections.

8.16.5.9 Go directly to a particular time

This function is not valid while the visualization is running.

- 1) To go directly to a particular time of the visualization, type the particular time in the Time (current display time) field. Ensure the format of the time corresponds to **##:##:##:###**. Press Enter (on the keyboard).



The visualization will display the particular time, corresponding step, and corresponding cell values in the display area.

Note: If a particular time does not occur in the .log file, then the results (Time, step, and cell values) corresponding to the next smaller time in the .log file will be displayed.

Go directly to a particular step

This function is not valid while the visualization is running.

- 1) To go directly to a particular step of the visualization, type the particular step number in the current step field. Press Enter (on the keyboard).



The visualization will display the particular step, corresponding time, and corresponding cell values in the display area.

In this example, step 25 of the Fire.drw visualization has been accessed directly, with corresponding display time 00:11:22:860.

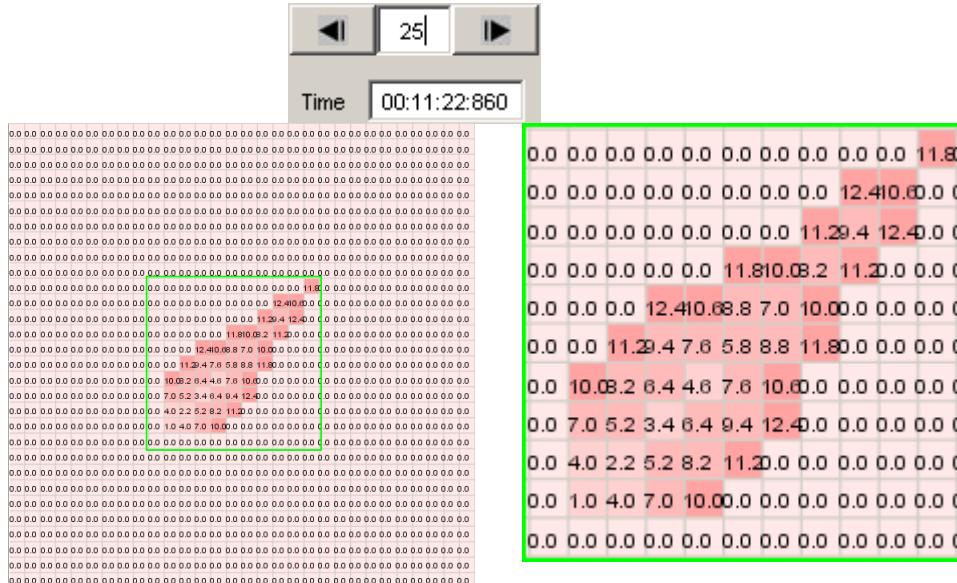


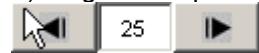
Figure 254: Going directly to particular step

Note: If a step number is larger than the current step *and* larger than the largest step of the visualization, then the results (Time, step, and cell values) corresponding to the current step will remain displayed.

Go to the previous/next step

This function is not valid while the visualization is running.

- 1) To go to the previous step in the visualization, left-click the previous button.



The visualization will decrement to the previous step, updating the display time (according to the .log file) and the cell values in the display area.

This example continues from the previous section - 'Go directly to a particular step'. In this example, step 24 (the previous step in the visualization of Fire.drw) has been accessed directly. The visualization is decremented to the previous display time of 00:11:21:093.



Figure 255: Visualization at 00:11:21:093

Note: If the visualization has reached the smallest step/time of the simulation, then the visualization will not decrement.

2) To go to the next step in the visualization, left-click the next button.



The visualization will increment to the next step, updating the display time (according to the .log file) and the cell values in the display area.

This example continues from the previous section - 'Go directly to a particular step'. In this example, step 26 (the next step in the visualization of Fire.drw) has been accessed directly. The visualization is incremented to the next display time of 00:11:30:766.



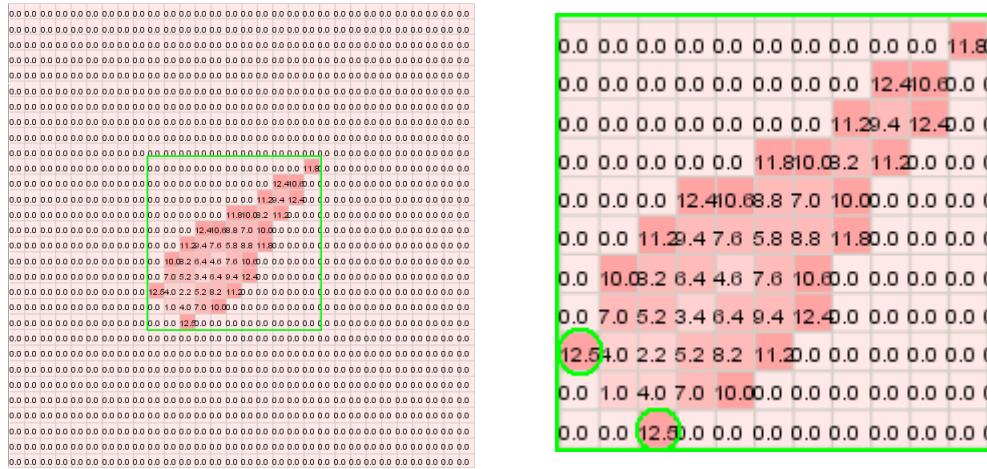


Figure 256: Visualization at 00:11:30:766

Note: If the visualization has reached the largest step/time of the simulation, then the visualization will not increment.

3) After the visualization has decremented/incremented to the previous/next display, it can be restarted (at the current display time), stopped, or decremented/incremented again to the previous/next display.

- After the visualization has decremented/incremented to the previous/next display, if the Start button is pressed, the visualization will restart at the current display time.
- The Start and Stop buttons are described in the preceding sections.
- The previous/next buttons are described in this section.

8.16.6 Cell-DEVS animation Example of multiple Atomic Cell-DEVS: Fire, SatelliteClouds

To run this example, download the Fire and SatelliteClouds projects.

Fire is a 2D (two-dimensional) atomic Cell-DEVS model.

SatelliteClouds is a 3D (three-dimensional) atomic Cell-DEVS model.

In this example, SatelliteClouds will be used as a 2D atomic Cell-DEVS model.

(SatelliteClouds will be used as a 3D model in the next Cell-DEVS animation example.)

The main functionality of the Cell-DEVS animation visualization window described in the previous example (Fire) also applies for this example. However, while the previous example (Fire) illustrated the visualization of a single atomic Cell-DEVS model, this example (Fire & SatelliteClouds) illustrates the visualization of multiple (unrelated) atomic Cell-DEVS models.

The differences in the functions of the Cell-DEVS animation visualization window when used for displaying multiple atomic Cell-DEVS model will be described.

For loading multiple models to be visualized:

- adding models to the Available list
- selecting multiple available models to be loaded
- loading selected models to be visualized

For running/navigating the visualization:

- play/start the visualization

8.16.6.1 Loading multiple (unrelated) models to be visualized

The multiple (unrelated) atomic models must first be loaded before being visualized.

The appropriate atomic models must first be added to the Available list. From the Available list, the models to be visualized must be selected (ie. to the Selected list). Once the atomic models to be visualized are in the Selected list, the models can then be loaded.

Adding models to the Available list

Using the same procedure as for the previous example (Fire), add the appropriate models to the Available list.

For the following sections, this example will be considered, where satelliteclouds.drw and Fire.drw have been added to the Available list.

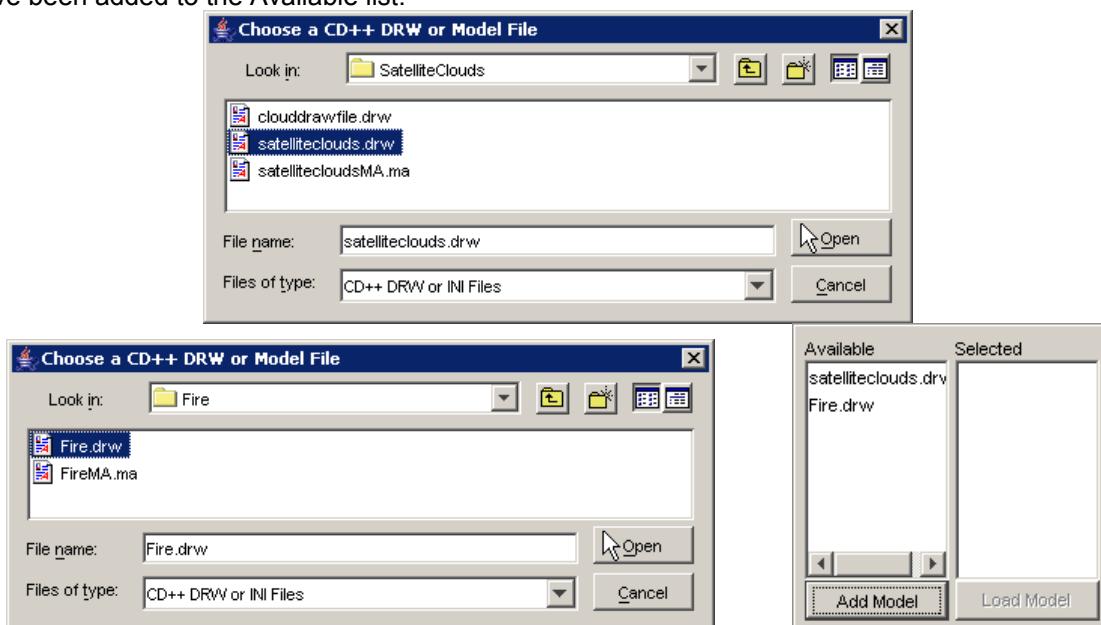


Figure 257: Adding models to available list

Selecting multiple available models to be loaded

Using the same procedure as for the previous example (Fire), from the Available list, select the models to be visualized.

Continuing this example, Fire.drw and satelliteclouds.drw have been selected from the Available list (ie. to the Selected list).

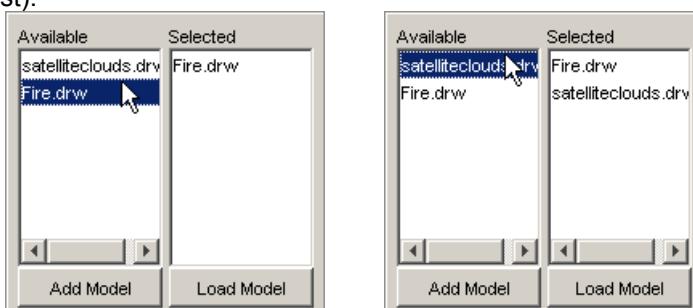


Figure 258:Selecting multiple available models

Note: When loaded, the order of the models in the Selected list influences the appearance of the visualization.

8.16.6.2 Loading selected models to be visualized

Using the same procedure as for the previous example (Fire), load the models in the Selected list.

In this example, Fire.drw and satelliteclouds.drw will be loaded (and visualized) concurrently.

Reminder: Please wait until all selected models have been loaded to the visualization cell display area. Since multiple models are being loaded, the waiting time required to load the models will be greater than if a single model was being loaded.

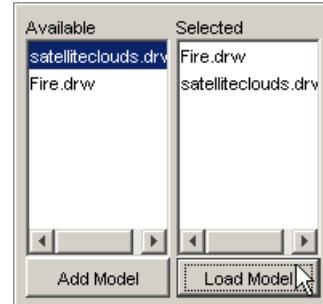


Figure 259: Loading models

The cell values of the Fire model and SatelliteClouds model will be displayed in the visualization cell display area, as seen below.

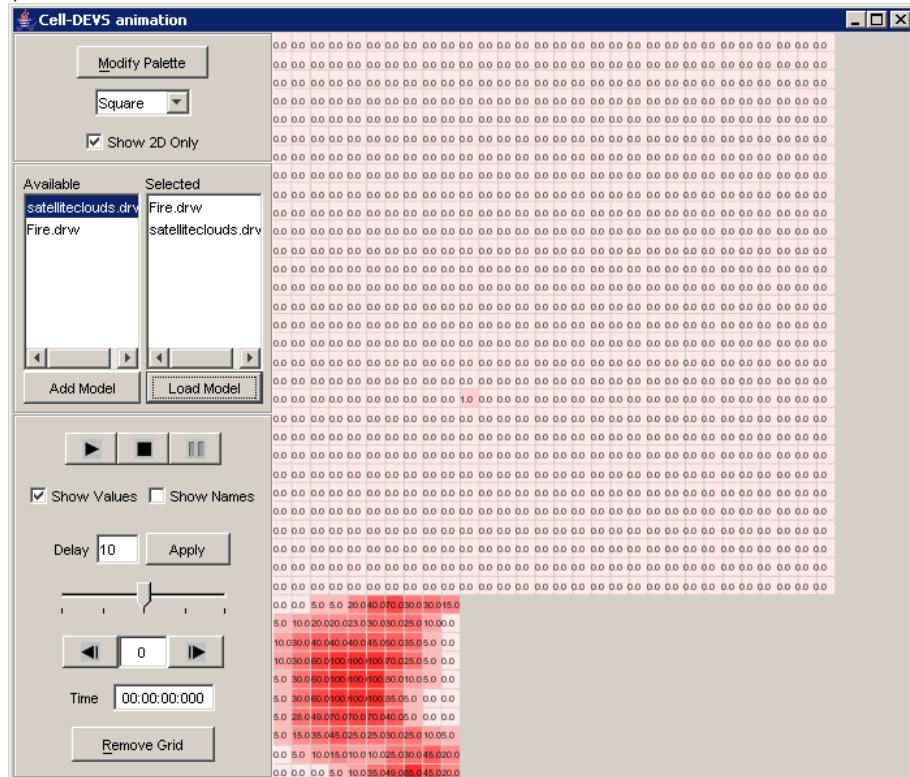


Figure 260: Visualizing loaded models

Reminder: The size of the Cell-DEVS animation visualization window can be increased/decreased to fit the multiple models in the visualization cell display area.

When multiple atomic models are loaded and visualized, the order of the models in the Selected list influences the appearance of the visualization.

- By default, the order of the models in the visualization cell display area corresponds to the order of the models in the Selected list. The first model in the Selected list will be displayed at the top of the cell display area. Each consecutive model in the Selected list will be displayed consecutively below the previous model in the list. (Check Show Names to display the names of the loaded models.)
- By default, if the first model in the Selected list has a palette (.pal) file with the same name as the model (ie. .drw file), then the .pal file will automatically be applied to all models in the visualization. In this example, since Fire.drw is the first model in the Selected list, Fire.pal is automatically used for

the visualization of both Fire.drw and satelliteclouds.drw. (Note: If the first model in the Selected list does not have a .pal file of the same name, then the .pal file of the previously-loaded visualization is used.) Thus, each loaded model in the visualization must use the same .pal file (ie. the .pal file of the first model in the Selected list).

From the previous figure, the order of the Selected list is: Fire.drw, satelliteclouds.drw.

Since Fire.drw is first in the Selected list, by default:

- Fire is located at the top of the cell display area, while SatelliteClouds is located directly below.
- Fire.pal is used for both loaded models.

Conversely, consider when the Selected list order is reversed: satelliteclouds.drw, Fire.drw.

	From	To	Color
Fire.pal settings	-100	0	
	0	1	
	1	5	
	5	10	
	10	15	
	15	25	
	25	40	
	40	60	
	60	75	
	75	180	

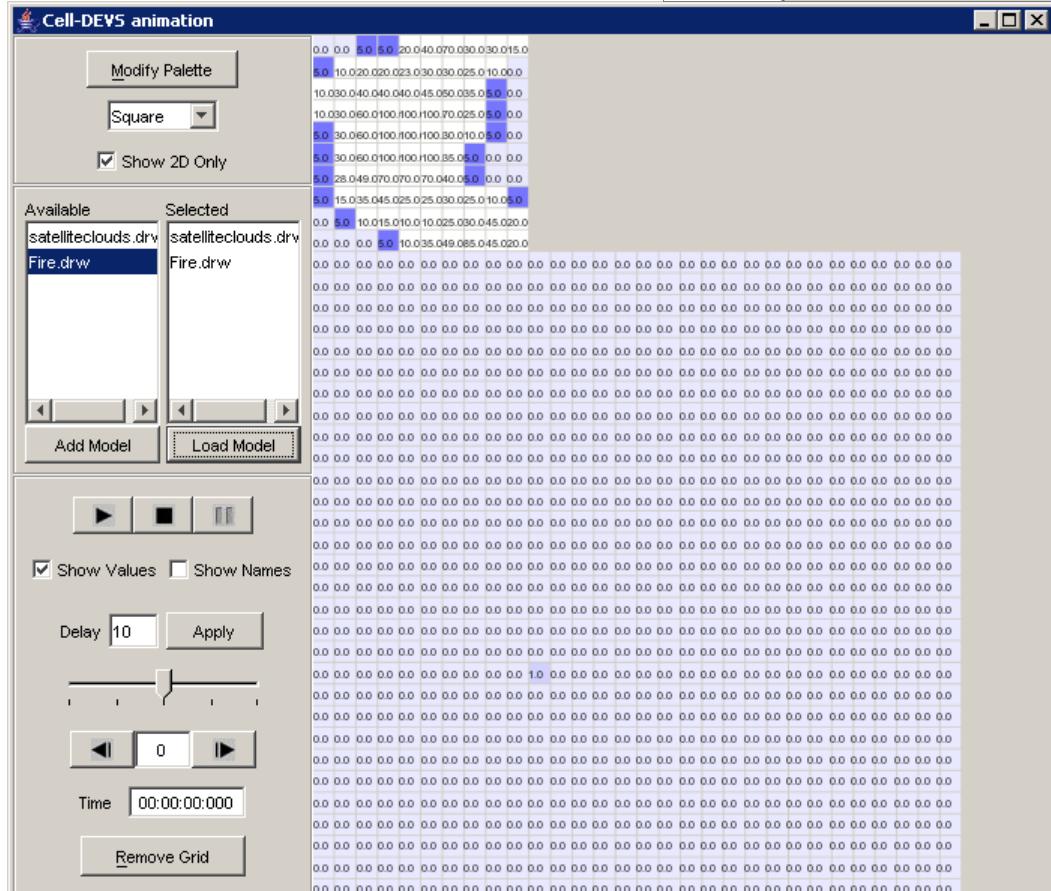


Figure 261: Visualization with list order reversed

Since satelliteclouds.drw is first in the Selected list, by default:

- SatelliteClouds is located at the top of the cell display area, while Fire is located directly below.
- satelliteclouds.pal is used for both loaded models.

	From	To	Color
satelliteclouds.pal settings	9	50	
	0	1	
	1	2	
	2	3	
	3	4	
	4	5	
	5	6	
	6	7	
	7	8	
	8	9	

8.16.6.3 Running/Navigating the visualization

Once the appearance of the visualization has been modified as desired, the visualization of the multiple models can be run/navigated.

Play/start the visualization

Using the same procedure as for the Fire example, the visualization can be started.

*The visualization will start at the earliest message time (of the .log files) of all loaded models.
After the visualization has started, if no other buttons are pressed, the visualization will continue until it reaches the last message time (of the .log files) of all loaded models.
(ie. If the loaded models have different simulation times, the visualization will continue until the end of the longer simulation time.)*

Also, if the loaded models have overlapping simulation times (ie. occur during the same time range), the individual visualizations of the loaded models will run concurrently.

If the loaded models do not have overlapping simulation times, the model of shorter simulation time will remain visible in the cell display area while the visualization of the model of longer simulation time runs.

For the .drw files used in this example:

The simulation of the SatelliteClouds model starts at 00:00:01:000, and ends at 00:00:30:000.

The simulation of the Fire model starts at 00:01:11:973, and ends at 01:59:40:578.

If the SatelliteClouds model was visualized individually, it would start at 00:00:01:000 (ie. step 1) and end at 00:00:30:000 (ie. step 30). (In the accompanying figures, the SatelliteClouds model is displayed with the palette Fire.pal applied.)

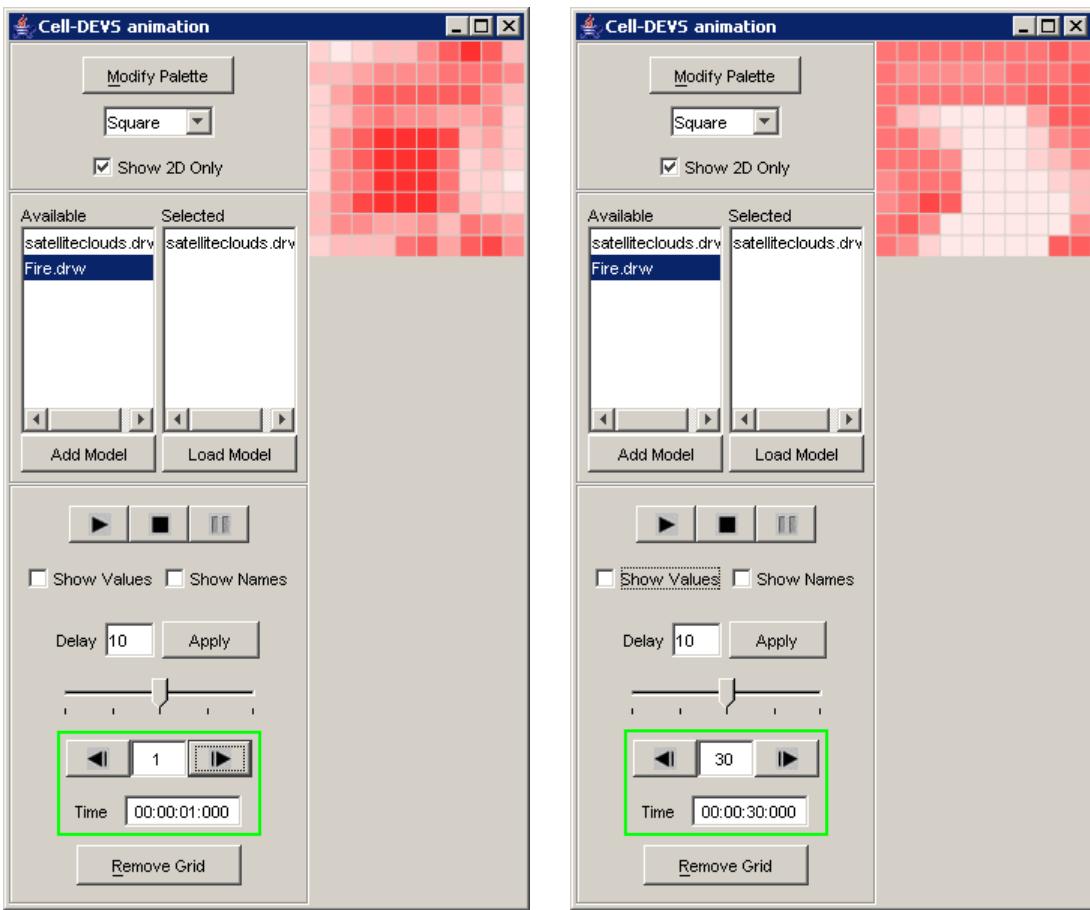


Figure 262: Model displayed with .pal applied

If the Fire model was visualized individually, it would start at 00:01:11:973 (ie. step 1) and end at 01:59:40:578 (ie. step 386).

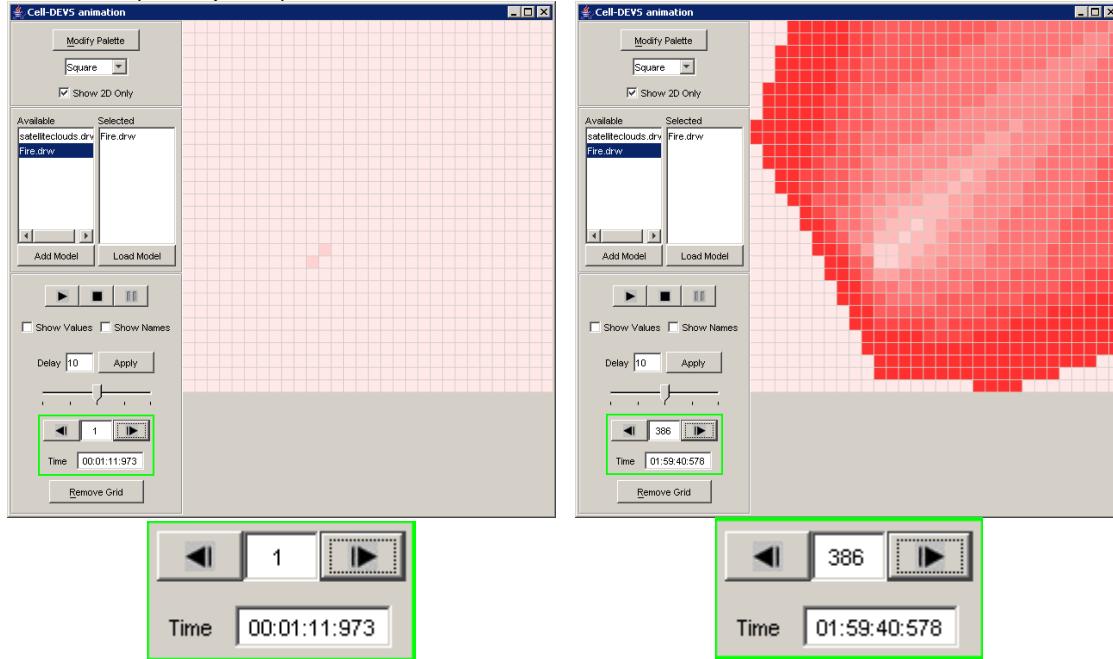


Figure 263: Fire model viewed individually

In this example with both the SatelliteClouds and Fire models loaded, after pressing the Start button, if no other buttons are pressed:

A)- The visualization will start with the SatelliteClouds model, which will run starting at 00:00:01:000 (ie. step 1) until 00:00:30:000 (ie. step 30).

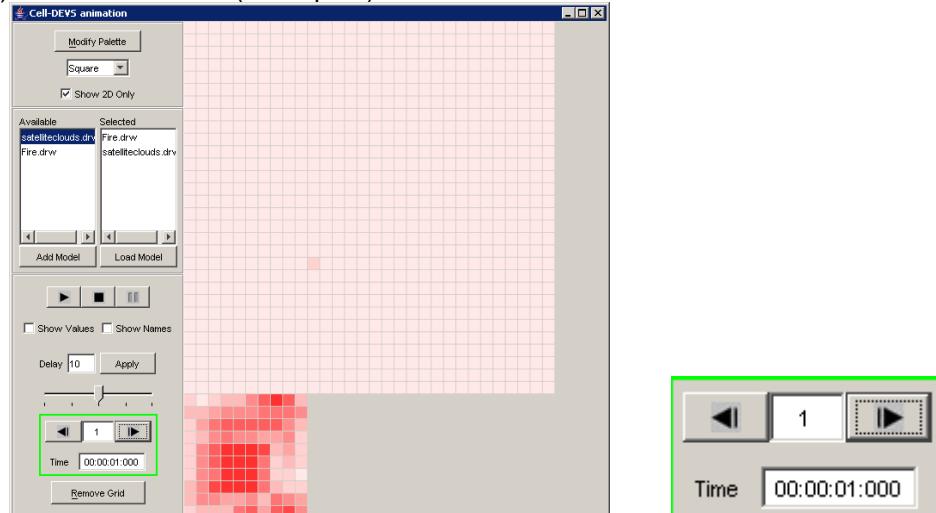


Figure 264: Starting visualization of SatelliteClouds model

B)- Since the message times of the SatelliteClouds and Fire models do not overlap, the visualization of SatelliteClouds will be complete before the visualization of the Fire model starts. After the SatelliteClouds visualization has completed (ie. after step 30), the cell values occurring at step 30 will

remain visible (in the SatelliteClouds cell display area) until the end of the entire visualization.

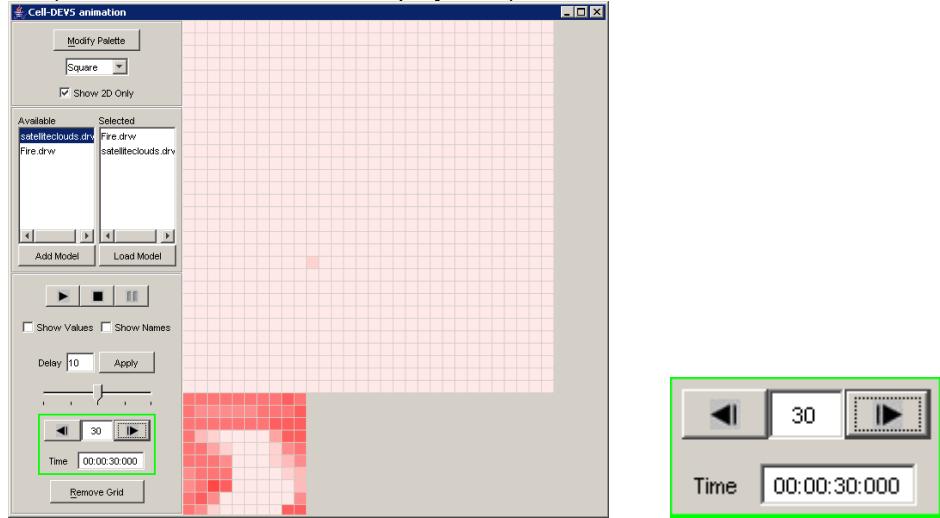


Figure 265: Viewing cell values (step 30)

C)- The visualization of the Fire model will run, starting at 00:01:11:973 (ie. step 31) until 01:59:40:578 (ie. step 416).

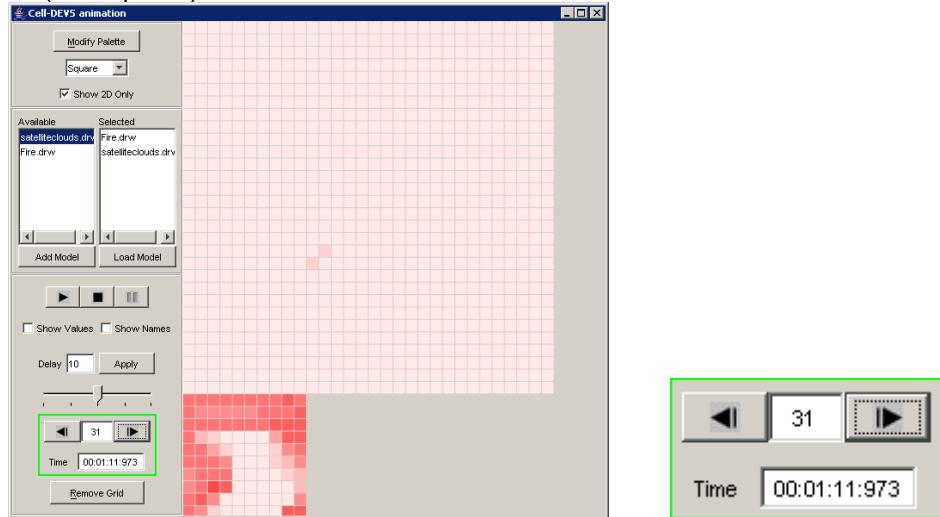


Figure 266: Visualization of fire model (step 31)

D)- The entire visualization will end upon reaching the last message time (ie. at step 416) of the Fire model.

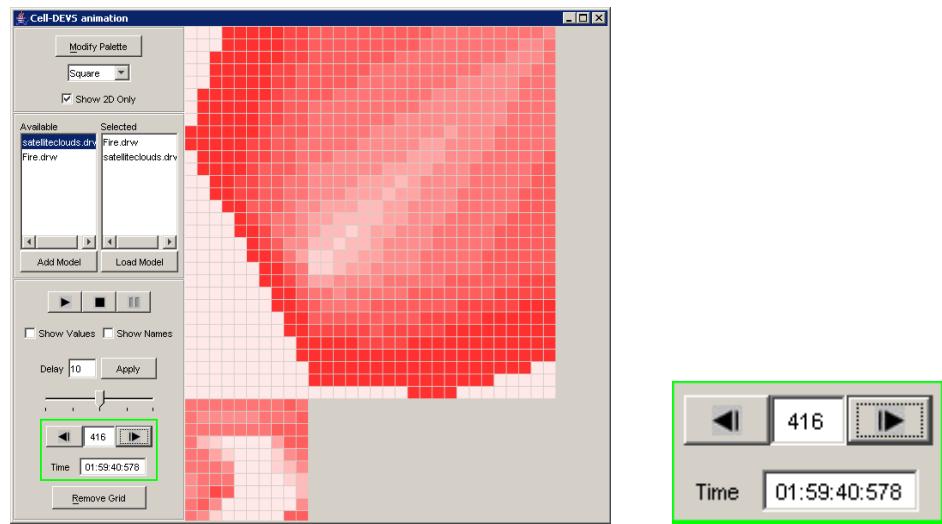


Figure 267: Visualization of Fire model (Step 416)

Cell-DEVS animation Example of 3D Atomic Cell-DEVS: SatelliteClouds

To run this example, download the SatelliteClouds project.

SatelliteClouds is a 3D (three-dimensional) atomic Cell-DEVS model.

The main functionality of the Cell-DEVS animation visualization window described in a previous example (Fire) also applies for this example. However, instead of illustrating the visualization of a 2D atomic Cell-DEVS model (as in Fire), this example (SatelliteClouds) illustrates the visualization of a 3D atomic Cell-DEVS model.

The differences in the functions of the Cell-DEVS animation visualization window when used for displaying a 3D atomic Cell-DEVS model will be described:

For modifying the appearance of the visualization:

-showing multiple dimensions of the loaded models

Show multi-dimensional display of loaded models

- For visualization of 3D models, if Show 2D Only is checked, only cells with the coordinates $(x,y,0)$ are displayed (ie. only cells in the first plane). If Show 2D Only is not checked, the cells in the planes are displayed from left to right, with the sequence: $(x,y,0), (x,y,1), (x,y,2)$.
- For visualization of multi-dimensional models, if Show 2D Only is checked, only cells in the first plane are displayed, ie. cells with coordinates $(x,y,0,0,\dots,0)$. If Show 2D Only is not checked, the cells in the planes are displayed from left to right, with the sequence: $(x,y,0,0,\dots,0), (x,y,1,0,\dots,0), \dots, (x,y,D2,0,\dots,0), (x,y,0,1,\dots,0), \dots, (x,y,D2,D3,\dots,DN)$.

By default, Show 2D Only is checked.

In this example, the 3D atomic model SatelliteClouds is displayed as a 2D model when loaded. (In the accompanying figure, the SatelliteClouds model is displayed with Show Values unchecked.)

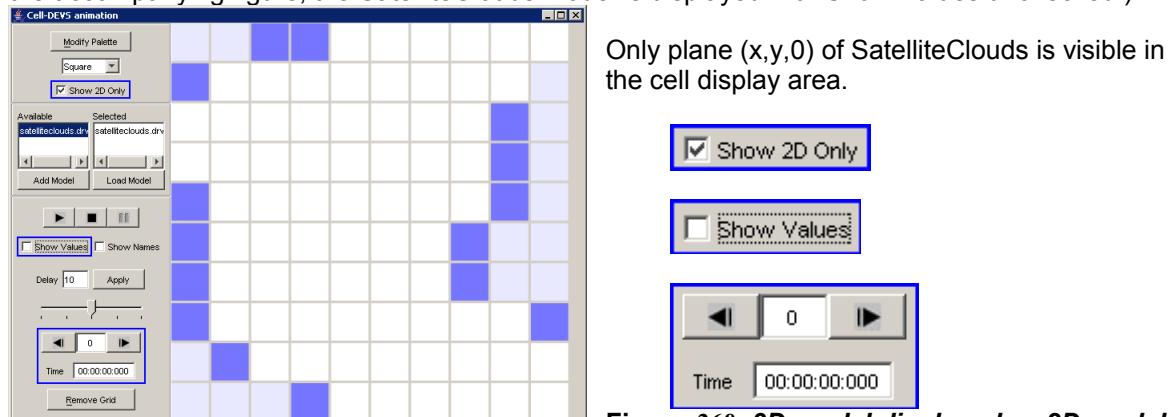


Figure 268: 3D model displayed as 2D model

After unchecking Show 2D Only, SatelliteClouds is displayed as a 3D model.

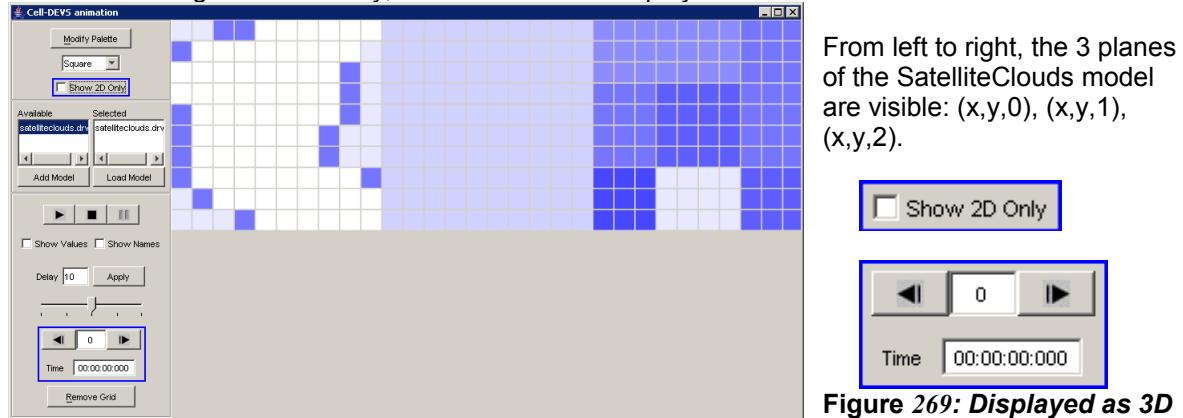


Figure 269: Displayed as 3D Model

8.16.7 Cell-DEVS animation Example of Coupled-DEVS: RiceField

To run this example, download the RiceField project.

RiceField is a coupled-DEVS model composed of multiple atomic Cell-DEVS models. The two component atomic Cell-DEVS models of RiceField are: diffusion, field.

The main functionality of the Cell-DEVS animation visualization window described in a previous example (Fire & SatelliteClouds) also applies for this example. However, instead of illustrating the visualization of multiple unrelated atomic Cell-DEVS models (as in Fire & SatelliteClouds), this example (RiceField) illustrates the visualization of multiple related atomic Cell-DEVS models.

The differences in the functions of the Cell-DEVS animation visualization window when used for displaying multiple related atomic Cell-DEVS model will be described:

For loading multiple models to be visualized:

- adding a coupled model to the Available list
- loading selected models to be visualized

For modifying the appearance of the visualization:

-modify the palette of the loaded models: load settings from an existing .pal file
(Running/Navigating the visualization will also be described.)

8.16.7.1 Loading multiple models to be visualized

The components of a coupled model must first be loaded before being visualized.

The appropriate coupled model must first be added to the Available list. From the Available list, the component models to be visualized must be selected (ie. to the Selected list). Once the component models to be visualized are in the Selected list, the component models can then be loaded.

Adding a coupled model to the Available list

To add a coupled models (consisting of multiple atomic cellular models) to the Available list, the same procedure as for the Fire & SatelliteClouds example is used.

As described in the Fire example, two options are available: (a) choose a .ma file and a .log file, or (b) choose .drw files.

(a) Choosing a coupled model's .ma file (and corresponding .log file) will add all the component models (of the coupled model, as specified in the component: parameter of the .ma file) to the Available list. All the component (ie. atomic cellular) models will use the same .log file.
Thus, all atomic cellular models of a coupled model can be made available by specifying an .ma/.log file (corresponding to the coupled model).

In this example, for the coupled model RiceField, rice.ma is chosen in conjunction with rice.log.

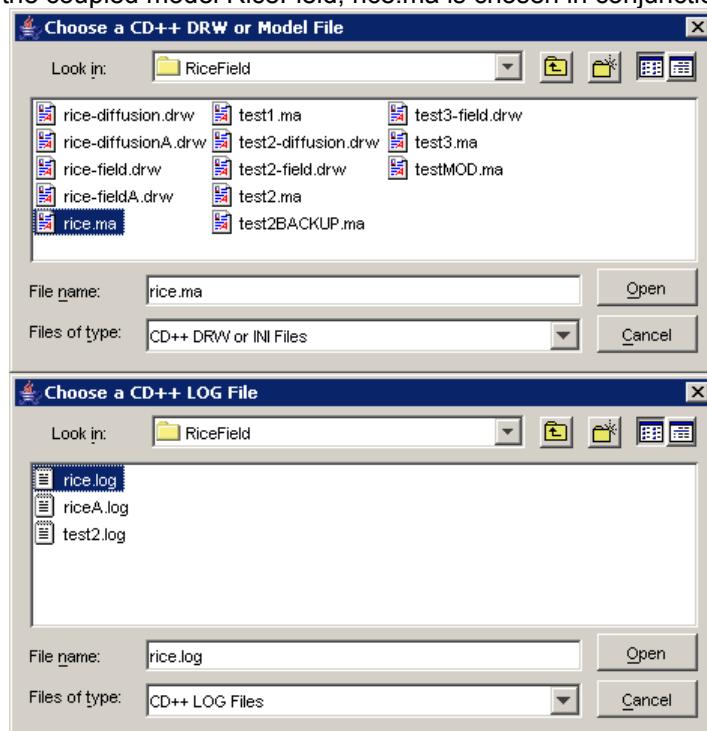


Figure 270: Choosing the rice model

Since the RiceField model consists of two components (ie. diffusion and field), both component models will appear in the Available list with the same .log file.

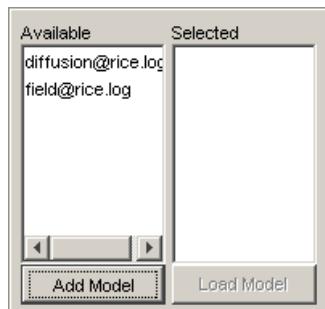


Figure 271: rice.log appears in the list

(b) Choosing a .drw file will add one component model (of the coupled model) to the Available list. (ie. Each .drw file corresponds to one atomic cellular model.) When using .drw files, each component's .drw file must be individually added to the Available list. Thus, only one atomic cellular model can be made available by specifying a .drw file (corresponding to one component of the coupled model).

In this example, for the diffusion component of the RiceField model, rice-diffusion.drw is chosen, and appears in the Available list.

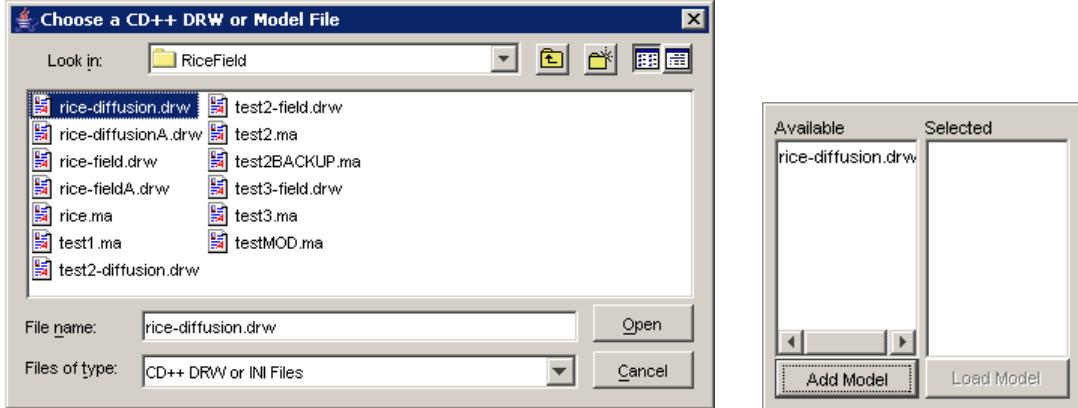


Figure 272: Choosing the diffusion component of the rice model

Also, for the field component of the RiceField model, rice-field.drw is chosen, and appears in the Available list after rice-diffusion.drw.

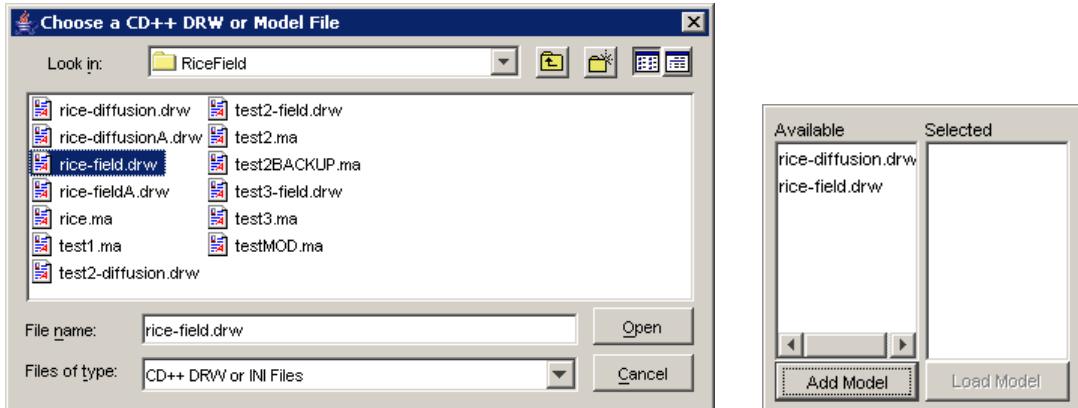


Figure 273: Choosing the field component of the rice model

For the following sections, this example will be considered, where rice-diffusion.drw and rice-field.drw have been added to the Available list.

The same procedure as for the Fire & SatelliteClouds example can be used for selecting multiple available models to be loaded.

Continuing this example, both rice-diffusion.drw and rice-field.drw have been selected from the Available list (ie. to the Selected list).

Reminder: When loaded, the order of the models in the Selected list influences the appearance of the visualization.

8.16.7.2 Loading selected models to be visualized

Continuing this example, rice-diffusion.drw and rice-field.drw will be loaded (and visualized) concurrently.

Reminder: Please wait until all selected models have been loaded to the visualization cell display area. Since multiple models are being loaded, the waiting time required to load the models will be greater than if a single model was being loaded.

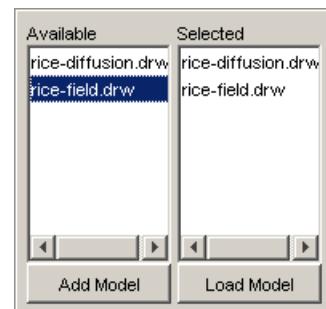


Figure 274: Loading files

The cell values of the selected component models (ie. diffusion and field) of the RiceField model will be displayed in the visualization cell display area, as seen below.

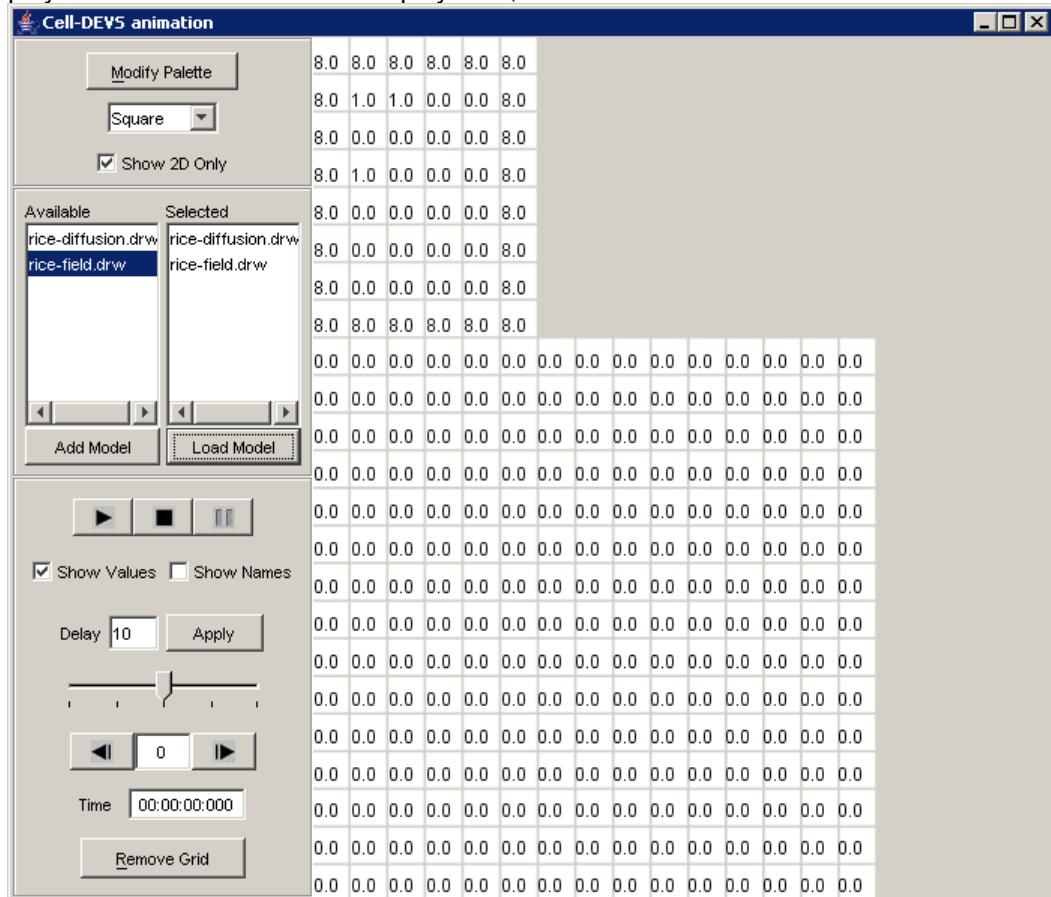


Figure 275: Cell Values displayed concurrently

Reminder: The size of the Cell-DEVS animation visualization window can be increased/decreased to fit the multiple models in the visualization cell display area.

Reminder: By default, if a palette (.pal) file exists with the same name as the first model in the Selected list, then the .pal file will automatically be applied to all models in the visualization.

From the previous figure, since rice-diffusion.drw is first in the Selected list, rice-diffusion.pal would be used by default. However, since rice-diffusion.pal does not exist, no palette is applied by default.

Conversely, consider when the Selected list order is reversed: rice-field.drw, rice-diffusion.drw.

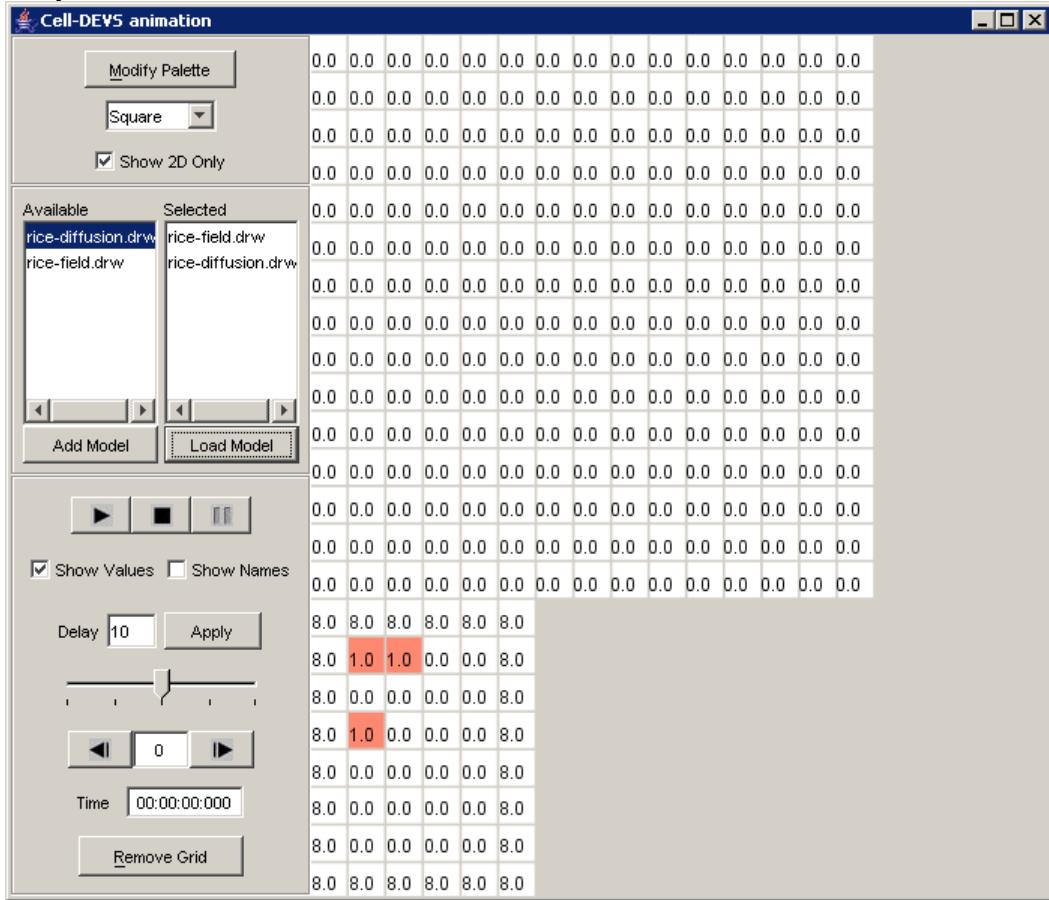


Figure 276: Concurrent display with reversed list order

From the previous figure, since rice-field.drw is first in the Selected list, field.pal would be used by default. So, field.pal is applied to both the field and diffusion components in the visualization.

field.pal settings	From	To	Color
	0	0.25	
	0.25	0.5	
	0.5	0.75	
	0.75	1	
	1	1.25	
	1.25	1.5	
	1.5	1.75	
	1.75	2	
	2	2.25	
	2.25	2.5	

8.16.7.3 Modifying the appearance of the visualization

The appearance of the visualization for a coupled model can be modified using standard procedures, as described in previous examples.

Modify the palette of the loaded models

For the RiceField model, the .pal corresponding to the diffusion component is named ditch.pal.

Load settings from an existing .pal file

Continuing with the example where the order of the Selected list is: rice-diffusion.drw, rice-field.drw. In the Modify Palette dialog, the settings from ditch.pal can be loaded and accepted.

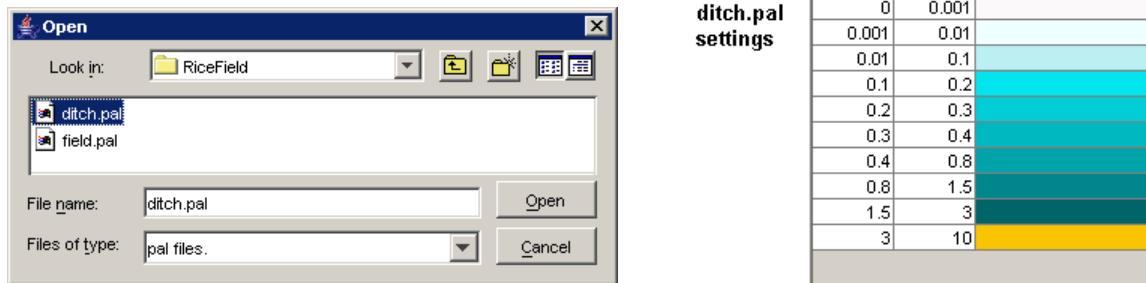


Figure 277: Modifying palettes of the loaded models

The settings of ditch.pal will be applied to both the diffusion and field components in the visualization.

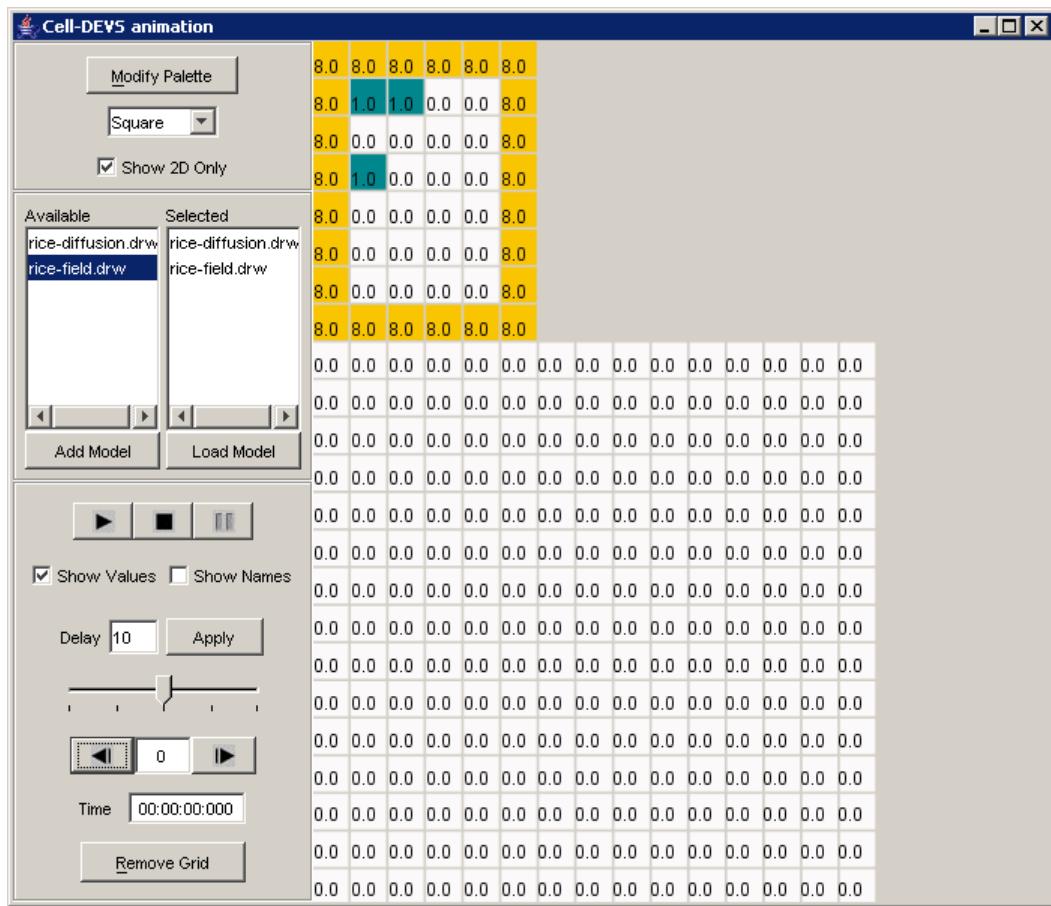


Figure 278: Settings applied to both loaded visualizations

Running/Navigating the visualization

The visualization of a coupled model can be run/navigated using standard procedures, as described in previous examples.

Continuing with this example, using the palette (ie. ditch.pal) for the diffusion component of the RiceField model, the cell display area at the end of the visualization appears as follows:

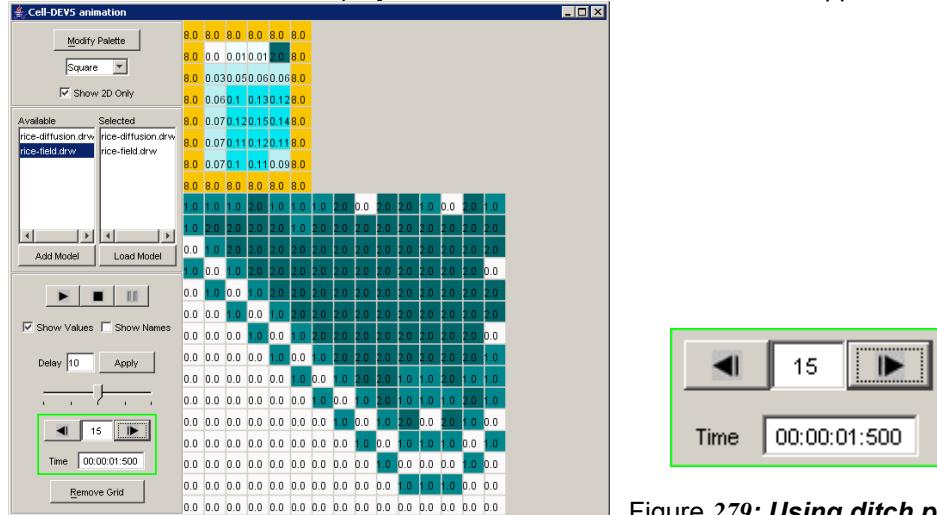
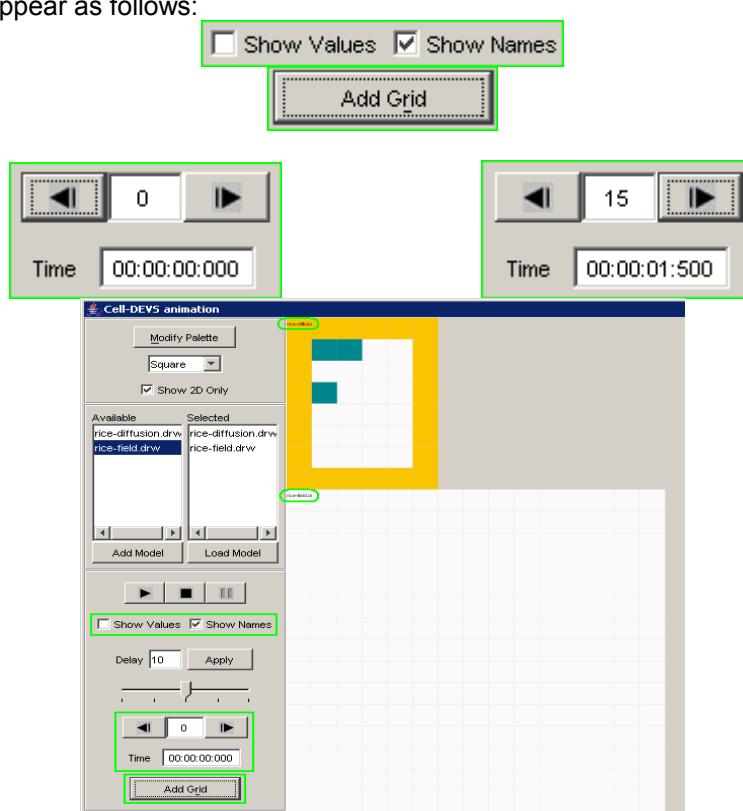


Figure 279: Using *ditch.pal* for the diffusion component

With Show Values unchecked, Show Names checked, and the Grid removed, the start and end of the visualization appear as follows:



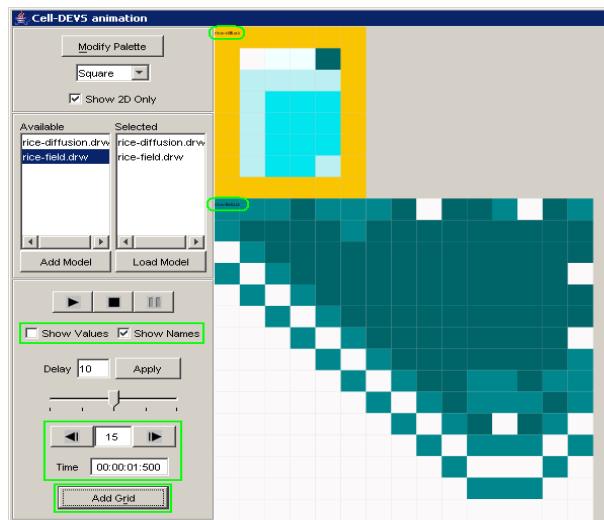


Figure 280: *Display for a specific set of options*

Note: When trying to view the results for a particular component of a coupled model, use the palette corresponding to the component.

Consider when the palette (ie. field.pal) for the field component of the RiceField coupled model is used. The cell display area at the start and end of the visualization will appear as follows:

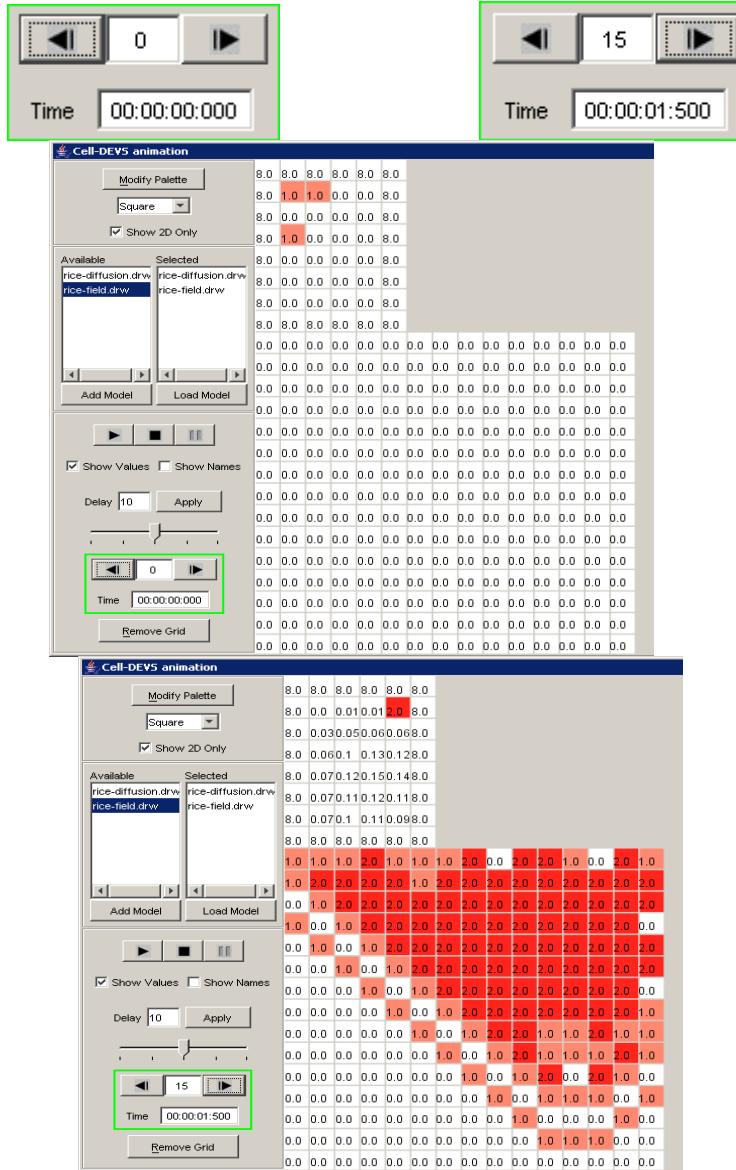
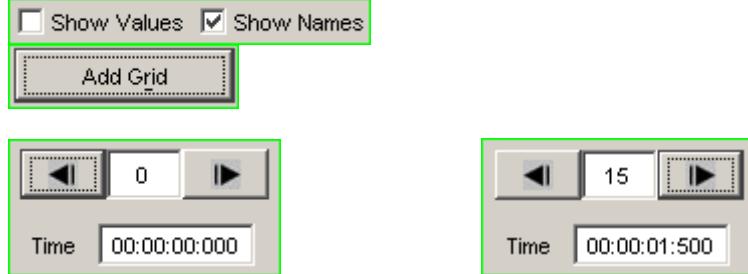


Figure 281: Cell display at start and end

With Show Values unchecked, Show Names checked, and the Grid removed, the start and end of the visualization appear as follows:



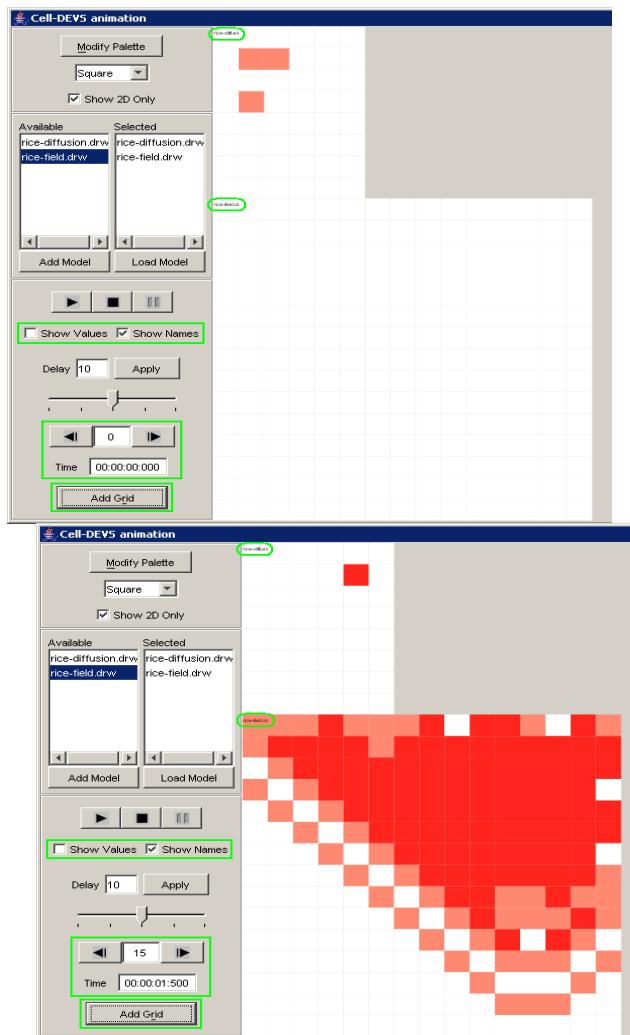


Figure 282: Start and end of visualization with a specific set of settings

ther tools (using the Execute menu commands)

This section describes the commands in the Execute menu, which are used for simulating DEVS models. (These commands seem similar to the tools in the CD++ Builder plugin for Eclipse.)

From the main menubar, left-click on Execute. The following menu will be displayed. Select the appropriate command, depending on the tool to be used for simulation.

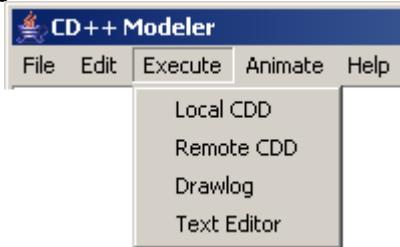


Figure 283: Execute Menu

The following sections describe the commands of the Execute menu for the newest stand-alone version of CD++ Modeler (as of Dec.1/2004), not the version included in the CD++ Builder plugin for Eclipse.

Local CDD

After selecting Local CDD (ie. left-click Local CDD from the Execute menu), the Run Simulator window appears:

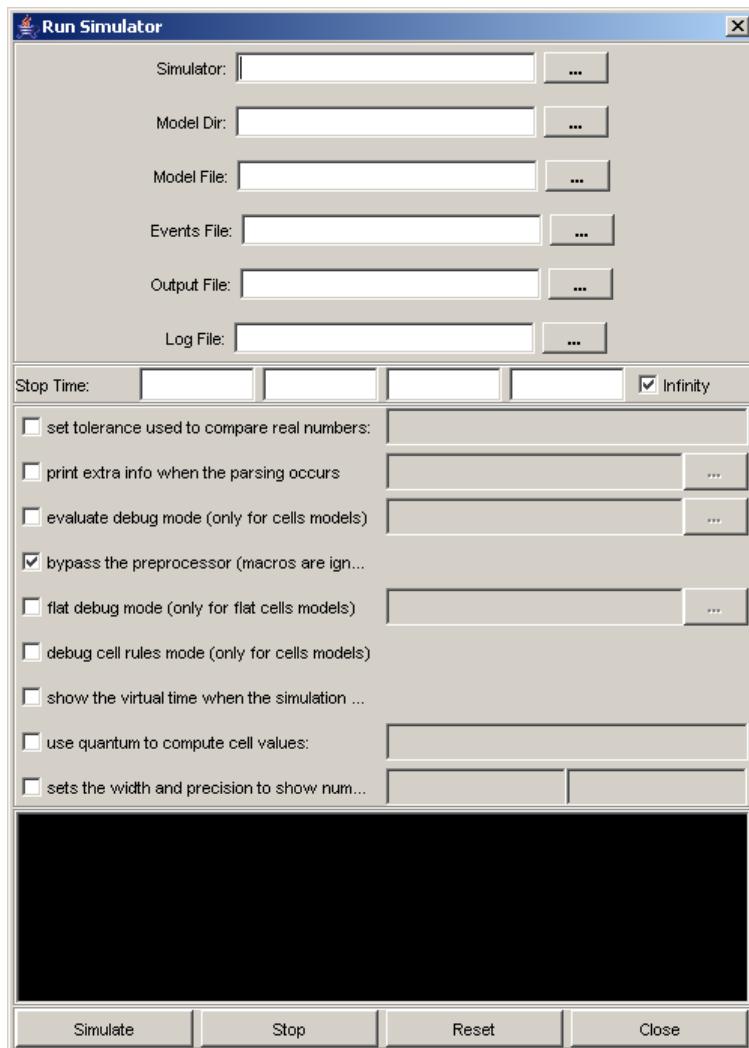


Figure 284: The Run Simulator Window

To simulate the specified Model File (using the specified Events File), press the Simulate button. The results will be output to the specified Output File and Log File.

To stop the simulation of the specified model at any time, press the Stop button.

To clear the dialog fields, press the Reset button.

To close the Run Simulator window, press the Close button.

For this example, the SatelliteClouds model was simulated, with the dialog fields filled as follows:

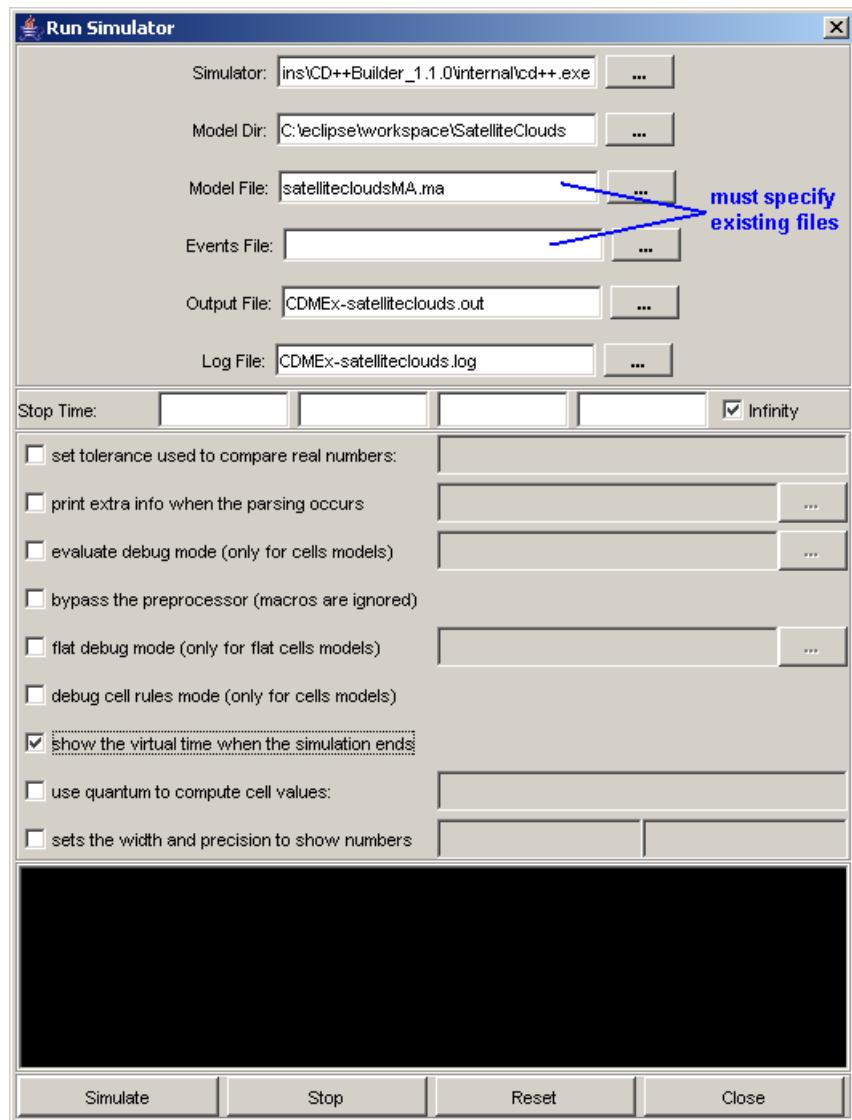


Figure 285: Specifying existing files

Note: The simulation for the SatelliteClouds model using Local CDD consumed more time than using the Simu! command in the CD++ Builder plugin. It is recommended that Local CDD is not used for simple models.

When the simulation of SatelliteClouds was stopped (before the simulation had completed), the console appeared as follows:

```
PCD++: A Tool to Implement n-Dimensional Cell-DEVS models
-----
Version 3.0 May-2001 - Compiled for StandAlone Simulation
Troccoli A., Rodriguez D., Wainer G., Banyiko A., Beyoglonian J.
Departamento de Computacion, Facultad de Ciencias Exactas y Naturales.
Universidad de Buenos Aires, Argentina.

Loading models from satellitecloudsMA.ma
Loading events from
Running parallel simulation. Reading models partition from
Model partition details output to: /dev/null*
Message log: CDMEEx-satelliteclouds.log
Output to: CDMEEx-satelliteclouds.out
Tolerance set to: 1e-08
Configuration to show real numbers: Width = 12 - Precision = 5
Quantum: Not used
Evaluate Debug Mode = OFF
Flat Cell Debug Mode = OFF
Debug Cell Rules Mode = OFF
Temporary File created by Preprocessor = /tmp/td20.0
Printing parser information = OFF

Stop at time: Infinity.
LP 0: initializing simulation objects
```

Figure 286:Console view after Simulation Stopped

Note: Local CDD appears to be a tool for parallel simulation. When testing the SatelliteClouds model, numerous log files (of the form *.log#, where # = {1,..., >100}) were generated.

(Please see Appendix D.)

Note: More testing is required for this tool.

Remote CDD

After selecting Remote CDD, a Run Simulator window appears which is identical to the Run Simulator window for Local CDD.

Text Editor

After selecting Text Editor, the following window appears:



After this window is resized:



Note: Text Editor appears to be non-functional.

(Please see Appendix D.)

8.16.7.4 Drawlog

After selecting Drawlog, the Run Drawlog window appears:

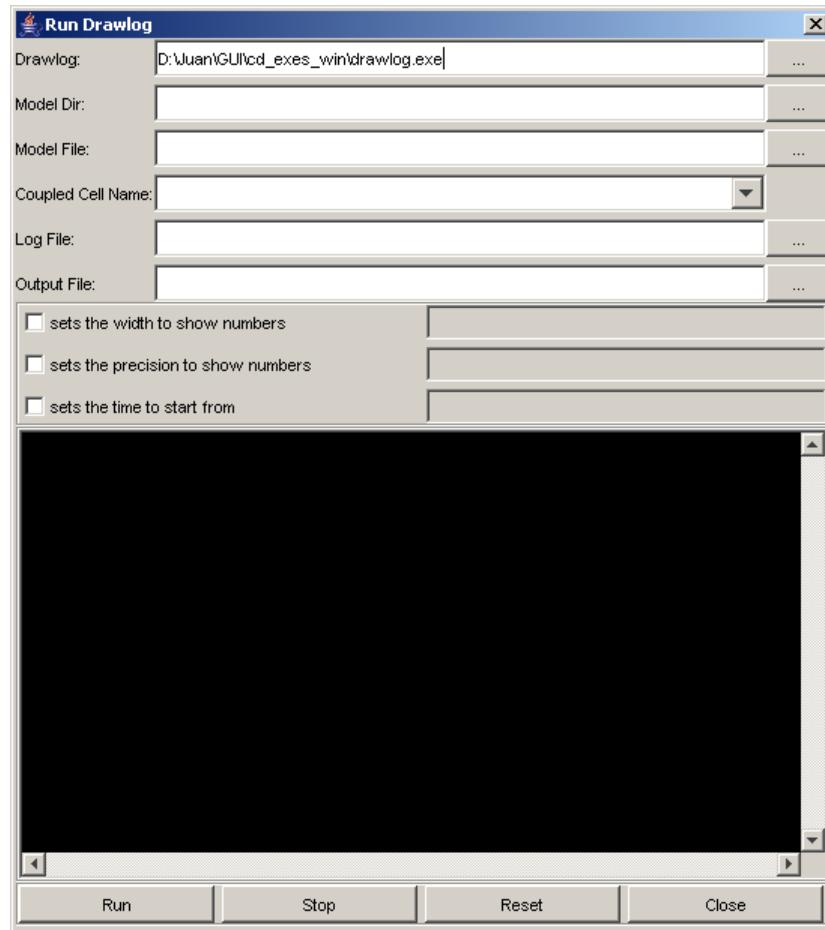


Figure 287: The Drawlog Window

To drawlog the chosen Coupled Cell Name of the specified Model File, press the Run button. The results will be output to the specified Log File and Output File (.drw).

To stop drawlog-ing the specified model at any time, press the Stop button.

To clear the dialog fields, press the Reset button.

To close the Run Drawlog window, press the Close button.

For this example, the SatelliteClouds model was drawlog-ed, with the dialog fields filled as follows:

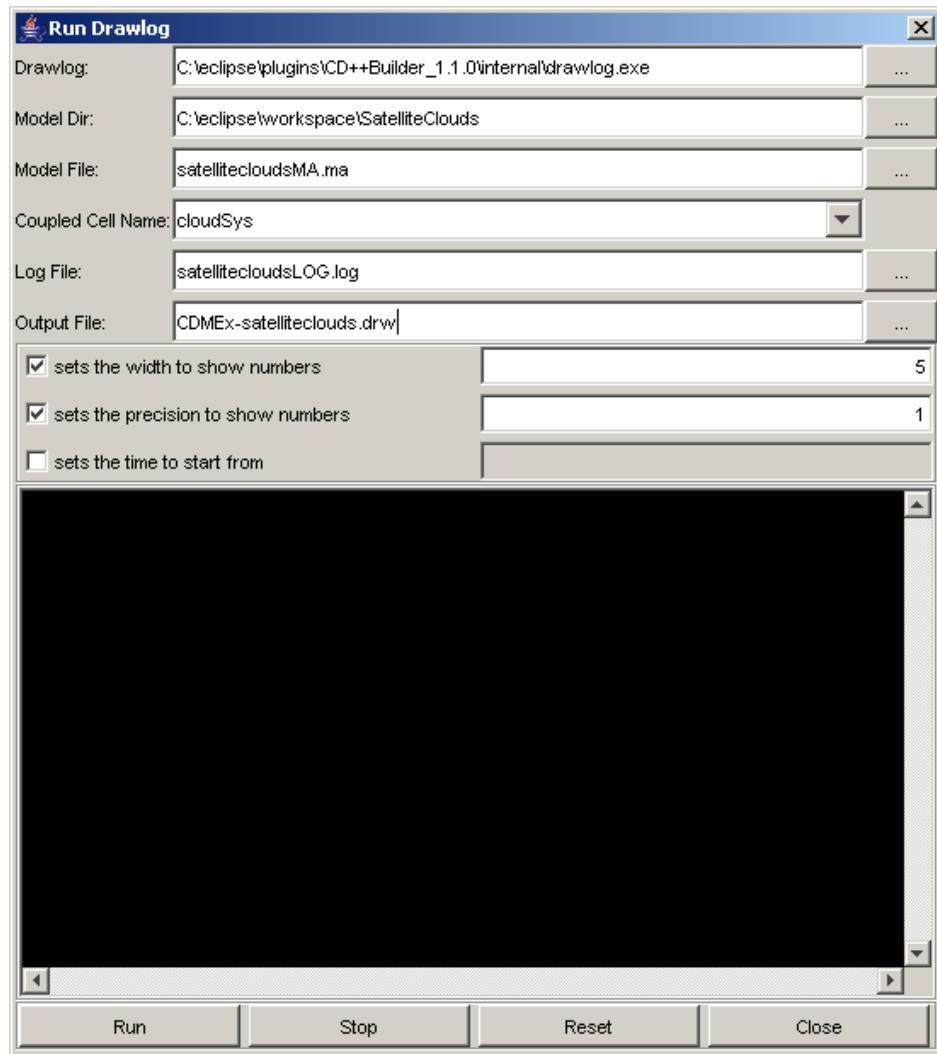


Figure 288: Entering Parameters in the Drawlog window

During the drawlog of SatelliteClouds, the console appeared as follows:

```
Writing output to file
```

A screenshot of a black terminal window. The only visible text is "Writing output to file" in green at the top left.

Figure 289: Console View While running Drawlog

When the drawlog of SatelliteClouds had completed, the console appeared as follows:

```
Writing output to file  
Process Finished  
Writing output to file finished
```

Figure 290:*Console view after process has finished*

This was also how the console appeared when the drawlog of SatelliteClouds was stopped before the drawlog had completed.

(Please see [Appendix D](#).)

8.16.8 CD++ Modeler: Suggestions & Bugs

The following are known bugs found in CD++Modeler. Suggestions for fixing these bugs are included for each bug.

Creating a Graphical Coupled Model (.gcm)

Note: "Resource panel" refers to the top lateral panel of the CD++ Modeler design space.

SCENARIO: Importing .cpp files.

CURRENT BEHAVIOUR: After importing a .cpp file of an atomic model, the atomic model does not appear in the Resource panel.

SUGGESTED BEHAVIOUR: After importing a .cpp file of an atomic model, the atomic model could appear in the Resource panel, in the Atomic folder.

CURRENT SOLUTION: Import the register.cpp file in which the atomic model is registered. The atomic model will appear in the Resource panel, in the Atomic folder.

ADDENDA: The current solution also has some functionality issues...

SCENARIO: Adding ports to resource models (imported via register.cpp).

CURRENT BEHAVIOUR: After importing a register.cpp file, the atomic models (registered in the register.cpp file) appear in the resource panel. After creating a unit of the atomic model, ports cannot be added to the unit within the Modelling panel.

CURRENT SOLUTION: Before creating a unit of the atomic model, right-click on the atomic model in the Resource panel. In the popup menu, to add a port to the atomic model, left-click Add Port, and type in the appropriate parameters in the dialog box. Once all ports have been added to the atomic model in the Resource panel, a unit of the atomic model can be created. The newly-created unit does not require (and does not allow) ports to be added.

SUGGESTION: The Resource panel could also display the ports available for each imported model.

ADDENDA: Once ports are added to a model, and units of the model are created, the ports of the units must be linked...

SCENARIO: Linking ports between units of models.

CURRENT BEHAVIOUR: A port cannot be accessed individually before a link is created involving the port.

CURRENT SOLUTION: To create a link between two ports, first create a link between the two units that contain the ports. Right-click on the link. In the popup menu, left-click Properties. Select the ports of the two units that are to be linked.

SCENARIO: Importing resources for an exploded model.

CURRENT BEHAVIOUR: After exploding a new coupled model, there are no resources in the Resource panel. The resources available in the "parent" editor are not automatically available for the exploded model. The exploded model must import all resources it requires.

SUGGESTED BEHAVIOUR: After exploding a new coupled model, the resources available in the "parent" editor could be automatically made available for the exploded model. So, the exploded model will not need to re-import the resources that the "parent" editor uses.

SCENARIO: CD++ Modeler frozen after period of time not in use.

AtomicAnimate: Suggestions & Bugs

Example: FunctionEval

UNKNOWN: units of values in From/To fields

ADDENDA: For the FunctionEval example only, the units for the From/To fields appear to correspond to milliseconds (since FunctionEval is a single atomic model, whose .log file messages occur at millisecond intervals).

SCENARIO: Showing the values of the output signal on the plot.

CURRENT BEHAVIOUR: For a given time interval, the Values checkbox is unchecked. The values are no longer visible on the plot. After the Previous/Next/Jump button is pressed (ie. the interval is changed), the Values checkbox is still unchecked. However, the values are visible on the plot of the current interval. When the Values checkbox is re-checked, the values are no longer visible on the plot of the current interval.

SUGGESTED BEHAVIOUR: For a given time interval, the Values checkbox is unchecked. The values are no longer visible on the plot. After the Previous/Next/Jump button is pressed (ie. the interval is changed), the Values checkbox is still unchecked. The values should not be visible on the plot of the current interval. When the Values checkbox is re-checked, the values should become visible on the plot of the current interval.

Example: 4BitCounter

INITIAL BEHAVIOUR (DEFAULT)

-
- The available signals for the first component in the drop-down list will be displayed.

* Note: 'interval' refers to the values in the From and To fields.
'time' refers to an absolute value of time.
'instance' refers to a step in the simulation.

- For example, from the 4BitCounter example:

the start time = 00:00:00:000	the start instance = 0
the end time = 00:00:00:400	the end instance = 662

- If the number of steps in the visualization is less than 1000:
 - the From field will contain the start instance/step (corresponding to the first message time in the simulation's .log file, usually 0);
 - the To field will contain the end instance/step (corresponding to the last message time in the simulation's .log file).
- The initial interval spans the entire simulation. For example, from the 4BitCounter example, the initial interval is [0,662].

- If the number of steps in the visualization is greater than 1000:
 - the From field will contain the start instance/step (corresponding to the first message time in the simulation's .log file, usually 0);
 - the To field will contain the value 1000.
 - This indicates an initial interval of [0,1000].
-

Note: The refresh button refreshes the display of the plots in the visualization window.

SCENARIO: Available output signals (for particular component) exceed space in left column.

CURRENT BEHAVIOUR: For example, for an interval spanning the entire simulation, after selecting the top component from the drop-down list, the fields (for setting the vertical scale) below the signal names are not entirely visible. There is not enough space in the left column to display the fields, and so it is not possible to set the vertical scales.

SUGGESTION 1: Horizontal scrollbar for left column. Would increase visibility of signal names and fields in left column. Could reduce the need to continually resize the visualization window.

SUGGESTION 2: Instead of displaying Set button and fields (for setting the vertical scale), could have a pop-up menu that appears when the signal name is right-clicked. In the pop-up menu, could have a 'Set vertical scale' command, in which a 'Set vertical scale' dialog (with a Set button and fields) could appear. The pop-up menu could also have other commands. Would reduce amount of space required for displaying signal names in left column.

SCENARIO: After changing interval, selected component not in drop-down list.

Refer to:

AtomicAnimate Example of Coupled-DEVS: 4BitCounter >***Availability of signals for the selected model/component***

CURRENT BEHAVIOUR: For a given interval, a particular component is selected. If the interval is changed, and the currently selected component does not have any available signals for the current interval, the component will remain selected (even though it is not in the drop-down list). The available signals displayed will correspond to the first component in the drop-down list, not the currently selected component. (Also, if another component is selected from the drop-down list, then it is not possible to re-select the originally selected component, until the interval is changed - in which the originally selected component has signals available.)

SUGGESTION: Visually indicate that the currently selected component does not have any available signals.

SUGGESTION 1: Do not display any signals, since none are available for the currently selected component.

SUGGESTION 2: Change the colour or disable the currently selected component.

SUGGESTION 3: As part of the visualization window, display a list of all components/models and their signals, ie. in a tree structure where left-clicking on a component name will expand a branch for corresponding signal names. For a given time interval, if a component has signals that are available, highlight/emphasize the component(s) and corresponding signal names. Could also have a pop-up menu when a signal name is right-clicked. In the pop-up menu, could have option for setting the colour of the signal's plot (and display a small colour box beside the signal name to indicate what colour is currently set for the signal's plot).

ADDENDA: Would help to clarify which signals correspond to which components.

SCENARIO: Colours of available signals.

CURRENT BEHAVIOUR: Colours of available signals automatically assigned based on order of signals in left column.

SUGGESTION: Allow user to associate particular colour with particular signal of particular component.

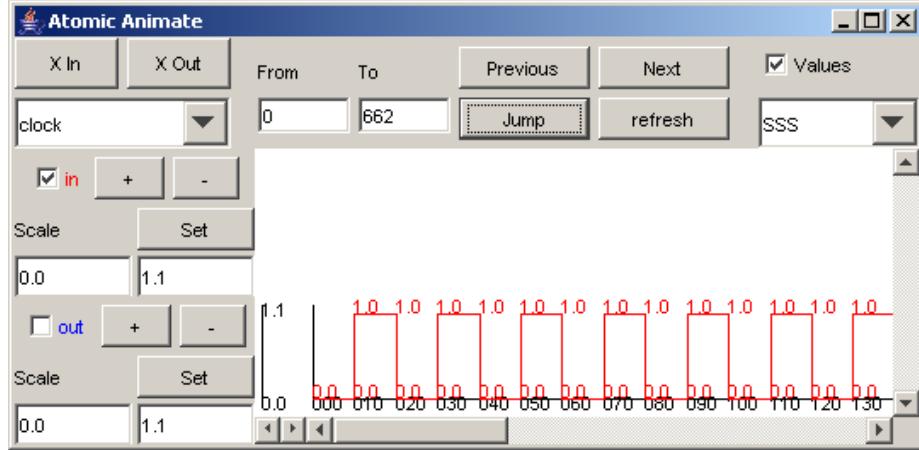
SCENARIO: Display of available signals for single component.

CURRENT BEHAVIOUR: The available signals of only one component can be displayed in the visualization window.

SUGGESTION: Allow available signals of multiple component to be displayed in the visualization window.

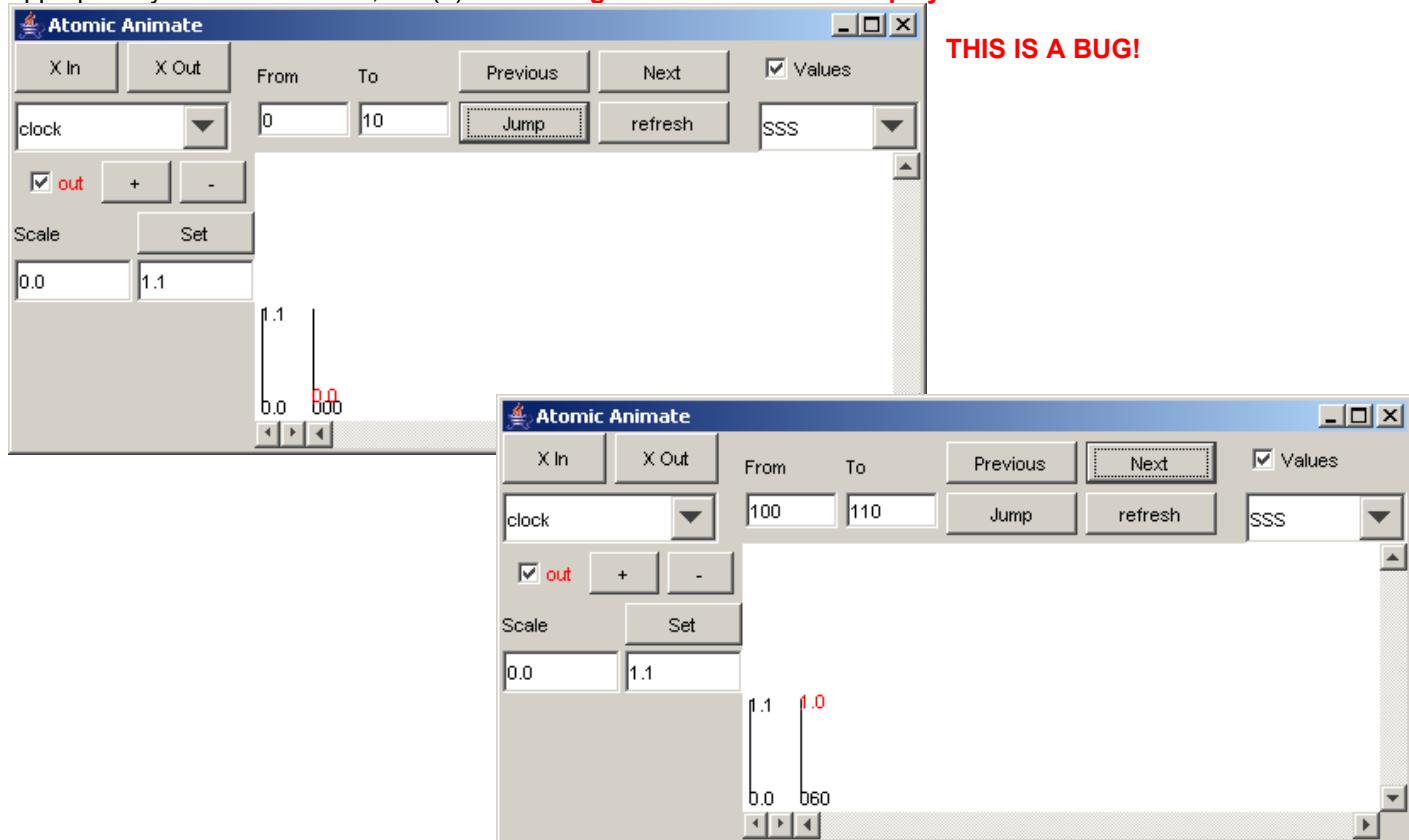
BUG SCENARIO: In the initial interval spanning the entire simulation, the out signal is unchecked.

So for any interval of the visualization, the plot of the out signal should not be displayed (until the out signal is rechecked).



CURRENT BEHAVIOUR: For any interval, since only the in signal is checked, only the plot of the in signal can be displayed.

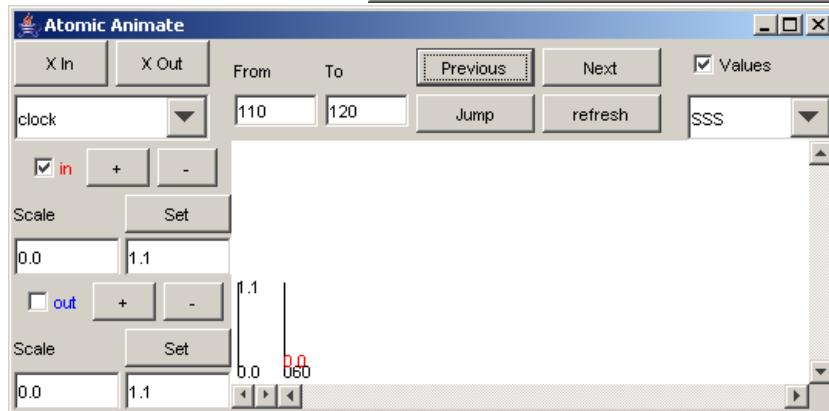
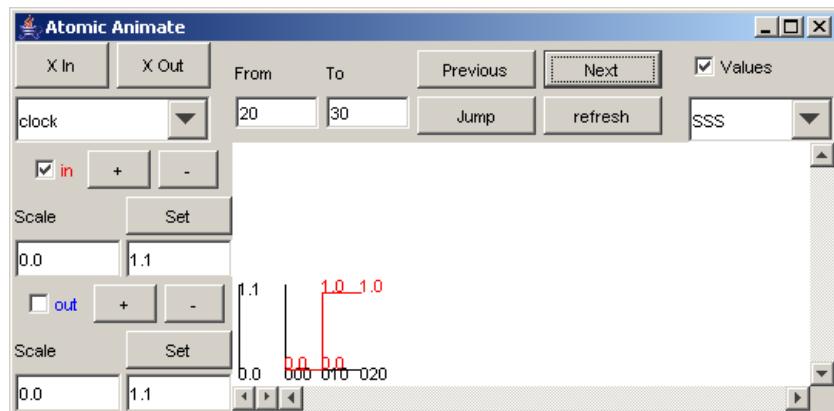
- However, in some intervals, where (1) only the out signal is available, (2) the out signal is appropriately in the left column, but (3) **the out signal is checked and displayed**:



THIS IS A BUG!

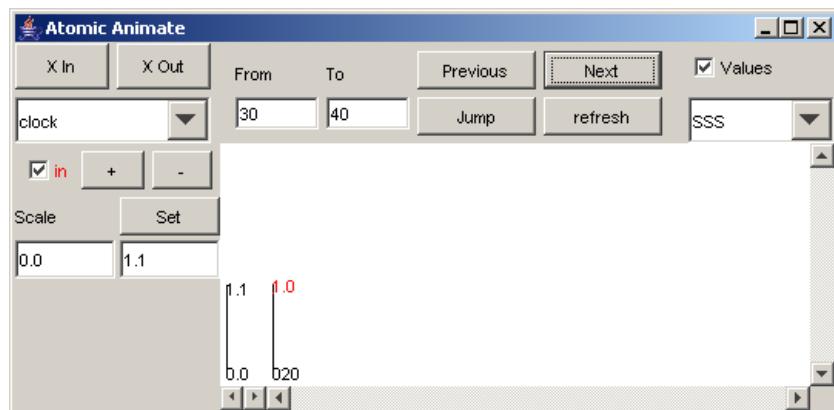
SUGGESTED BEHAVIOUR: The out signal should not be checked, and should not be displayed.

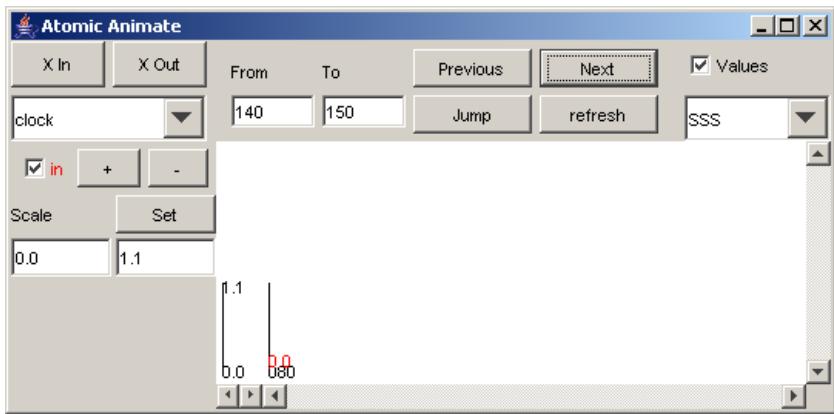
- In other intervals, where (1) both the in signal and out signals are available, (2) both signals are appropriately in the left column, and (3) the in signal is properly displayed and the out signal is properly unchecked:



THIS BEHAVIOUR IS CORRECT.

- In other intervals, where (1) only the in signal is available (and the out signal is unavailable), (2) the in signal is appropriately in the left column (and the out signal is appropriately not in the left column), and (3) the in signal is properly displayed:





THIS BEHAVIOUR IS
CORRECT.

Cell-DEVS animation: Suggestions & Bugs

UNKNOWN: functionality of horizontal scrollbar

UNKNOWN: units of value in Delay field

ADDENDA: assume units of milliseconds, but not verified during testing

Note 1: When adding a coupled model using an .ma file and a .log file, all cellular components of the coupled model will be added to the Available list.

Note 2: When adding a model using a .drw file, only the model specific to the .drw file (ie. one specific model/component) is added to the Available list.

SCENARIO: Organization of visualization window commands.

SUGGESTION: Organize components of visualization window according to command type, ie. for (1)loading model, (2) modifying appearance of visualization, (3) running/navigating visualization.

SCENARIO: Single palette file applied.

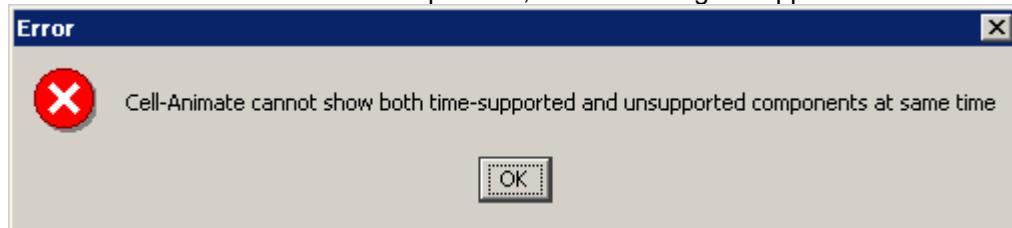
CURRENT BEHAVIOUR: Each loaded model (in the Selected list) uses the same .pal file.

SUGGESTION: Allow each loaded model (in the Selected list) to use different .pal files

SCENARIO: Loading both time-supported ('regular' .drw files, with border) and non-time-supported (.drw files created using the -f option in Drawlog) models.

Refer to the excerpt (on the following page) from the original manual.

CURRENT BEHAVIOUR: If both time-supported and non-time-supported models are in the Selected List when the Load Model button is pressed, an error dialog will appear.



8.13.14 Visualizing Cell-DEVS models

- The display sequence of selected models is in the order of their names in selected list. At any time after you have made changes to the selected list, press load model button to re-display the new selected models. In case of the selected list is empty, the load model button is automatically disabled.
- When all selected models have been loaded, CDModeler tries to load PAL file according to the name of the first entry in selected list as following. If such pal file does not exist or values are not defined in PAL file, CDModel now use white (previous use BLACK) as default.

Model name	Searched PAL file
<modelname>	<MODELNAME>.PAL
<modelname>@<log_filename>	<log_filename>.pal

- Currently, CDModeler does not allow loading both time-unsupported and time-supported models at same time. An error message is generated when that happens.

Model Type	Model Generated from
Time-unsupported model	Drawlog with -f argument
Time-supported model	Drawlog without -f argument
	Load directly from log file

Display simulation results for multiple models

4. Multiple Models from DRW Files and Log Files

- This can be used to validate the newly-added functionality.

Step 1. Load segment1a.drw file.

Step 2. Load traffic.ma, traffic.log files.

Step 4. Choose segment1a, crossing1a, segment1a@traffic and crossing1a@traffic models by double-clicking on them. Load the models by clicking on the Load Model button.

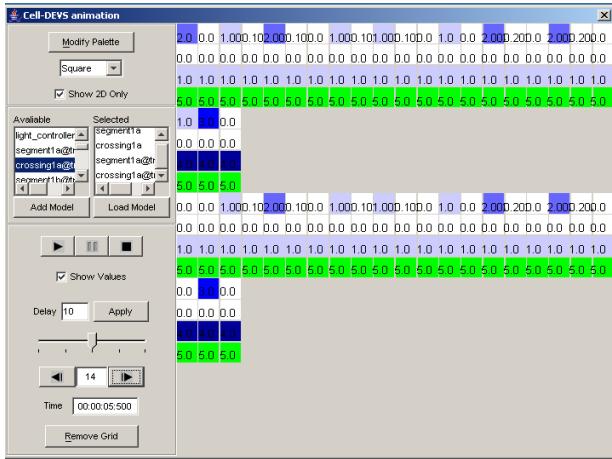


Figure 291. Traffic Models: from DRW and LOG Files

- Here, we can clearly see the bug of drawlog program: at this time, the (0,0) cell of both segment1a and crossing1a models send value 2 and 1 out through in_space port, but these values are considered as value of (0, 0) cells. The lower two models loaded from log file are displayed correctly.

8.13.15 Execute menu commands: Suggestions & Bugs

Local CDD

Notes:

- Appears to be a tool for parallel simulation.
- Takes up a lot of processing time (ie. can't use other running applications while waiting for Local CDD simulation to finish).
- Not recommended for simple models.
- Can't copy from console.

Text Editor

Notes:

- Non-functional, empty dialog.
- No menus/commands.
- Cannot input/edit text.

Drawlog

FIX: spelling in console output messages

SCENARIO: Console output for stopping drawlog prior to completion is the same as console output for drawlog completed in entirety.

CURRENT BEHAVIOUR:

- (1) After the Run button is pressed, when the drawlog of a model is complete, the console outputs the messages: > Writing output to file. > Process Finished. > Writing output to file finished.
- (2) If, after the Run button is pressed, before the drawlog of the model is complete, the Stop button is pressed, the console will output the same messages as (1).

SUGGESTION: For case (2), different messages should be output, indicating the drawlog was

stopped by the user prior to completion. The same messages should not be output for both (1) and (2).

SCENARIO: "Invalid parameter" warning generated in resulting .drw file.

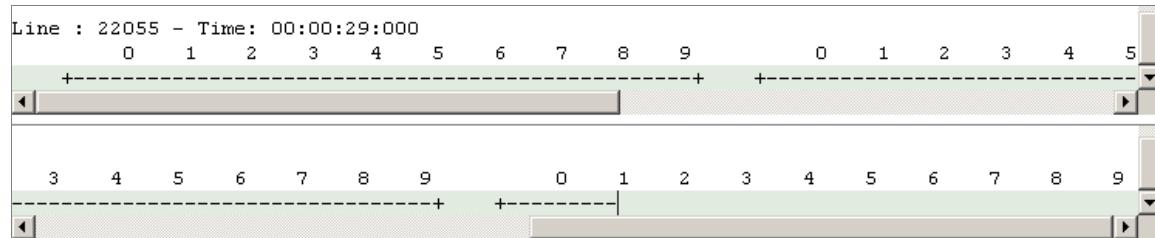
CURRENT BEHAVIOUR: After the drawlog of SatelliteClouds was complete, in the generated CDMEx-satelliteclouds.drw file, there is a warning at the top of the file:

Warning... invalid parameter >CDMEx-satelliteclouds.drw!

When the name specified for the Output File (.drw) was changed to satellitecloudsDRW.drw, the warning was still generated:

Warning... invalid parameter >satellitecloudsDRW.drw!

ADDENDA: Not sure what format the name of the Output File (.drw) should take so that a warning is



not generated.

SCENARIO: Results in .drw file end prematurely.

CURRENT BEHAVIOUR: In the CDMEx-satelliteclouds.drw file (as well as the satellitecloudsDRW.drw file), the results at end the beginning of 00:00:29:000.

SUGGESTED BEHAVIOUR: The results should end after 00:00:30:000, such as in the .drw files generated using the Drawlog command in the CD++ Builder plugin.

7 CGGAD Graphic Tool – User Manual

Introduction

The Coupled GGAD Graphic tool is used for designing coupled models. Some of the functions available are:

- Graphical design of a coupled DEVS model, which can be composed of: atomic and/or coupled models, the interconnections/links between the models, input ports, and output ports.
- Graphical design of the structure and functionality of an atomic DEVS model, which can be composed of: states, internal and external transitions, input and output ports, state variables, and actions associated with the transitions.
- Storage of generated models in ‘internal’ format, as well as in ‘export’ format for simulation using the CD++ application.

Definition

The CD++ simulation tool runs simulations of coupled DEVS models. For the definition of a coupled model to simulate, a language is used that describes the links between the atomic and/or coupled models that the coupled model contains. The atomic models can be classified as: “primitive” (ie. predefined operation/functionality in the CD++ simulator) or “GGAD” (ie. structure and operation defined within a user-generated text file).

The Coupled GGAD Graphic tool is comprised of 2 parts. One part permits the creation and edition (editing) of a coupled DEVS model; the other part permits the creation and edition (editing) of GGAD atomic DEVS models. The initial or root coupled model can include: other coupled models, GGAD atomic models, or primitive atomic models.

Each coupled model has a name and a text file that describes its composition. Each GGAD atomic model also possesses a name and a text file. Each primitive atomic model, however, possesses a name and a list of parameters with associated values. Since the CD++ simulator already ‘knows’ the structure and operation of each primitive atomic model, a text file is not required for primitive atomic models.

A coupled (root) model created with the Coupled GGAD Graphic tool should be exported to the CD++ format in order to be simulated. By exporting a coupled model, a .ma file will be created. The .ma file will have a section for each model that is integrated within the coupled root model. The sections corresponding to primitive atomic models will only have parameters. The sections corresponding to GGAD atomic models (or coupled models) will have the complete description of the GGAD atomic (or coupled) model’s structure, or the name of a file that contains this description.

9.1 Edition (Editing) of coupled GGAD (CGGAD) models

Initial (Beginning)

Once initialized, the application will appear as seen in the following figure.

Here, the division of the window into 3 groups of information can be seen:

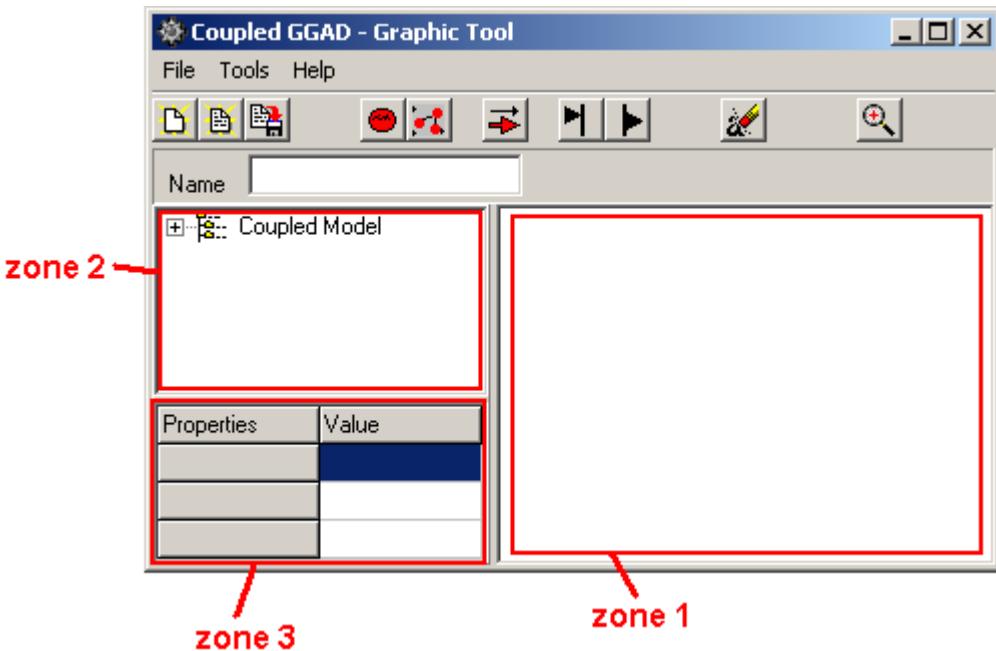


Figure 292: *The GGAD Window*

- Zone 1: the drawing of the coupled model (“drawing zone”)
- Zone 2: the hierarchical definition of the coupled model (“hierarchy zone”)
- Zone 3: the properties and values of the selected object (“properties zone”)

In order to start designing the coupled (root) model, component objects of the coupled model can be added. To add an object, in the Name field, type the name for the object to be added, and press the corresponding button in the toolbar.

Component objects that can be added: coupled models, atomic models, input or output ports of the coupled root model, input or output ports of each atomic model, and links between ports.

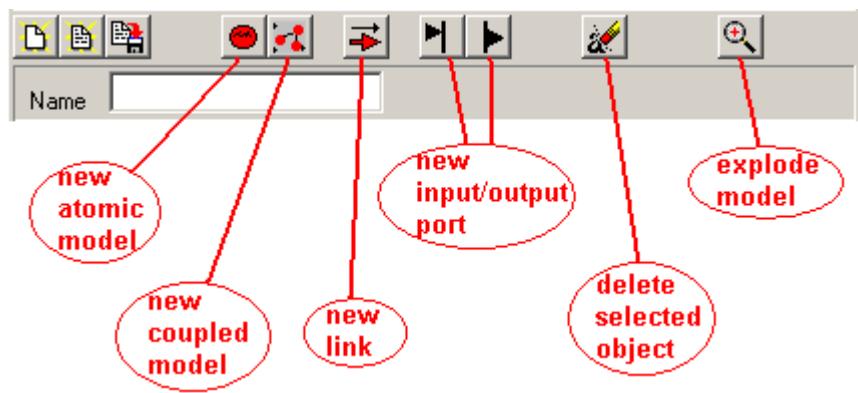


Figure 293: The GGAD Toolbar

Add atomic models

In order to add an atomic model, first type the name of the atomic model in the Name field (step 1), and then press the new atomic model button (step 2).

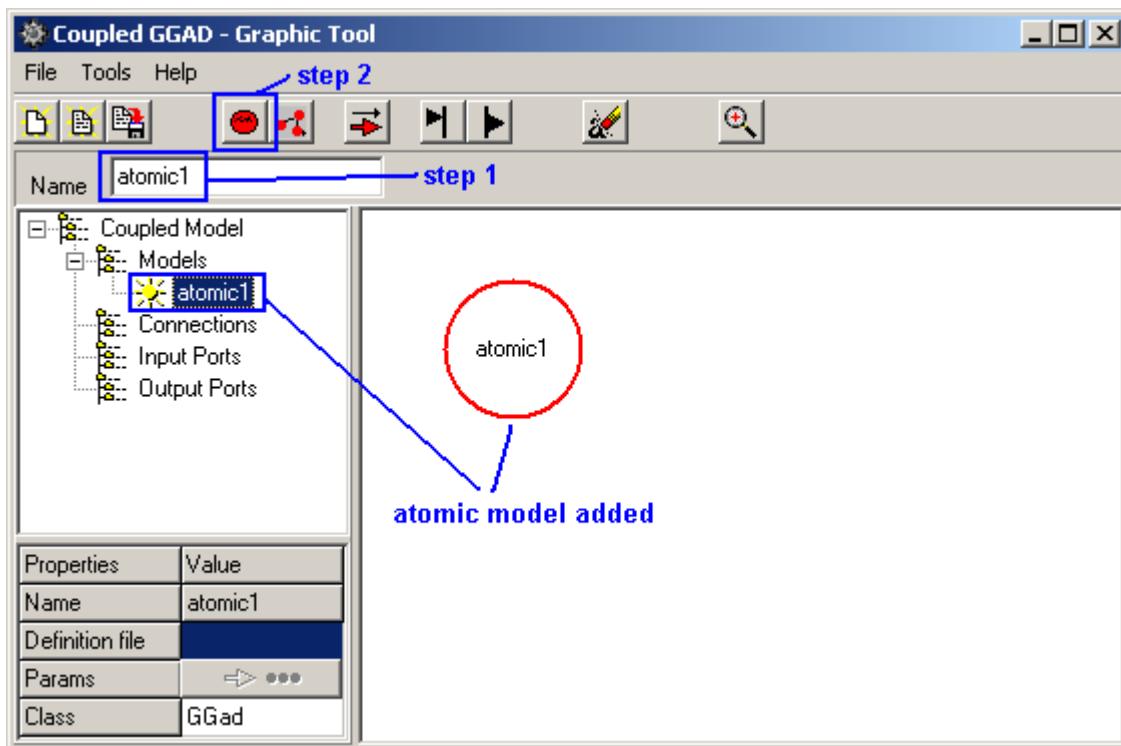


Figure 294: Adding Atomic Models

Once the atomic model is added, the values of the attributes in the properties zone can be changed. To do this, first select the atomic model, and then select the attribute in the properties zone.

A model can be selected either by clicking on the model in the drawing zone, or by selecting the name of the model in the hierarchy zone. The model will be highlighted either red (when selected within the drawing zone), or dark blue (when selected within the hierarchy zone).

Attributes of an atomic model:

- **Name:** can not be modified (assigned when the atomic model is first created)
- **Definition file:** file name (*.ggd) that defines this atomic model. Can be assigned when the atomic model is first created, so that the atomic model has the structure of an existing file. Alternatively, the atomic model can be exploded, and designed graphically in the new window, then saved in a file. (This alternative will be described in later sections.)
- **Params:** list of parameters for this atomic model. Can be used for defining parameters for an atomic GGAD model.
- **Class:** name of the class of the model. In the case of a GGad model, the class will be GGad. When adding a primitive model, its class should be specified.

Add coupled models

In order to add a coupled model, first type the name of the coupled model in the Name field (step 1), and then press the new coupled model button (step 2).

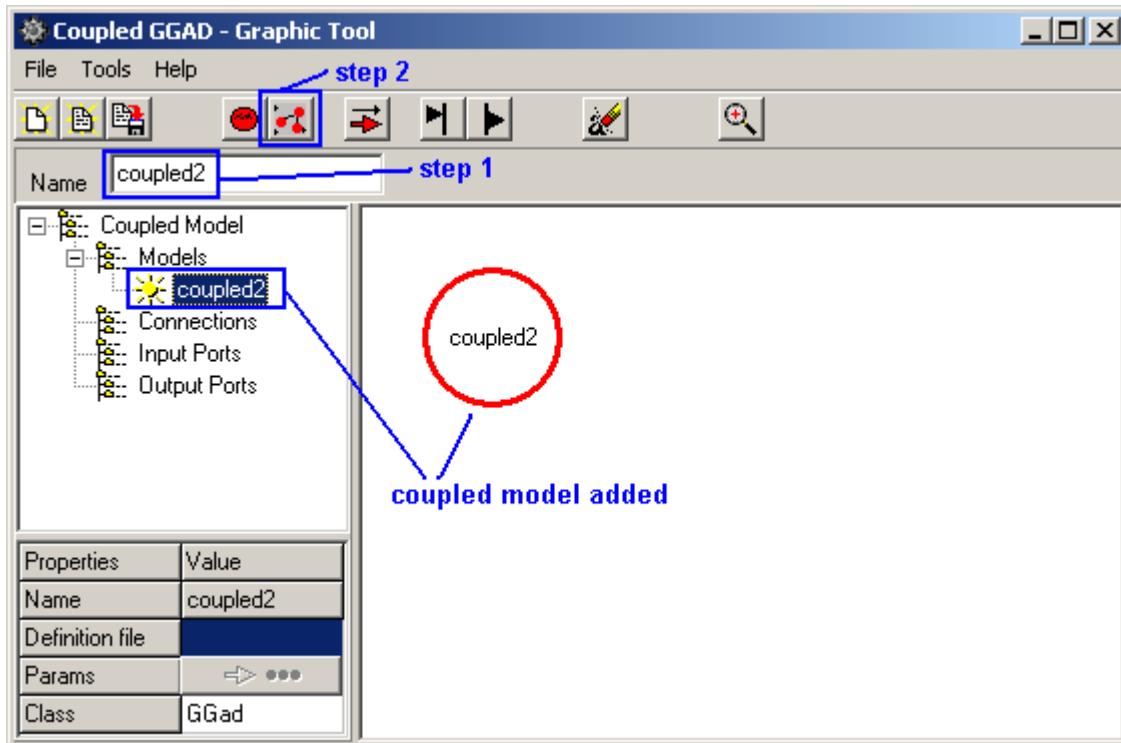


Figure 295: Adding Coupled Models

Once the coupled model is added, the values of the attributes in the properties zone can be changed. To do this, first select the coupled model, and then select the attribute in the properties zone.

Attributes of a coupled model:

- **Name:** can not be modified (assigned when the coupled model is first created)
- **Definition file:** file name (*.cgd) that defines this coupled model. Can be assigned when the coupled model is first created, so that the coupled model has the structure of an existing file. Alternatively, the coupled model can be exploded, and designed graphically in the new window, then saved in a file. (This alternative will be described in later sections.)
- **Params:** list of parameters for this coupled model. Should be used for coupled models that include primitive atomic models (eg. those predefined in CD++) that (after compiling) require parameters in order to be instantiated.
- **Class:** name of the class of the model. In the case of a GGad model, the class will be GGad.

Add ports

In order to add a port to a model (atomic, root, or coupled), first select the model, type the name of the port in the Name field, and then press the new input or output port button, depending if the port is input or output, respectively.

Ports can be created in two levels:

- as ports of the coupled (root) model
- as ports of the component, or “inner”, models (primitive atomic models or coupled models), in order to be able to define the links that occur between component models of the coupled (root) model

The model (to which a port will be added) can be selected either in the drawing zone or in the hierarchy zone. When a model is selected in the drawing zone, the same model will automatically be selected in the hierarchy zone. When a model is selected in the hierarchy zone, however, no changes will occur in the drawing zone, and so the model will not be selected automatically in the drawing zone.

Thus, if the model selected in the drawing zone is different from the model selected in the hierarchy zone, the hierarchy zone will take precedent. (ie. The port will be created for the model selected in the hierarchy zone.)

If the selected model is an atomic component, the port will be created for the selected atomic model. If the selected model is a coupled component, the port will be created for the selected coupled model. However, if the selected object is neither an atomic nor a coupled component, then the port will be created for the coupled (root) model.

component type		port will be added to ...
... selected in drawing zone	... selected in hierarchy zone	
atomic	atomic	atomic component *
atomic	coupled	coupled component *
atomic	not atomic, not coupled	coupled (root) model
coupled	atomic	atomic component *
coupled	coupled	coupled component *
coupled	not atomic, not coupled	coupled (root) model
none	atomic	atomic component *
none	coupled	coupled component *
none	not atomic, not coupled	coupled (root) model

* (Please see Appendix A for related bugs.)

Ports must be created for primitive atomic models.

For GGAD atomic models, the correct method of adding a port is to explode and edit the model. (Exploding an atomic model will be described in later sections.)

9.1.1 Add connections (links) between ports

The connections/links between ports have a starting and an ending point. The starting point can be: an input port of the coupled (root) model, or an output port of one of the inner (atomic or coupled) models. The ending point can be: an output port of the coupled (root) model, or an input port of another one of the inner (atomic or coupled) models.

In order to make a connection/link, follow these steps:

1. Select the starting port (in the hierarchy zone) for the link.
2. Press the new link button in the toolbar.
3. From the Choose Port dialog, click the ending port for the link, and press OK.

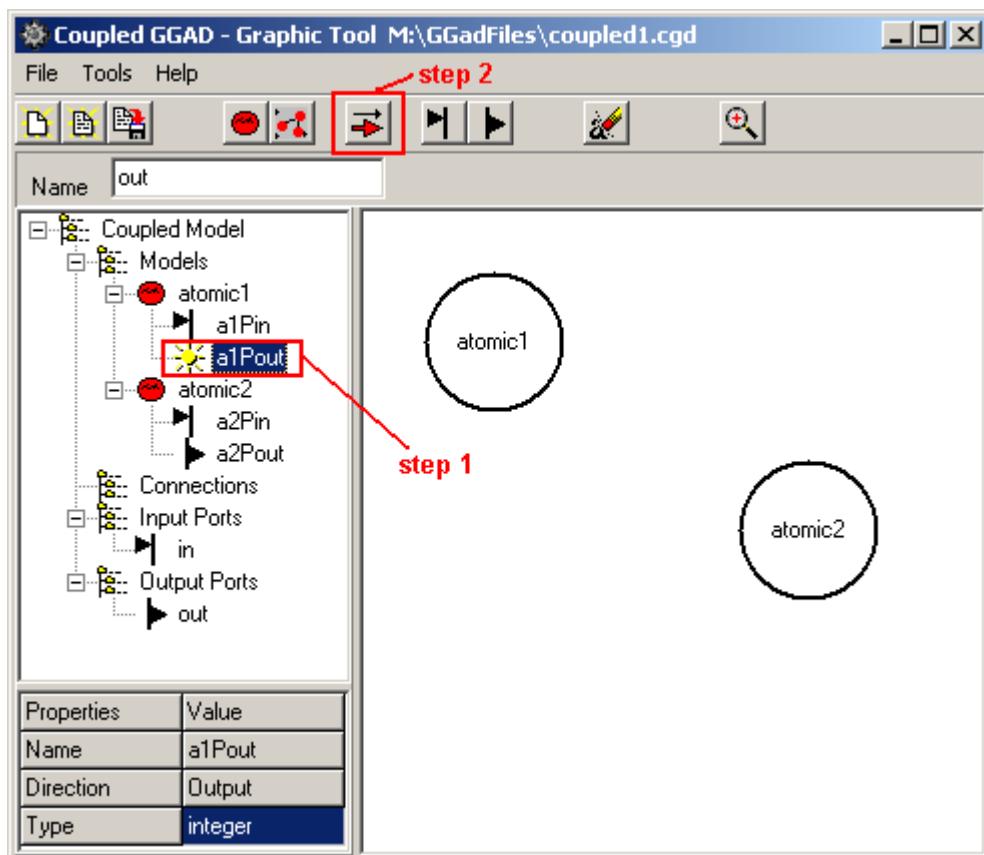


Figure 296:Adding Connections/Links Between ports(step1)

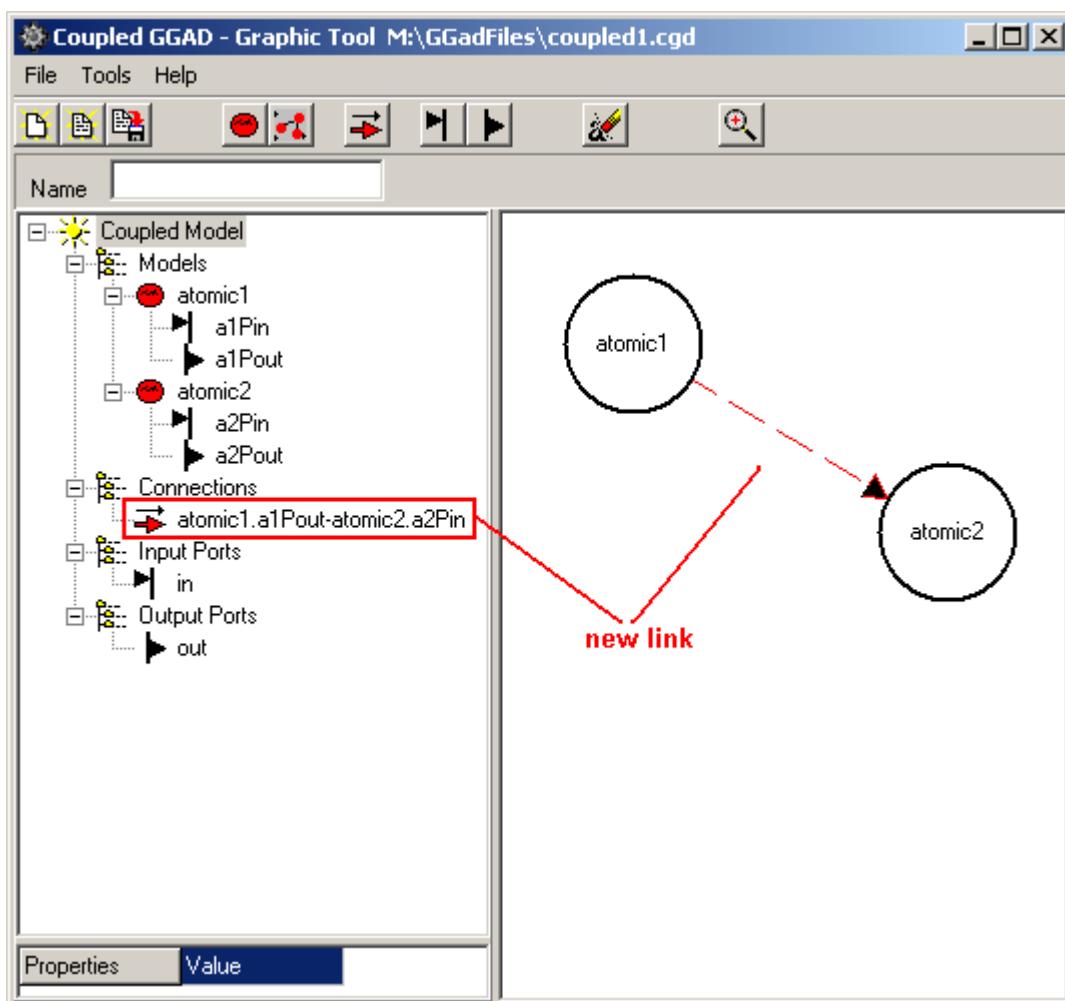
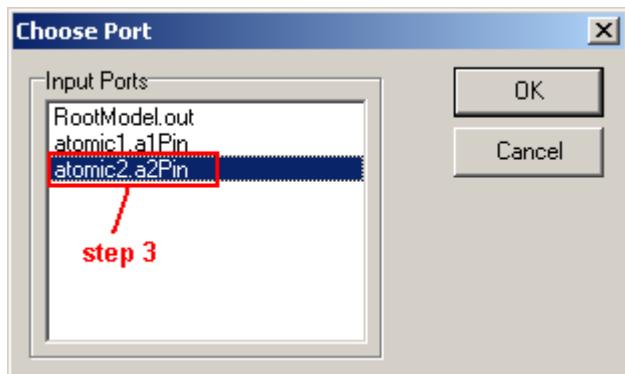


Figure 297: **Creating new link**

Note: An alternative way of making links in the hierarchy zone is to drag and drop the starting

port onto the ending port.

Delete objects

In order to delete an object, first select the object, and then press the delete selected object button in the toolbar. The component models and links between ports can be selected individually in the drawing zone or in the hierarchy zone. However, individual ports can be selected only from the hierarchy zone, since ports are not represented graphically in the drawing zone.

When an object is removed, all dependent objects are automatically also deleted. For example, if an atomic model is deleted, all the ports of the atomic model, as well as links to the ports of the atomic model, will also be deleted.

Explode a model

In order to explode an inner model (atomic or coupled) to be modified, first select the model, and then press the explode selected model button in the toolbar.

- If no object is selected, or if the selected object is not a model, nothing will be exploded.



Figure 298:**Message to select object before exploding**

- If the selected object is an inner coupled model without a Definition file:
- Another GGAD Graphic Tool window is opened, with an empty model representing the exploded coupled model which can be edited. (Section 3.x describes how to modify an exploded coupled model.)
- Once editing of the exploded coupled model is completed and saved in a file (*.cgd), can return to the original window by closing the exploded window. The changes made in the exploded coupled model should be reflected in the “Definition file” property of the selected inner coupled model.
- If the inner coupled model had defined ports or links before it was exploded, those ports and links will be updated (ie. removed/replaced) according to the Definition file.

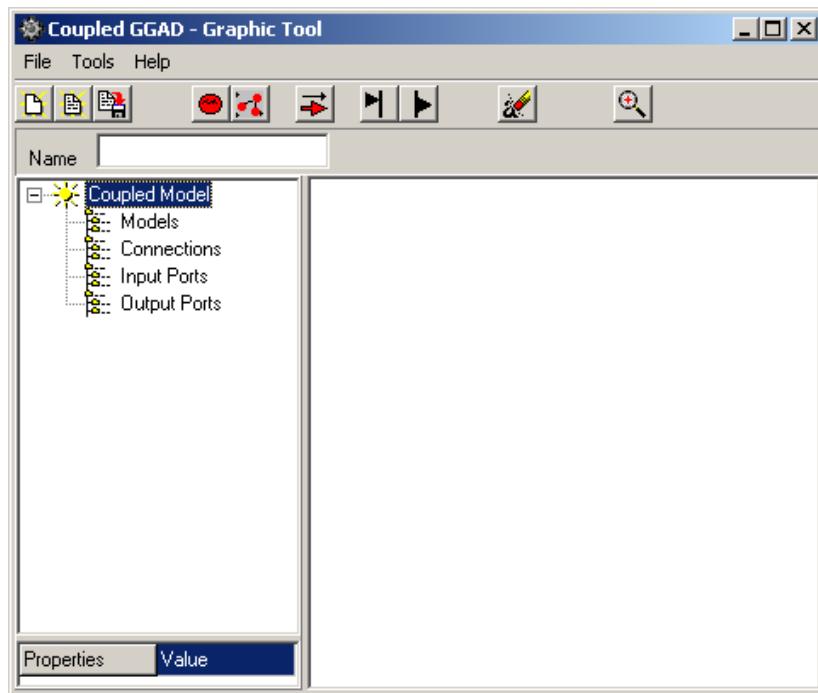


Figure 299: **Ports and Links updated for Coupled Models**

- If the selected object is an inner atomic model without a Definition file nor defined parameters:
- Another GGAD Graphic Tool window is opened, with an empty model representing the exploded atomic model which can be edited. (Section 4.x describes how to modify an exploded atomic model.)
- Once editing of the exploded atomic model is completed and saved in a file (*.ggd), can return to the root model level by closing the exploded window. The changes made in the exploded atomic model should be reflected in the “Definition file” property of the selected inner atomic model.
- If the inner atomic model had defined ports or links before it was exploded, those ports and links will be updated (ie. removed/replaced) according to the Definition file.

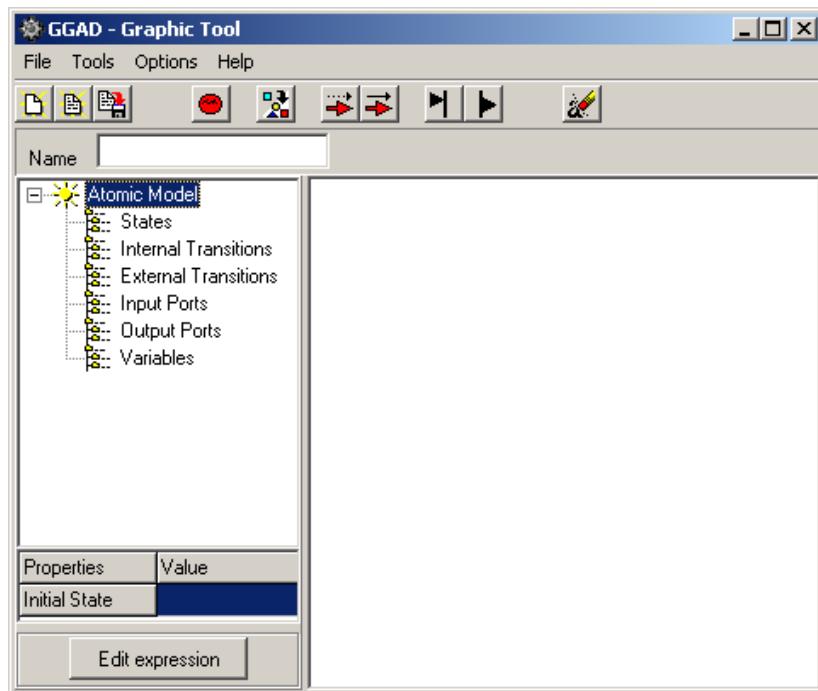


Figure 300: Ports and Links updated for Atomic Models

- If the selected object is an inner atomic or inner coupled model with a Definition file:
- Similar to the scenario for an inner atomic or inner coupled model without a Definition file.
- However, instead of opening “with an empty model”, the window is opened “with a model corresponding to the Definition file”.

When a selected inner model is exploded, the inner model is “detached” from the coupled (root) model. After closing an exploded model and returning to the original window, the attributes and ports that were created for the selected inner model prior to being exploded will be lost.

Save and export the coupled model

Once the coupled (root) model is completed, it can be saved in a file (*.cgd) for future modifications, and/or exported (as *.ma) for simulation in the CD++ simulator.

To save the coupled (root) model, select “Save” or “Save As...” from the File menu.

To export the coupled (root) model, select “Export to CD++” from the File menu. The .ma file format for exporting the coupled (root) model will only save information relevant for the CD++ simulator. So, the coupled (root) model should always first be saved in .cgd format, before being exported in .ma format.

Before exporting a model to .ma format, ensure the model is first saved in .cgd format (for coupled models).

Edition (editing) of GGAD atomic models

Initial (Beginning)

After selecting an inner atomic model and pressing the explode selected model button, a new GGAD Graphic Tool window will appear as seen in the following figure.

Here, the division of the window into 3 groups of information can be seen:

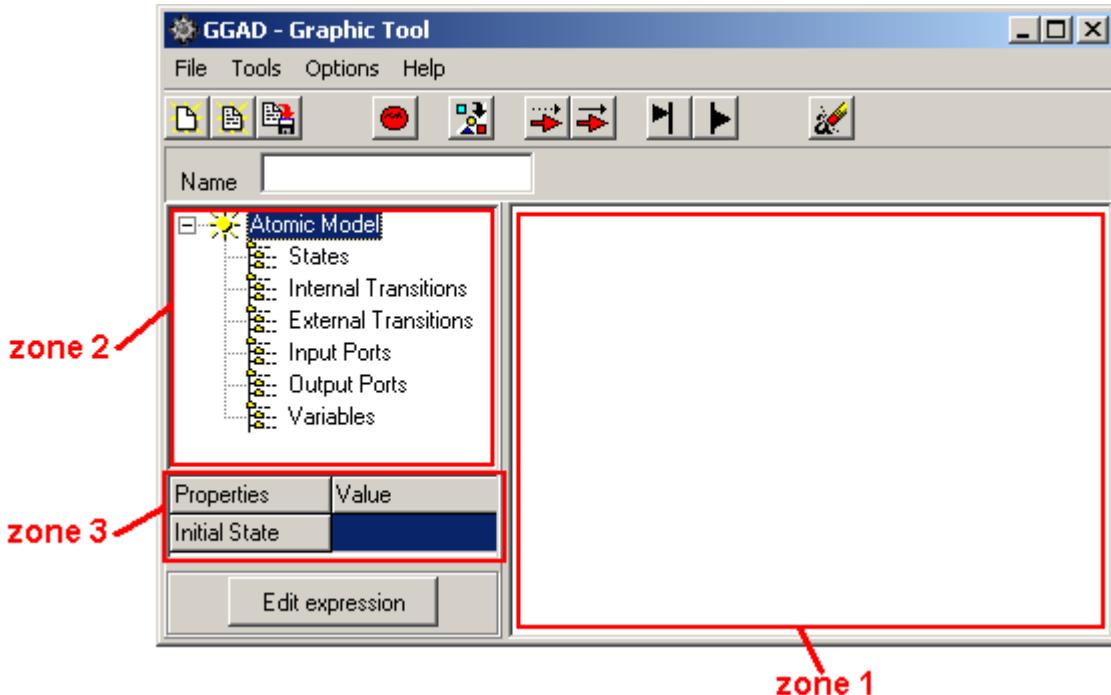


Figure 301: Dividing the window into 3 groups or zones

- Zone 1: the drawing of the atomic model (“drawing zone”)
- Zone 2: the hierarchical definition of the atomic model (“hierarchy zone”)
- Zone 3: the properties and values of the selected object (“properties zone”)

In order to start designing an atomic model, component objects of the atomic model can be added. To add an object, in the Name field, type the name for the object to be added, and press the corresponding button in the toolbar.

Component objects that can be added: states, internal or external transitions, input or output ports, variables of the model, actions for the transitions.

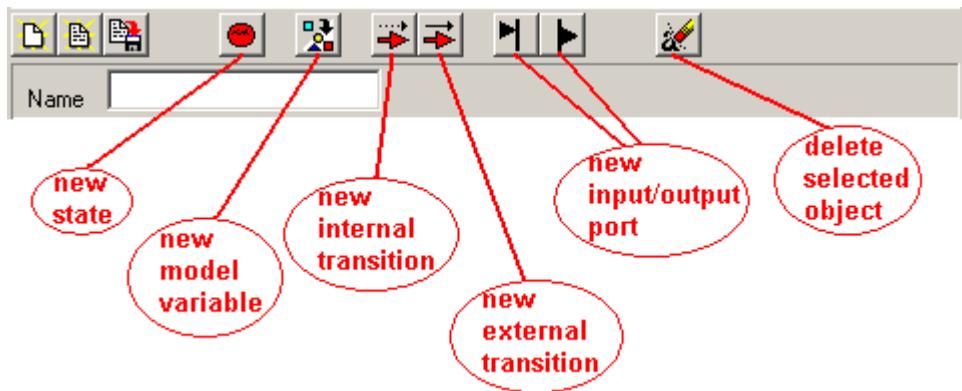


Figure 302: Toolbar available for user

Add states

In order to add a state, first type the name of the state in the Name field (step 1), and then press the new state button (step 2).

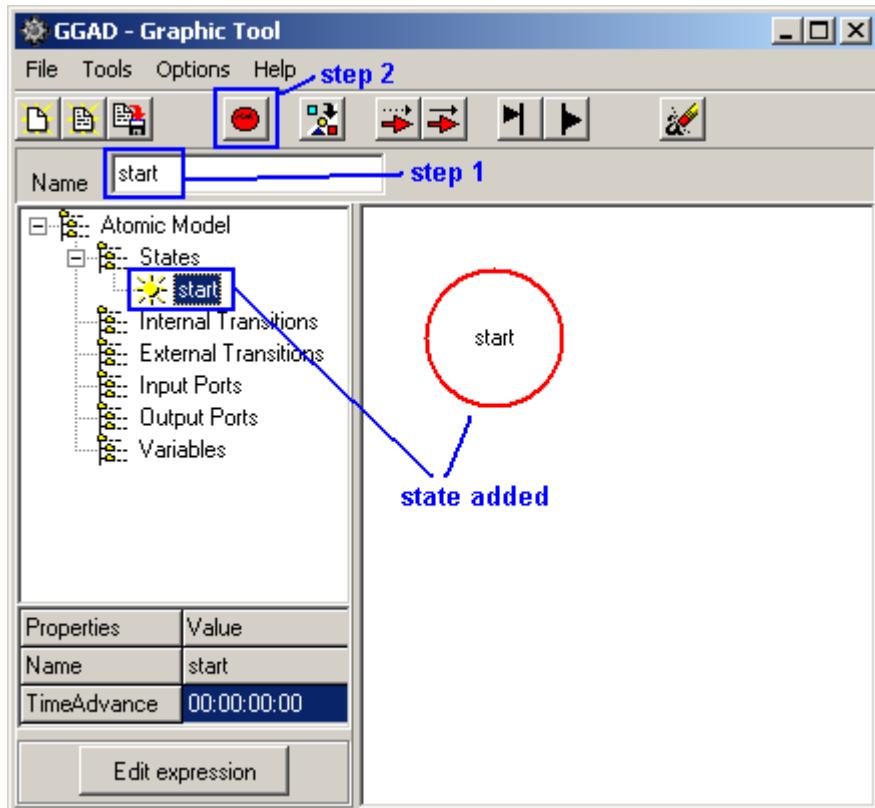


Figure 303: Adding states

Once the state is added, the values of the attributes in the properties zone can be changed. To do this, first select the state, and then select the attribute in the properties zone.

A state can be selected either by clicking on the state in the drawing zone, or by selecting the name of the state in the hierarchy zone. The state will be highlighted either red (when selected within the drawing zone), or dark blue (when selected within the hierarchy zone).

Attributes of a state:

- **Name** : can not be modified (assigned when the state is first created)
- **Time advance**: the time that the simulator remains in this state before triggering an internal transition. Its value can be expressed in units of time in the format 00:00:00:000 (hours:minutes:seconds:hundredths of second) or as infinite (**without commas**) if the desired time advance is infinite.

Add ports

In order to add a port to the atomic model, first type the name of the port in the Name field (step 1), and then press the new input or output port button (step 2), depending if the port is input or output, respectively.

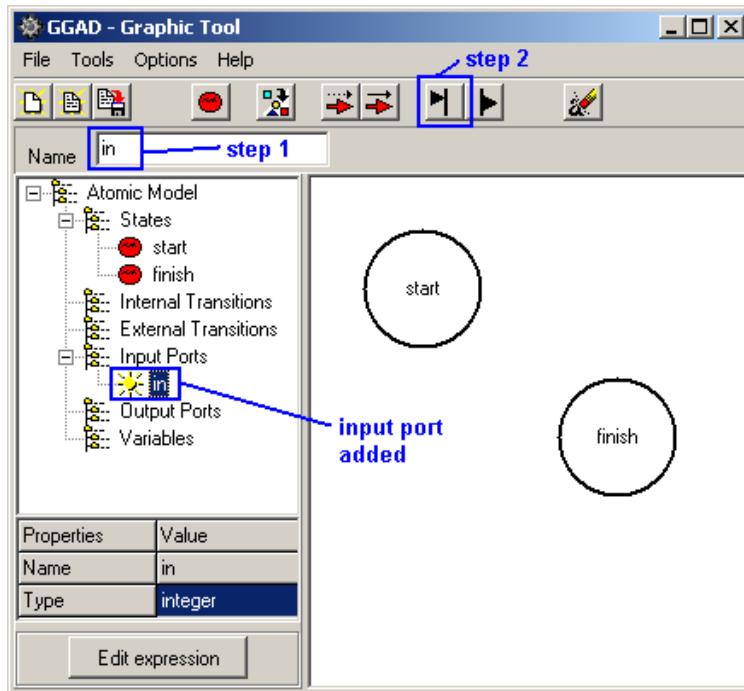


Figure 304: Adding Ports

Ports can be created only at one level: as ports of the exploded atomic model. Ports cannot be created for states. So, the selection of a state will not influence the creation of a port.

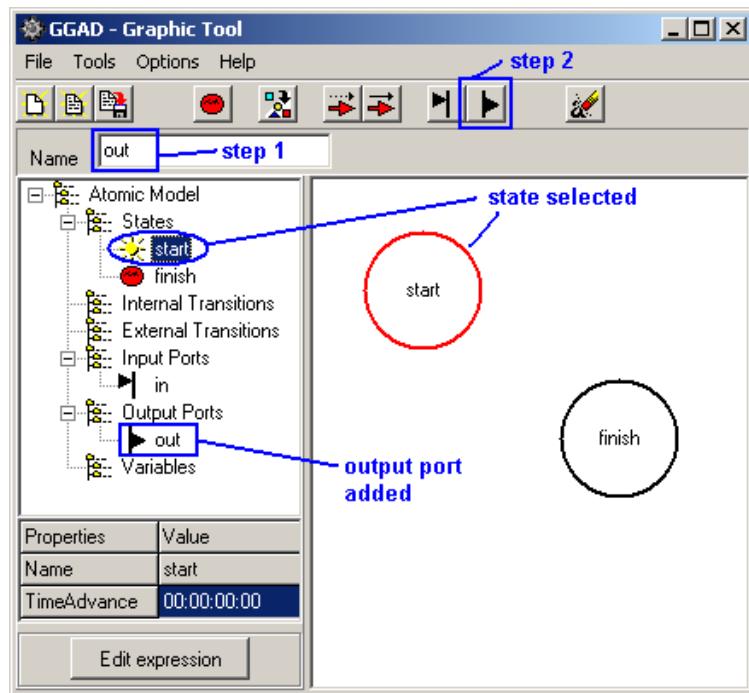


Figure 305:*Ports not being added to states*

Add state variables

In order to add a state variable, first type the name of the state variable in the Name field (step 1), and then press the new model variable button (step 2).

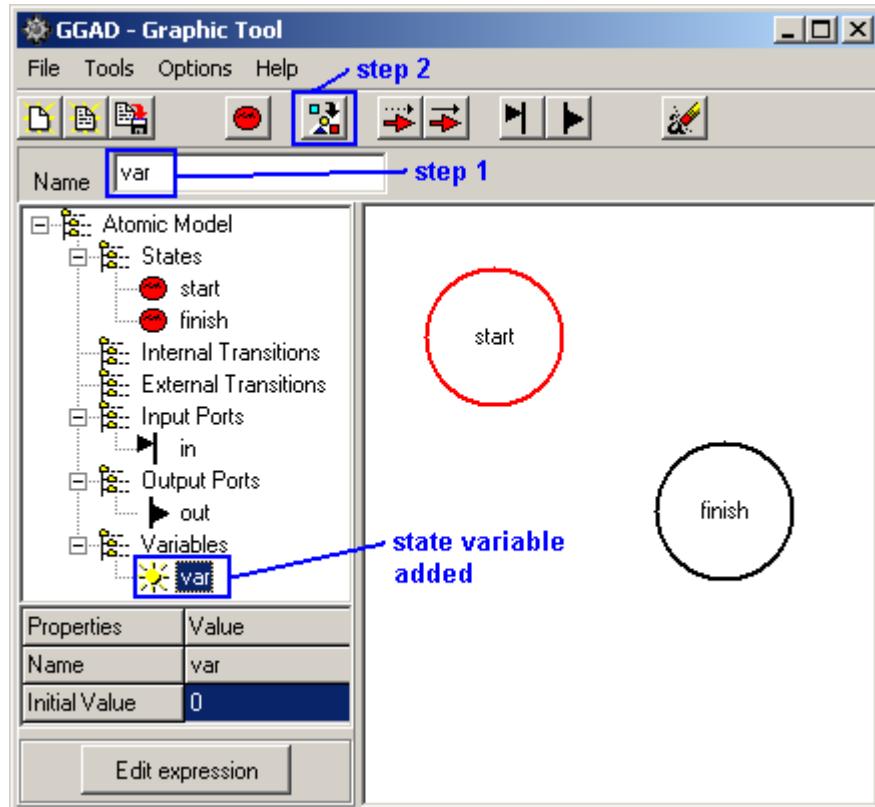


Figure 306: Adding state variables

Once the state variable is added, the initial value of the variable can be changed in the properties zone.

A state variable can only be selected in the hierarchy zone, since state variables are not visually represented in the drawing zone.

Some of the possible uses for state variables:

- Ascertain the conditions required for activation of an external transition
- Store the current numerical values for future use
- Establish or set counters

9.1.2 Add internal transitions

An internal transition can occur between 2 states, and can have an associated output function. The output function consists of zero or more port-value pairs. Each port-value pair specifies the distinct value that will be sent to a particular output port when the internal transition occurs. Port-value pairs must be specified before the occurrence of the internal transition.

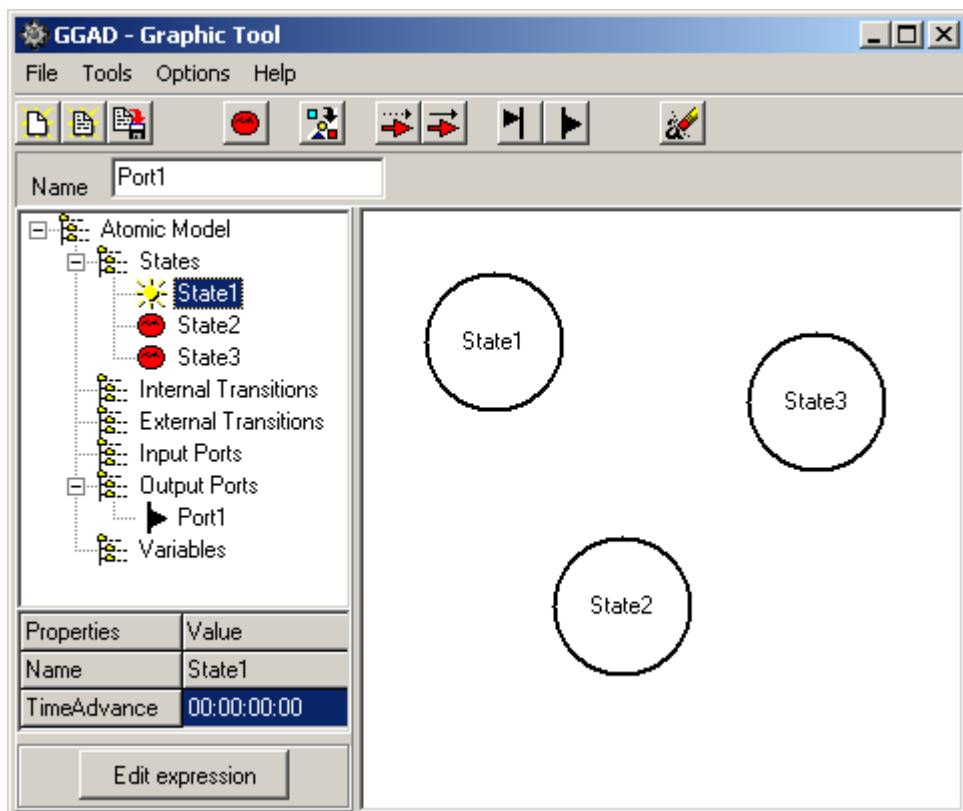


Figure 307: Adding internal transitions

In order to create an internal transition, follow these steps:

- 1- Select (ie. click on) the initial state in the drawing zone.



- 2- Select the final state in the drawing zone while pressing the Ctrl key.



3- If the internal transition is to have an associated output function, select the appropriate output port in the hierarchy zone.

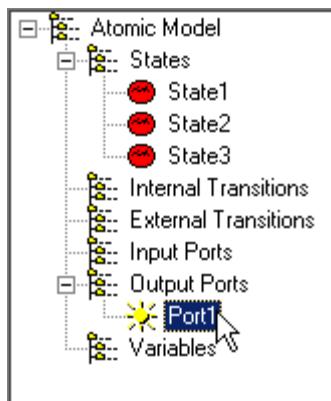


Figure 308: Select appropriate output port

4- Press the new internal transition button.

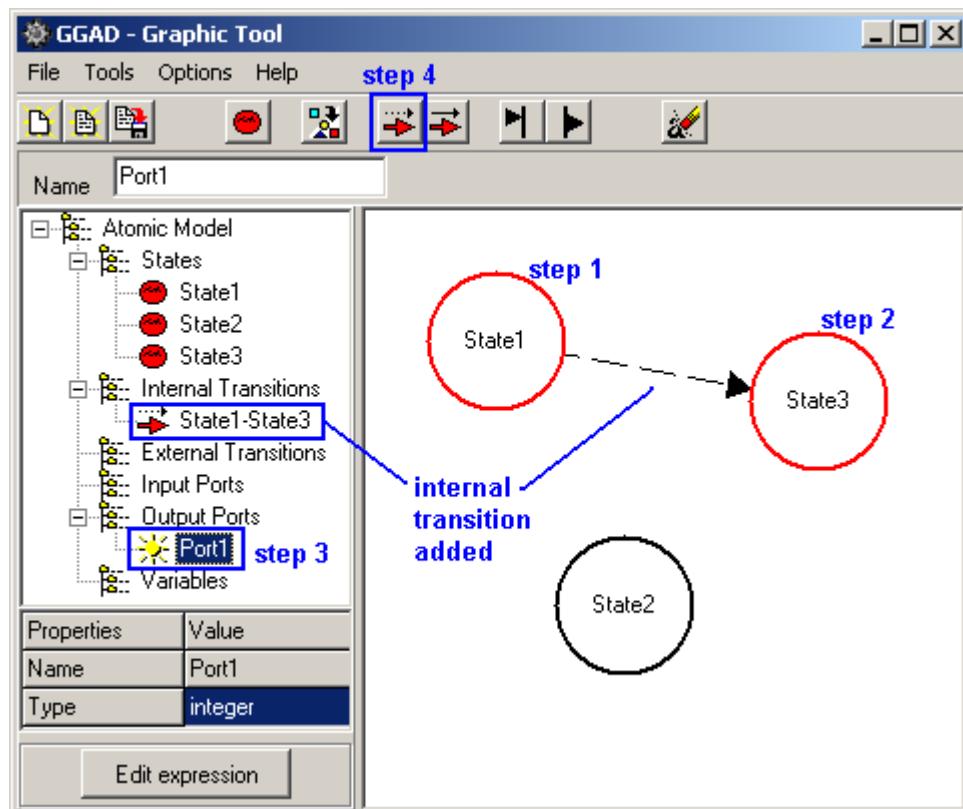


Figure 309: New internal transition

In order to add more output values in other ports:

- 1- Select the internal transition in the drawing zone or hierarchy zone.

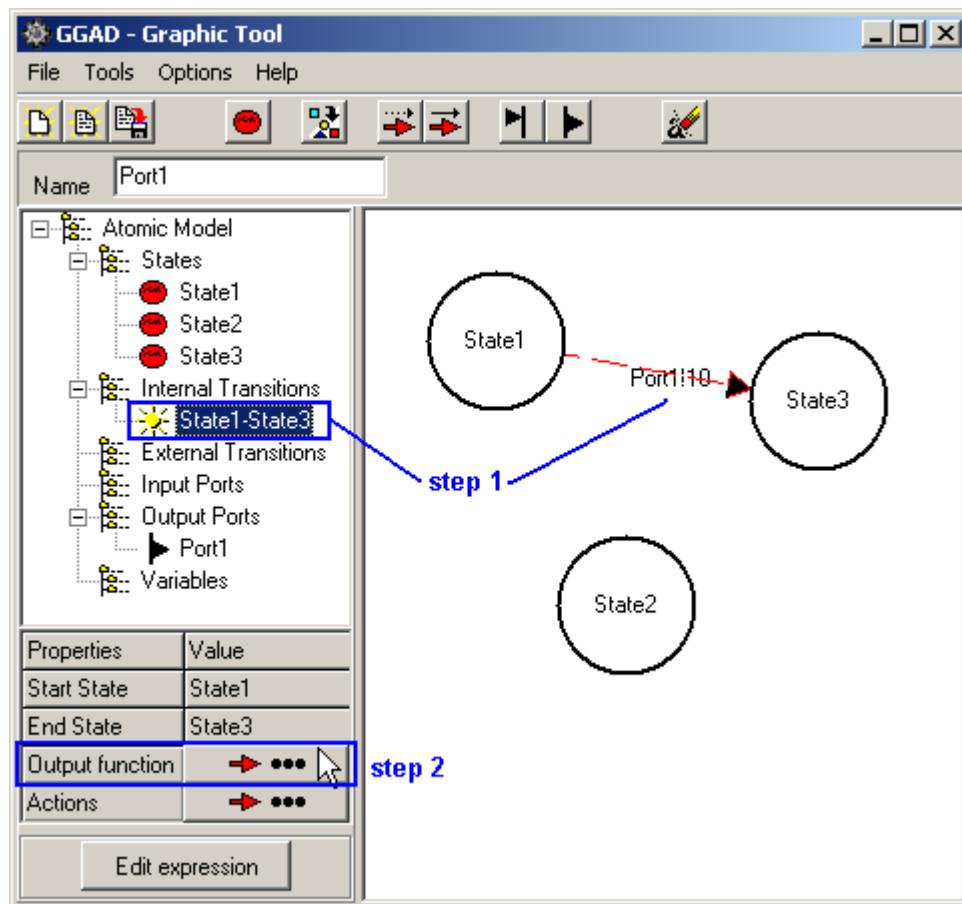


Figure 310: Selecting the internal transition on the hierarchy

- 2- In the properties zone, press the Output function button.

- 3- In the Edit output function dialog, fill the table with the ports and values to be sent to the ports. The specified Port must be an output port (ie. cannot be an input port). The specified value can be an expression composed of calls to built-in functions whose parameters are constants or variables of the model.

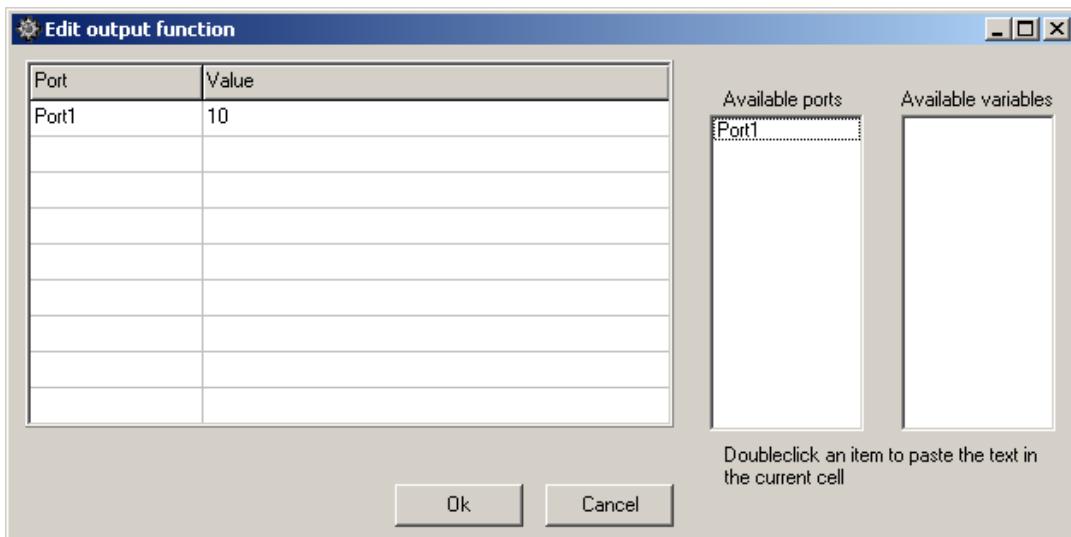


Figure 311:Entering values to be sent to the port

Add external transitions

An external transition can occur between 2 states, and is triggered by an external event/stimulus. An external event/stimulus is defined as the arrival of a particular value in a particular input port of the model.

Each external transition possesses a condition that must be fulfilled in order to be activated. This condition is defined as a function that can have as parameters: values received in input ports, state variables, or numeric constants. Such functions must to be supported by the CD++ simulator.

The current version of the CD++ simulator supports the following functions:

- Add(n1,n2): returns the sum of n1 and n2
- And(n1,n2): boolean, returns true if both parameters are real
- Any(p1): boolean, returns true if the port p1 receives a value in this event
- Between(n1,n2,n3): boolean, returns true if n2 is greater than n1 and less than n3
- Compare(n1,n2,n3,n4,n5): returns n3, n4, or n5 if n1 is greater than, equal to, or less than n2 respectively.
- Divide(n1,n2): returns $(n1 / n2)$
- Equal(n1,n2): returns true if n1 is equal to n2
- Minus(n1,n2): returns $(n1 - n2)$
- Multiply(n1,n2): returns $(n1 * n2)$
- NotEqual(n1,n2): returns true if n1 is different from n2 (ie. n1 does not equal n2)
- Pow(n1,n2): returns $(n1 ^ n2)$, ie. n1 to the power of n2
- Value(p1): returns the value received in port1 in this event

In order to create an external transition, follow these steps:

- 1- Select the initial state in the drawing zone.
- 2- Select the final state in the drawing zone while pressing the Ctrl key.
- 3- Select the first input port (if using a function that takes one or more ports).
- 4- Press the new external transition button.

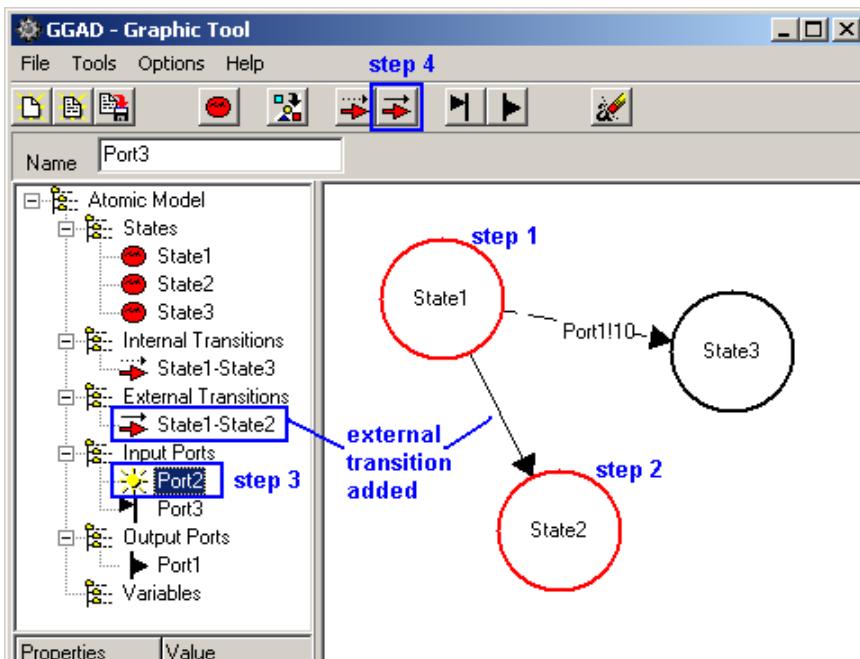


Figure 312: New external transitions

In order to modify the condition that triggers this transition:

- 1- Select the transition in the drawing zone or hierarchy zone.

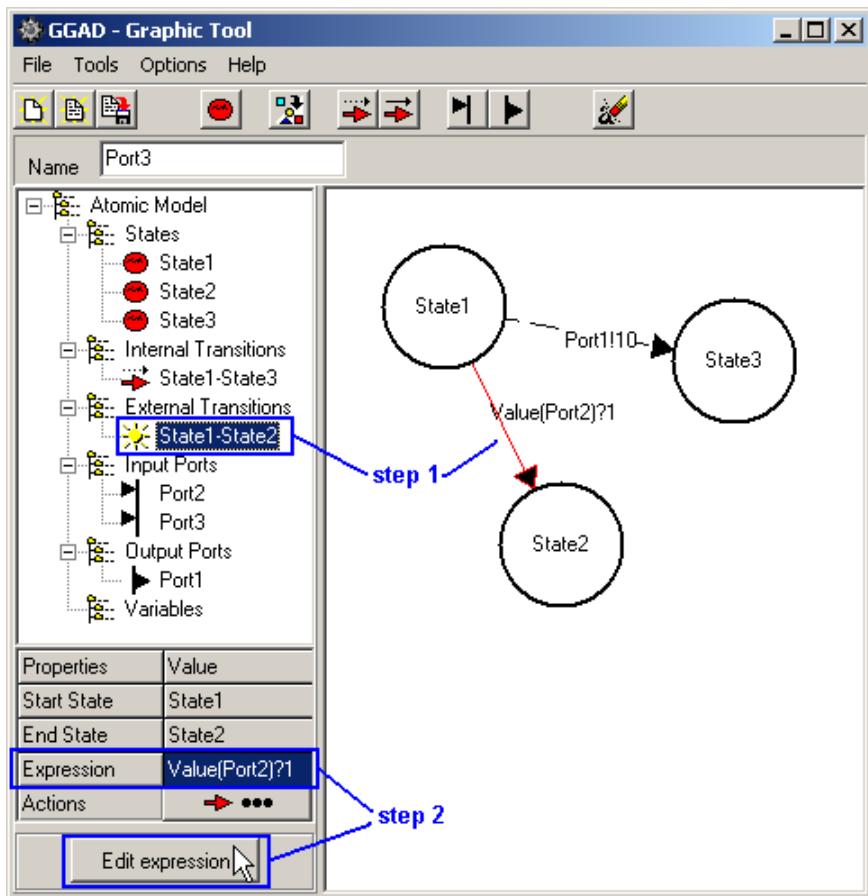


Figure 313: Selecting the transition in the hierarchy

2- Type the condition (that will trigger the selected transition) directly into the Expression field in the properties zone. Or, press the Edit expression button, and an Edit Expression dialog will appear.

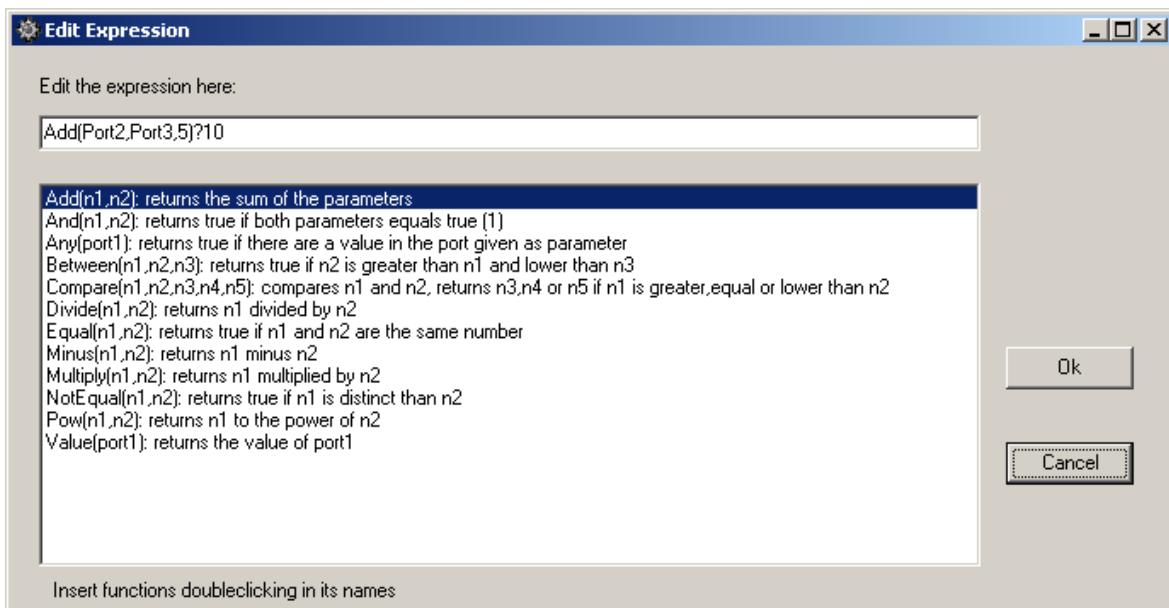


Figure 314: Edit Expression Dialog Box

In this example, from the field labeled “Edit the expression here:” in the Edit Expression dialog, the external transition will occur when a sum of 10 results from the addition of the values of Port2, Port3, and the constant 5.

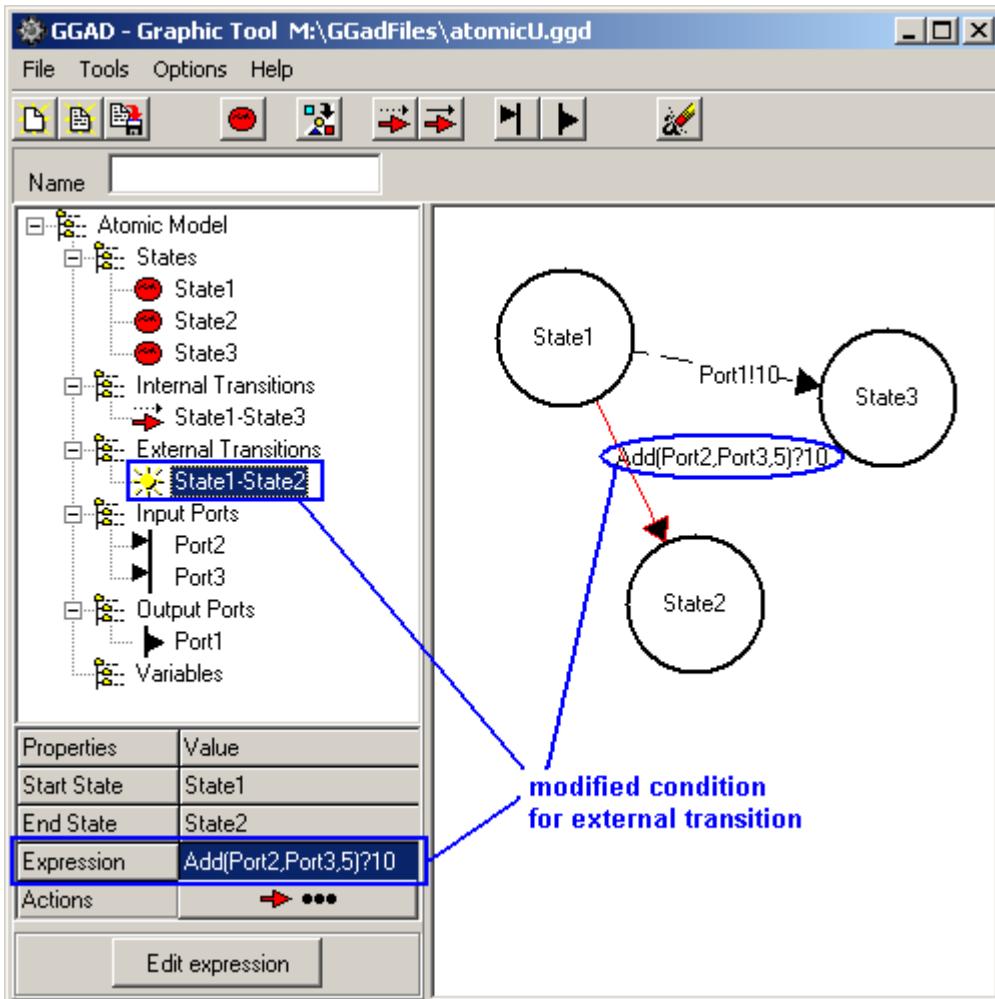


Figure 315: Entering an expression

Recall that the expression is a function with corresponding parameters. The parameters can be: constants, state variables, input ports, and other nested functions.

Add actions to the transitions

Both internal and external transitions can have an associated list of actions, where each action can be executed if certain conditions are satisfied. Actions allow values to be assigned to the state variables during execution of the model. The values assigned can be expressions that include: values present in the input ports of the transition, variables of the model, and constants.

In order to add an action to a transition, follow these steps:

1- Select (in the drawing zone or the hierarchy zone) the desired transition.

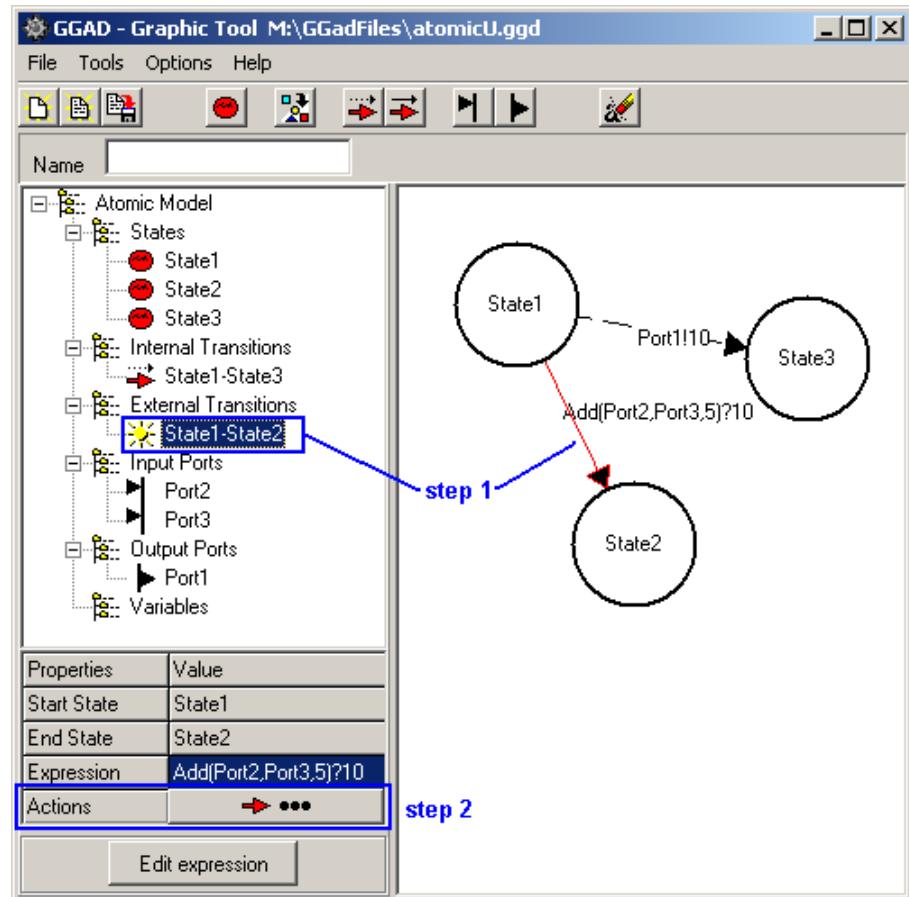


Figure 316: Selecting desired transition

2- In the properties zone, press the Actions button.

3- In the Edit Actions dialog, edit the table of allocations. The first column contains the variables, and the second column contains the numerical expressions that can include: variables, constants, or input ports.

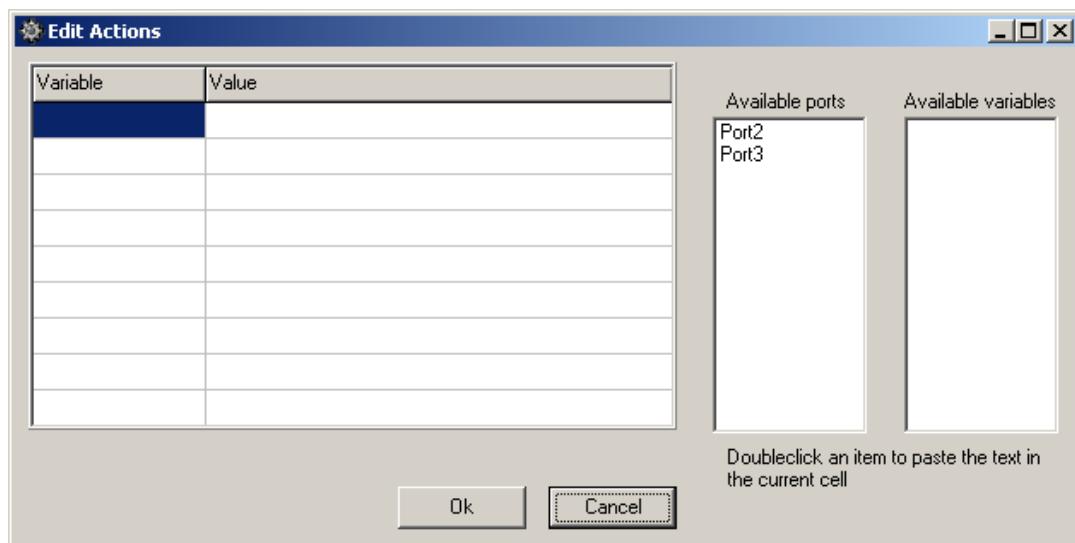


Figure 317: Editing actions

Delete objects

In order to delete an object, select the object in the drawing zone, and press the delete selected object button in the toolbar.

When an object is deleted, all the dependent objects are automatically deleted.

For example, if a state is deleted (eg. State3), the internal and external transitions that are connected to the state are also deleted (eg. internal transition from State1-State3).

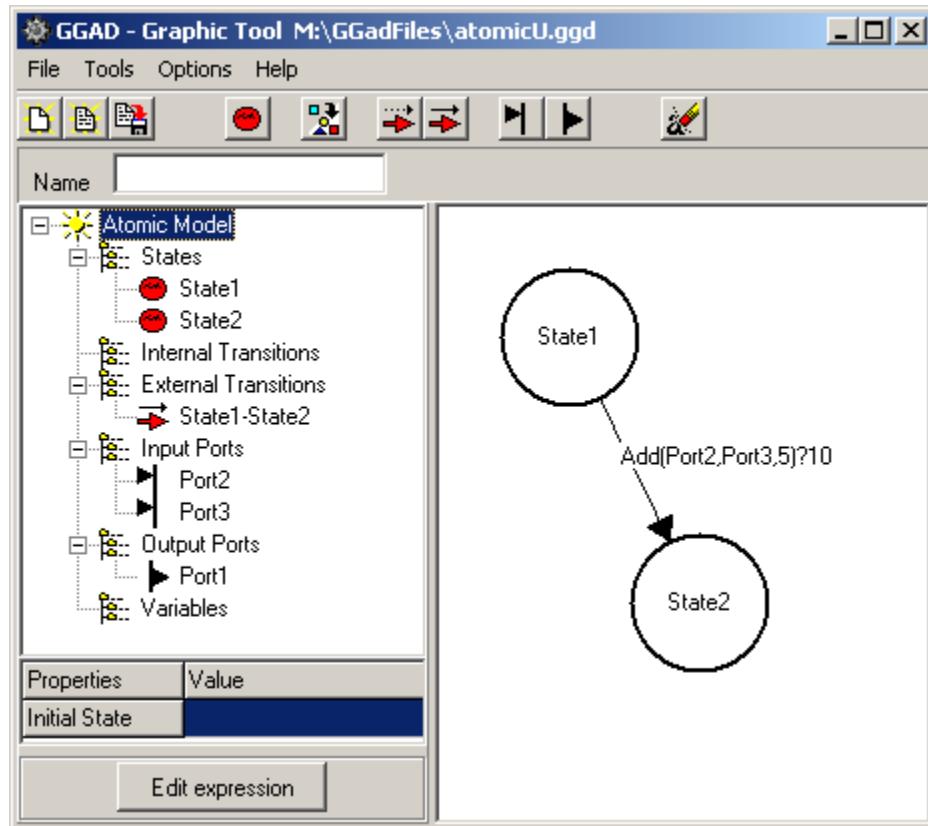


Figure 318: Deleting a state

Menus

In the Tools menu, there are commands for adding and removing objects, which correspond to the buttons in the toolbar.

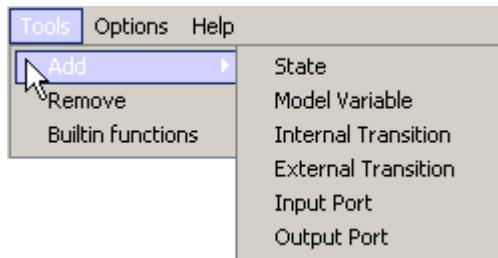


Figure 319: The Tools Menu

Furthermore, there is a “Builtin functions” command for editing the built-in functions available for the expressions of the external transitions.

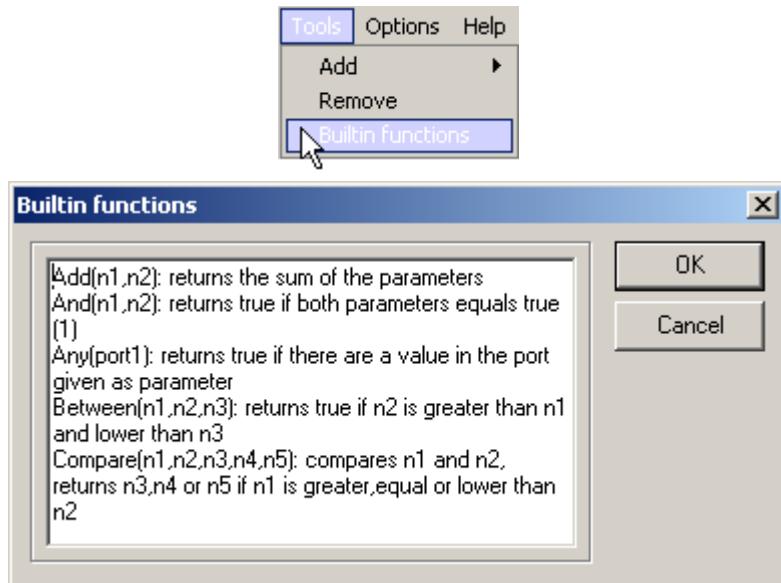


Figure 320: Builtin Functions Command

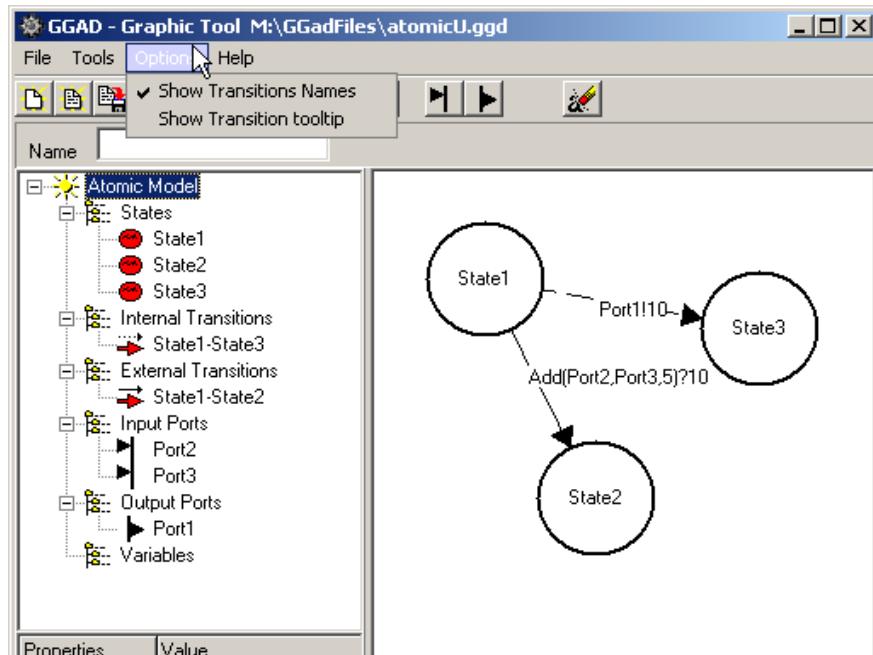
These functions should be implemented in the CD++ simulator.

When a function is added to the Builtin functions dialog, it will be available in the Edit Expression dialog (for use in External Transitions).

When a function is removed from the Builtin functions dialog, it will no longer be available in the Edit Expression dialog.

In the Options menu, there are 2 commands:

- **Show Transitions Names**: Activated by default, this command adds a description for each transition to the drawing zone. For external transitions, this description contains the condition for activating the transition. For internal transitions, this description contains the output port-value pair.
- **Show Transition Tooltip**: De-activated by default, this command provides the same information as the Show Transitions Names command, except in the form of a tooltip in the drawing zone when the mouse pointer is positioned above a transition.



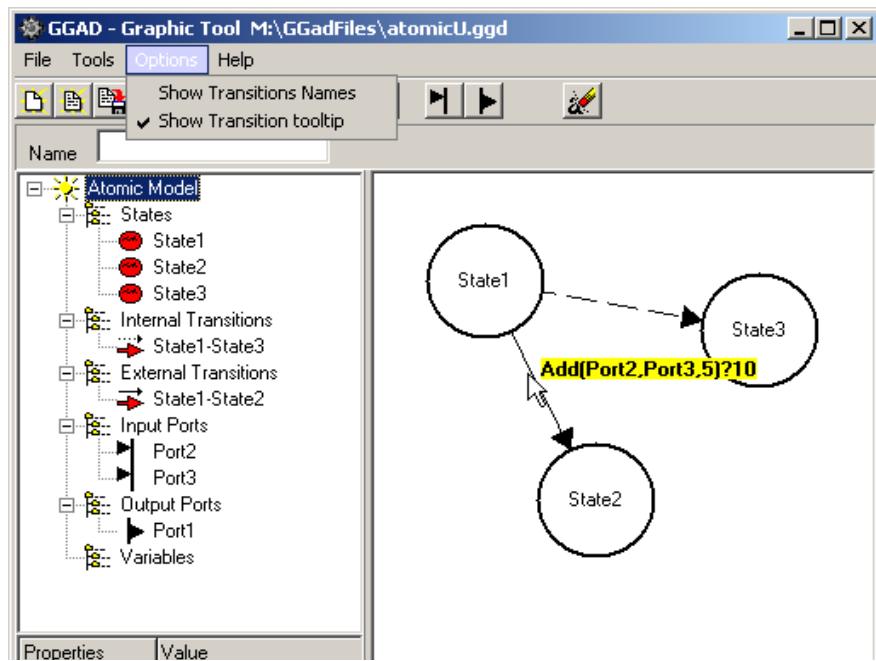


Figure 321: Options-> Show Transitions Names and Options-> Show Transition tooltip

Save the atomic model

Once the atomic model is complete, it can be saved in a file (*.ggd) and/or exported (as *.ma).

To save the atomic model, select “Save” or “Save As...” from the File menu.

To export the atomic model, select “Export to CD++” from the File menu.

Before exporting a model to .ma format, ensure the model is first saved in .ggd format (for atomic models).

In order to run simulations containing only an atomic model, first generate a coupled model that only contains the atomic model, and then export the coupled model.

9.2 Coupled GGAD (CGGAD) Models

Add ports

Note 1: A given port name can be used once by a model.



Figure 322: Warning for Blank or existing name

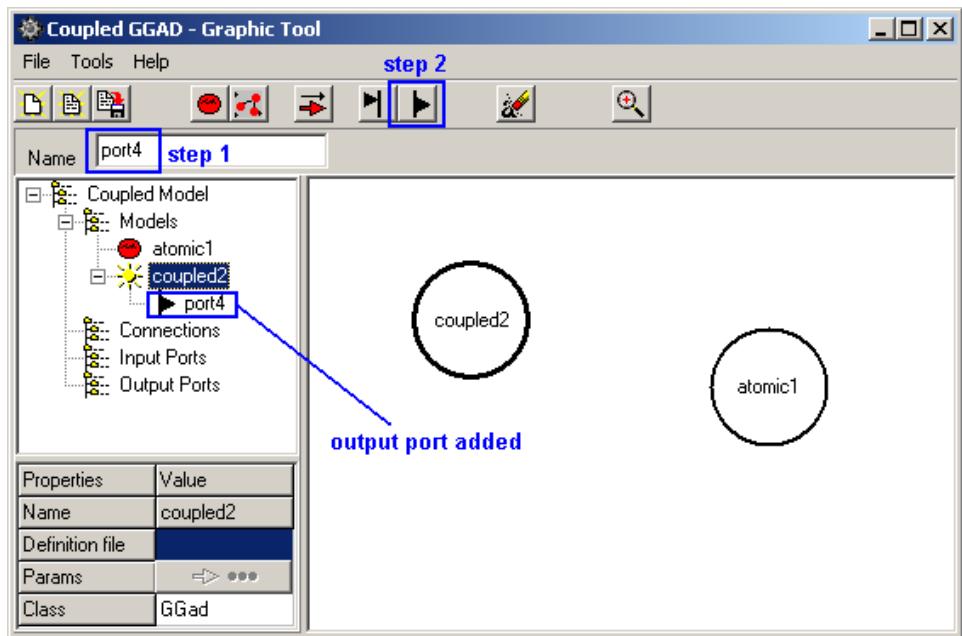
Note 2: Different models can use the same given port name.

BUG SCENARIO: Adding an input port to an inner coupled model.

CURRENT BEHAVIOUR: When adding an input port to an inner coupled model, a dialog appears as follows:



Note: This does not occur when adding an output port to an inner coupled model.



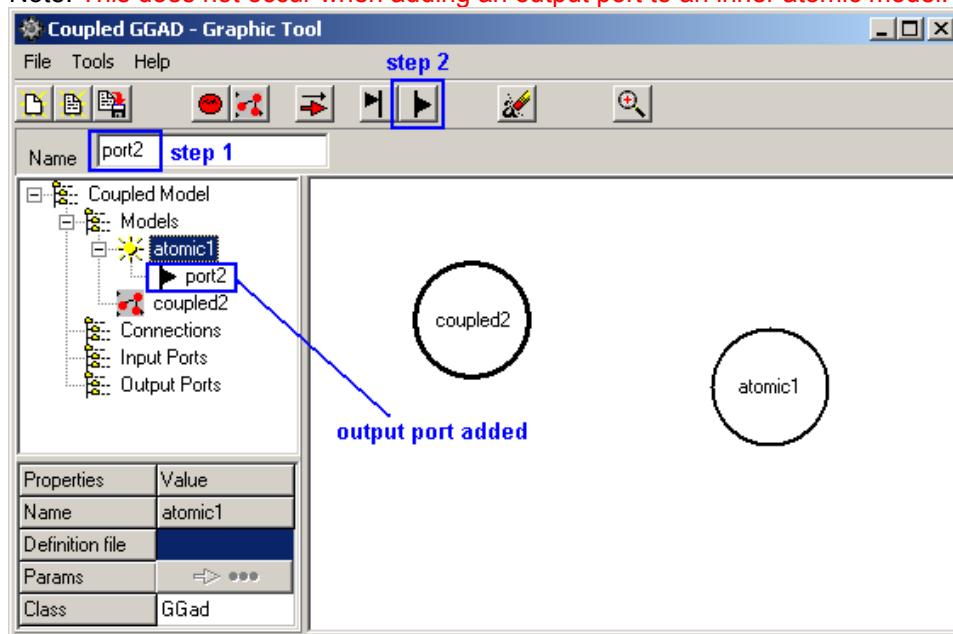
SUGGESTED BEHAVIOUR: Although unsure of correct behaviour, assume that dialog should also appear when trying to add an output port to an inner coupled model.

BUG SCENARIO: Adding an input port to an inner atomic model.

CURRENT BEHAVIOUR: When adding an input port to an inner atomic model, a dialog appears as follows:



Note: This does not occur when adding an output port to an inner atomic model.



SUGGESTED BEHAVIOUR: Although unsure of correct behaviour, assume that dialog should also appear when trying to add an output port to an inner atomic model.

Delete objects

SCENARIO: Deleting an inner model from the hierarchy zone.

CURRENT BEHAVIOUR: After selecting an inner (atomic or coupled) model from the hierarchy zone (and nothing is selected in the drawing zone), and pressing the Delete selected object button, the inner model is not deleted, and still appears in both the hierarchy and drawing zones.

ADDENDA TO CURRENT BEHAVIOUR: (The following behaviour is correct and currently functions

properly.) After selecting an inner (atomic or coupled) model from the drawing zone, and pressing the Delete selected object button, the inner model is deleted, and disappears from both the hierarchy and drawing zones. Thus, an inner model can only be deleted if first selected in the drawing zone.

SUGGESTED BEHAVIOUR: After selecting an inner (atomic or coupled) model from the hierarchy zone (and nothing is selected in the drawing zone), and pressing the Delete selected object button, the inner model should be deleted, and should disappear from both the hierarchy and drawing zones.

ADDENDA 2: The current behaviour is somewhat counterintuitive, since when adding ports (see section 3.4), selections in the hierarchy zone take precedent over selections in the drawing zone. (Or, perhaps the behaviour described in 3.4 is incorrect, and the current behaviour for 3.6 is correct.)

SUGGESTION: A single guideline should be used. For example, (1) selections in the hierarchy zone always take precedent over selections in the drawing zone, or vice versa. Or (2) if selections in the drawing zone do not match the selections in the hierarchy zone, then the command is deactivated. Or (3) if an object is selected in the drawing zone, then it should automatically be selected in the hierarchy zone, and vice versa.

SCENARIO: Deleting a connection/link from the hierarchy zone.

CURRENT BEHAVIOUR: After selecting a connection/link in the hierarchy zone, and pressing the Delete selected object button, the following dialog appears:



The connection/link is not deleted.

ADDENDA TO CURRENT BEHAVIOUR: (*The following behaviour is correct and currently functions properly.)* After selecting a connection/link in the drawing zone, and pressing the Delete selected object button, the connection/link is deleted, and disappears from both the hierarchy and drawing zones. Thus, a connection/link can only be deleted if first selected in the drawing zone.

SUGGESTED BEHAVIOUR: After selecting a connection/link in the hierarchy zone, and pressing the Delete selected object button, the connection/link should be deleted, and should disappear from both the hierarchy and drawing zones.

ADDENDA: See Addenda 2 and Suggestion from previous scenario.

9.3 GGAD Atomic Models

Delete objects

SCENARIO: Deleting an state from the hierarchy zone.

CURRENT BEHAVIOUR: After selecting a state from the hierarchy zone (and nothing is selected in the drawing zone), and pressing the Delete selected object button, the state is not deleted, and still appears in both the hierarchy and drawing zones.

ADDENDA TO CURRENT BEHAVIOUR: (*The following behaviour is correct and currently functions properly.*) After selecting an state from the drawing zone, and pressing the Delete selected object button, the state is deleted, and disappears from both the hierarchy and drawing zones. Thus, a state can only be deleted if first selected in the drawing zone.

SUGGESTED BEHAVIOUR: After selecting an state from the hierarchy zone (and nothing is selected in the drawing zone), and pressing the Delete selected object button, the state should be deleted, and should disappear from both the hierarchy and drawing zones.

SUGGESTION: If an object is selected in the drawing zone, then it should automatically be selected in the hierarchy zone, and vice versa.

BUG SCENARIO: Deleting a state variable from the hierarchy zone.

CURRENT BEHAVIOUR: After selecting a single variable from the hierarchy zone, and pressing the Delete selected object button, the single variable is deleted, and disappears from the hierarchy zone. **However, the rest of the variables disappear from the hierarchy zone.**

ADDENDA TO CURRENT BEHAVIOUR: It is possible to add a new state variable that has the same name as the previously deleted single variable. However, it is not possible to add a new state variable that has the same name as any of the variables that disappeared.

SUGGESTED BEHAVIOUR: After selecting a single variable from the hierarchy zone, and pressing the Delete selected object button, the single variable should be deleted, and should disappear from the hierarchy zone. The rest of the variables should not disappear, and should remain visible in the hierarchy zone.

BUG SCENARIO: Deleting a state from the drawing zone.

CURRENT BEHAVIOUR: After selecting a state from the drawing zone, and pressing the Delete selected object button, the state is deleted, and disappears from both the hierarchy and drawing zone. **However, all the variables disappear from the hierarchy zone.**

ADDENDA TO CURRENT BEHAVIOUR: It is not possible to add a new state variable that has the same name as any of the variables that disappeared.

SUGGESTED BEHAVIOUR: After selecting a state from the drawing zone, and pressing the Delete selected object button, the state should be deleted, and should disappear from both the hierarchy and drawing zone. The variables in the hierarchy zone should be unaffected.

BUG SCENARIO: Deleting a port from the hierarchy zone.

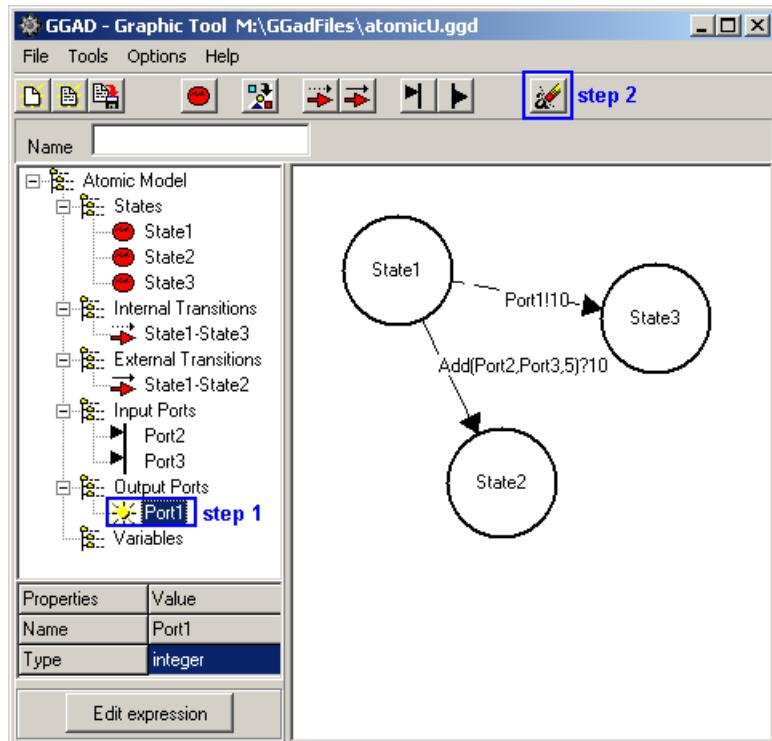
CURRENT BEHAVIOUR: After selecting a port from the hierarchy zone, and pressing the Delete selected object button, the port is deleted, and disappears from the hierarchy zone. **However, all the variables disappear from the hierarchy zone.**

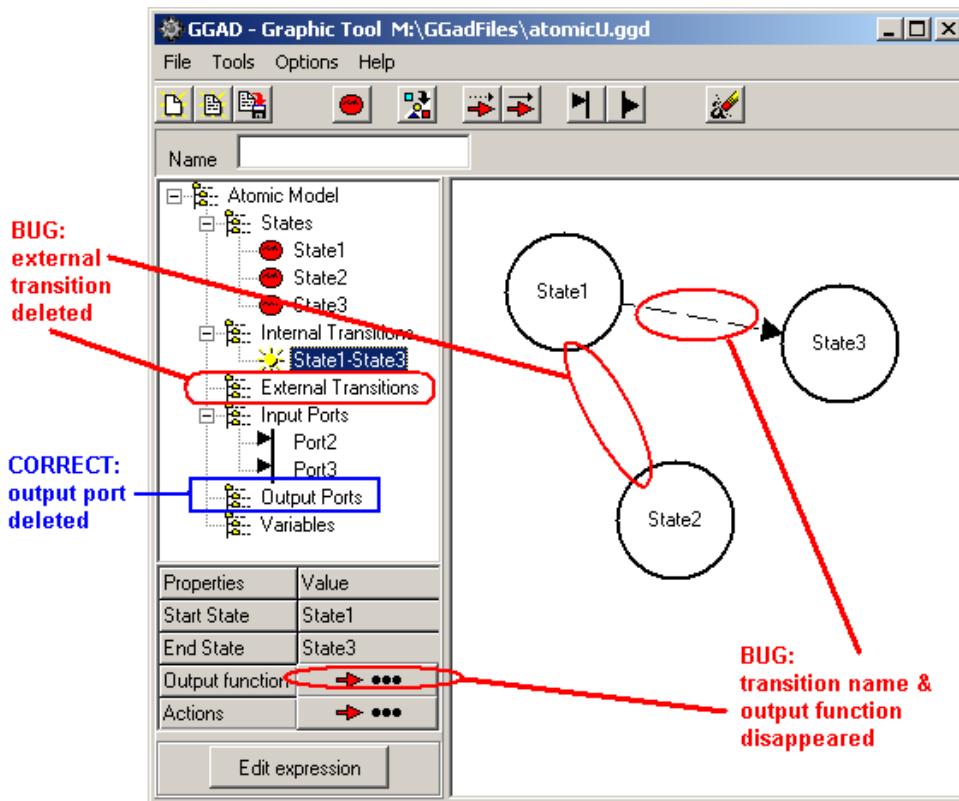
ADDENDA TO CURRENT BEHAVIOUR: It is not possible to add a new state variable that has the same name as any of the variables that disappeared.

SUGGESTED BEHAVIOUR: After selecting a port from the hierarchy zone, and pressing the Delete selected object button, the port should be deleted, and should disappear from the hierarchy zone. The variables in the hierarchy zone should be unaffected.

BUG SCENARIO: Deleting an output port from the hierarchy zone.

CURRENT BEHAVIOUR: After selecting an output port from the hierarchy zone, and pressing the Delete selected object button, the port is deleted, and disappears from the hierarchy zone. However, the transition names also disappear from the drawing zone, and external transitions appear to be deleted (from both the hierarchy and drawing zones).





SUGGESTED BEHAVIOUR: Other components should be unaffected when deleting output port.

SCENARIO: Deleting a transition from the hierarchy zone.

CURRENT BEHAVIOUR: After selecting an internal or external transition from the hierarchy zone (and nothing is selected in the drawing zone), and pressing the Delete selected object button, the following dialog appears:



The transition is not deleted.

ADDENDA TO CURRENT BEHAVIOUR: (*The following behaviour is correct and currently functions properly.*) After selecting an internal or external transition in the drawing zone, and pressing the Delete selected object button, the transition is deleted, and disappears from both the hierarchy and drawing zones. Thus, a transition can only be deleted if first selected in the drawing zone.

SUGGESTED BEHAVIOUR: After selecting a transition in the hierarchy zone (and nothing is selected in the drawing zone), and pressing the Delete selected object button, the transition should be deleted, and should disappear from both the hierarchy and drawing zones.

SUGGESTION: If an object is selected in the drawing zone, then it should automatically be selected in the hierarchy zone, and vice versa.

10 Appendix A - CD++Builder – Installation Guide for Windows

This section provides users with a step-by-step instruction set about how to install the CD++Builder plug in, in a Windows environment.

Installation

You will need three main programs to run this plug-in in Windows:

- (a) **Java JRE** (Java Runtime Environment) enables Java-based applications, such as Eclipse, to run.
- (b) **Eclipse** is a software development workbench. It provides a plug-in-based framework, allowing greater ease to create and utilize software tools. The CD++ Builder plug-in will be incorporated into Eclipse.
- (c) **Cygwin** is a Unix emulator for Windows. Since the CD++ toolkit was originally developed in the Linux/Unix platform, a Unix emulator is required to run the toolkit's binary files in Windows. (A Unix emulator is not required to run CD++ in Linux/Unix.)

STEP 1: Installing Java JRE

- 1) To download the Windows version of the Java JRE, first go to <http://java.sun.com/j2se/1.4.2/download.html> .
- 2) Under the heading "J2SE v 1.4.2_05 JRE includes the JVM technology", click on **Download J2SE JRE**.
- 3) **Accept** the License Agreement, and click **Continue**. Depending on your platform (ie. Windows), right-click the appropriate version of J2SE v 1.4.2_05 JRE, and save the executable (.exe) file to a known folder. *The Java JRE should now be downloaded.*
- 4) To install the Windows version of the Java JRE, first run the installer, ie. from the known folder, double-click the executable (.exe) file.
- 5) In the Welcome window, click **Next**.
- 6) Accept the License Agreement, then click **Next**.
- 7) Select "**Typical Installation**", then click **Next**.
- 8) Click **Finish**. *The installation will now proceed.*
- 9) A "Java Web Start" icon will be placed on the desktop. *The Java JRE should now be installed.*

STEP 2: Installing Eclipse SDK

Note: Proceed with installing Eclipse only if the Java JRE has been installed successfully.

- 1) To download the Windows version of the Eclipse SDK, first go to <http://download.eclipse.org/downloads> .
- 2) Under the heading "Latest Releases", click on the newest of the **2.x** versions of builds.
- 3) Under the heading "Eclipse SDK", depending on your platform (ie. Windows), click on the **http** link of the appropriate version of Eclipse SDK v 2.x, and save the specified .zip file to a known folder. *The Eclipse SDK should now be downloaded.*
- 4) To install the Windows version of the Eclipse SDK, first, from the known folder, double-click the .zip file. In the Winzip introductory window, click **I Agree**.

5) From the toolbar, select **Actions > Extract ...**. In the "Extract To:" drop-down menu, choose **C:**. Select "**All Files**", as well as "**Use Folder Names**". Click **Extract**. *The Eclipse SDK should now be extracted.* Close Winzip.

Note: Do not include spaces in the Eclipse nor project directory names.

6) From C:\eclipse, double-click **eclipse.exe**. After Eclipse has successfully opened, close Eclipse. *The Eclipse SDK should now be installed.*

7) Make a shortcut to C:\eclipse\eclipse.exe on the desktop and start menu, ie. from C:\eclipse, right-click **eclipse.exe**, select **Create Shortcut**, then drag "Shortcut to eclipse.exe" to the desktop.

- To download the Windows version of the Eclipse SDK, first go to <http://www.eclipse.org/downloads/download.php?file=/eclipse/downloads/drops/R-3.1.2-200601181600/eclipse-SDK-3.1.2-win32.zip>.

Choose the mirror for downloading by clicking on one of the offered

STEP 3: Installing Cygwin

NOTE: To beginning installation for Cygwin you will need ADMINISTRATOR RIGHTS to the computer in which you are installing Cygwin.

1. To download the Cygwin setup file, first go to <http://www.sce.carleton.ca/faculty/wainer/wbgraf/distrib/cygwin.zip>, ask Professor Wainer for login information] and save the specified .zip file to a known folder.

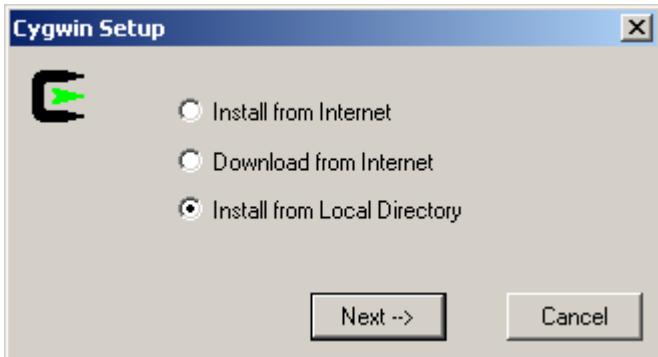
Note: The tools may not function properly with other versions of Cygwin.

2. To install Cygwin, first, from the known folder, double-click the .zip file. In the Winzip introductory window, click **I Agree**.
3. From the toolbar, select **Actions > Extract ...**. In the "Extract To:" drop-down menu, choose **C:**. Select "**All Files**", as well as "**Use Folder Names**". Click **Extract**. *The Cygwin setup file should now be extracted.* Close Winzip.
4. Go to the folder where you extracted cygwin and into the directory called **latest**. Once there, make a new folder called **gdb**.
5. Next, go to <ftp://mirrors.rcn.net/mirrors/sources.redhat.com/cygwin/release/gdb/> and download the file called **gdb-20041228-3.tar.bz2** (3,747 Kb)
6. After that, copy this file in to ...**cygwin\latest\gdb**, i.e the new folder that you made in the **latest** directory.

7. From C:\cygwin, double-click **setup.exe**. In the Cygwin Setup window, click **Next** (as seen in the following figure).



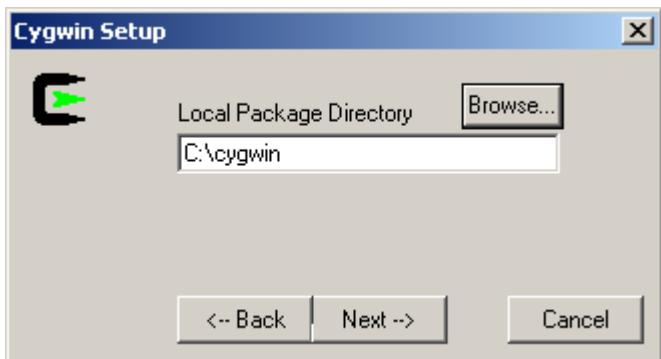
8. Select "**Install from Local Directory**", then click **Next**.



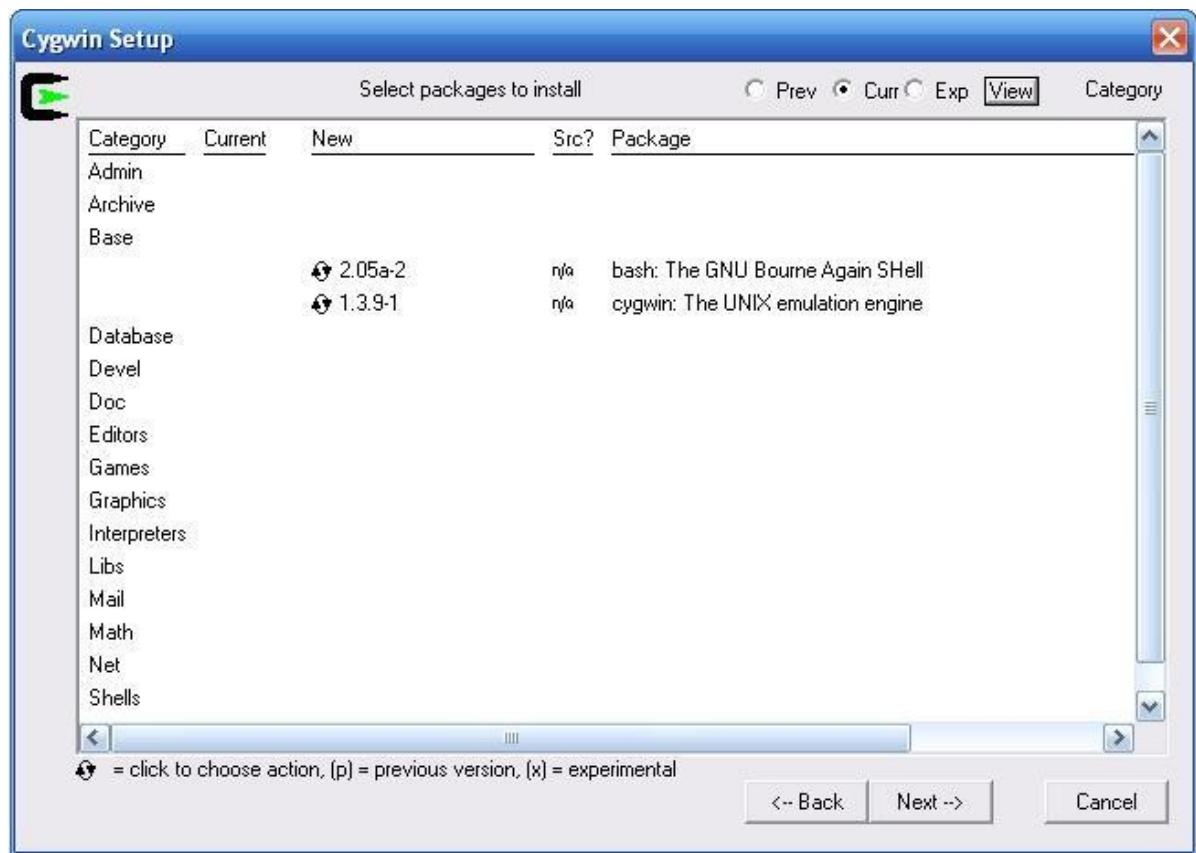
9. In the "Select install root directory", type in **C:\cygwin**. For "Default Text File Type:", select "**Unix**". For "Install For:", select "**All**". Click **Next**.



10. In the "Local Package Directory:", type in **C:\cygwin** (or any existing empty folder). *The local package directory will contain the temporary installation files for Cygwin.* Click **Next**.

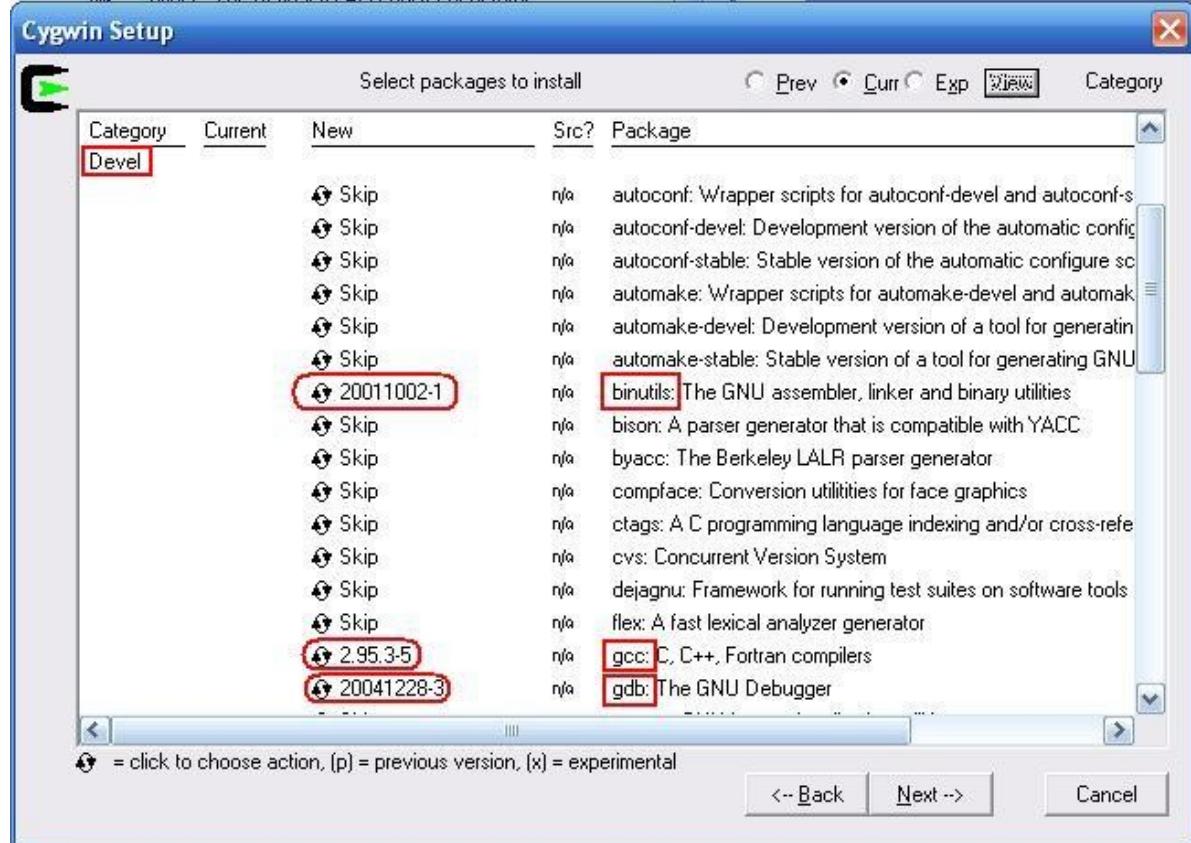


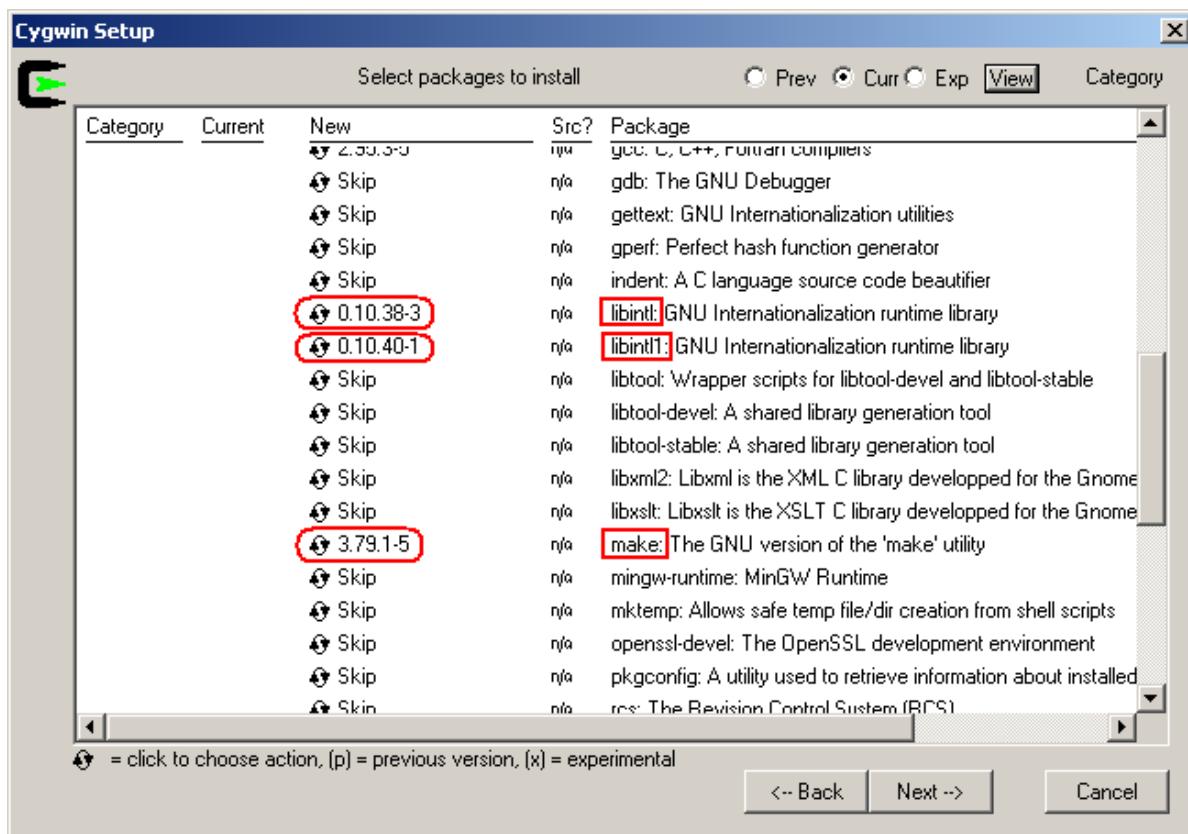
11. First make sure that the very next screen you see is the following,



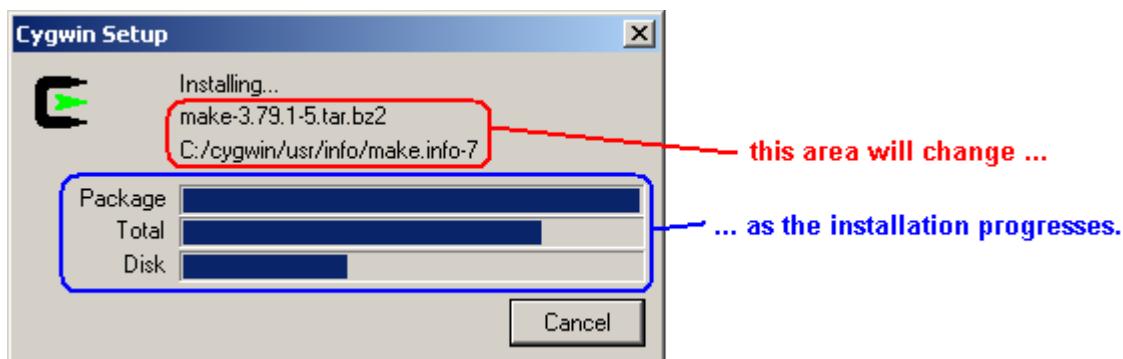
If not then immediately go to 10.1.5.2 'Simu.exe: No such file or directory; Simu was was not created!' and carry out the steps mentioned there before resuming the cygwin installation.

12. Under the "Category" column, click on the word **Devel**. Under the "New" column, click on the word **Skip** for each of these "Packages": **binutils**, **gcc**, **gdb**, **libintl**, **libintl1**, **make**. The word Skip should be replaced by the version number of the "Package", as seen in the following figures. Click **Next**.

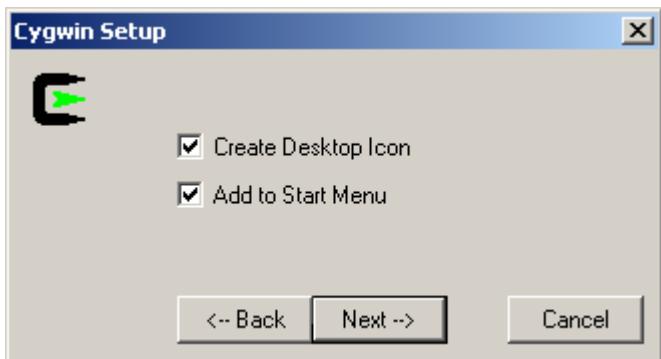




The installation will now proceed.



13. When prompted, create shortcuts on the desktop and start menu. Click **Next**.



Cygwin should now be installed.

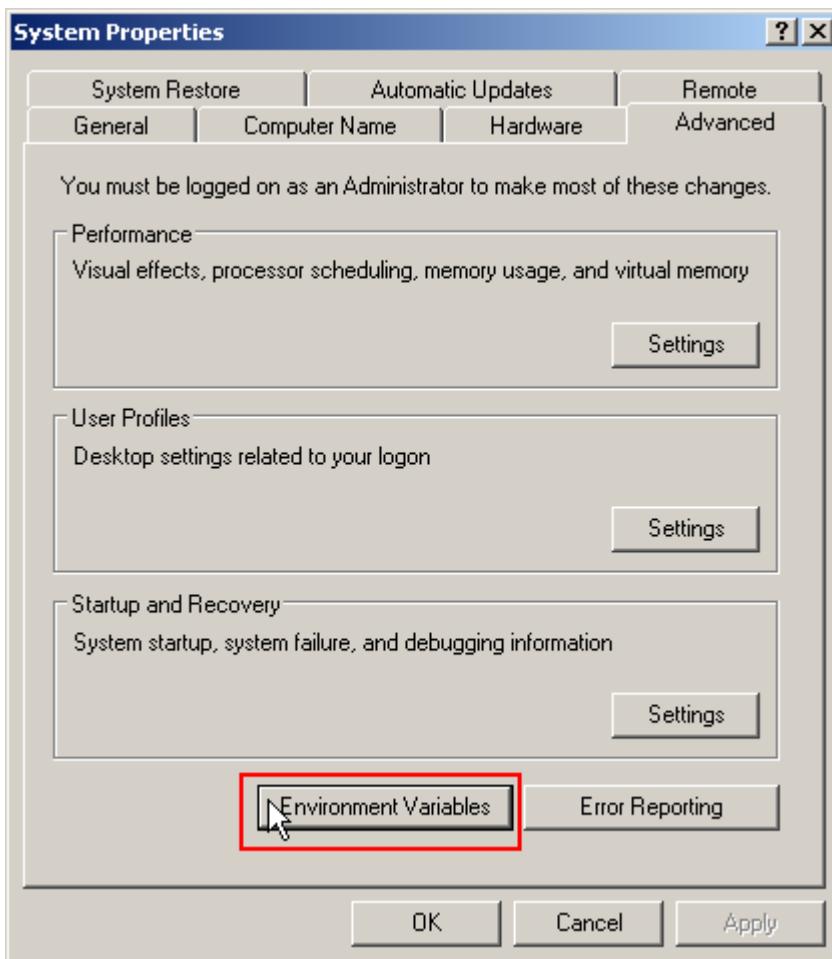


14. After clicking OK go to: <http://chat.carleton.ca/~omkanwar/cygwinBin.rar> [or ask Professor Gabriel Wainer (gwainer@sce.carleton.ca) for the CD labeled "Jing and Omair- Summer 2005" because it contains this file too] and download the file called **cygwin-Bin.rar**.
15. Next, unpack the contents of the file using WinRar, and then go to,
...cygwinBin→bin(gcc and gdb working)
16. Rename **bin(gcc and gdb working)** to **bin** and copy and paste it into the folder where you installed cygwin so that it overwrites the original **bin** in the cygwin directory. If an overwrite message pops up just click | **Yes to All** | everytime as shown below.

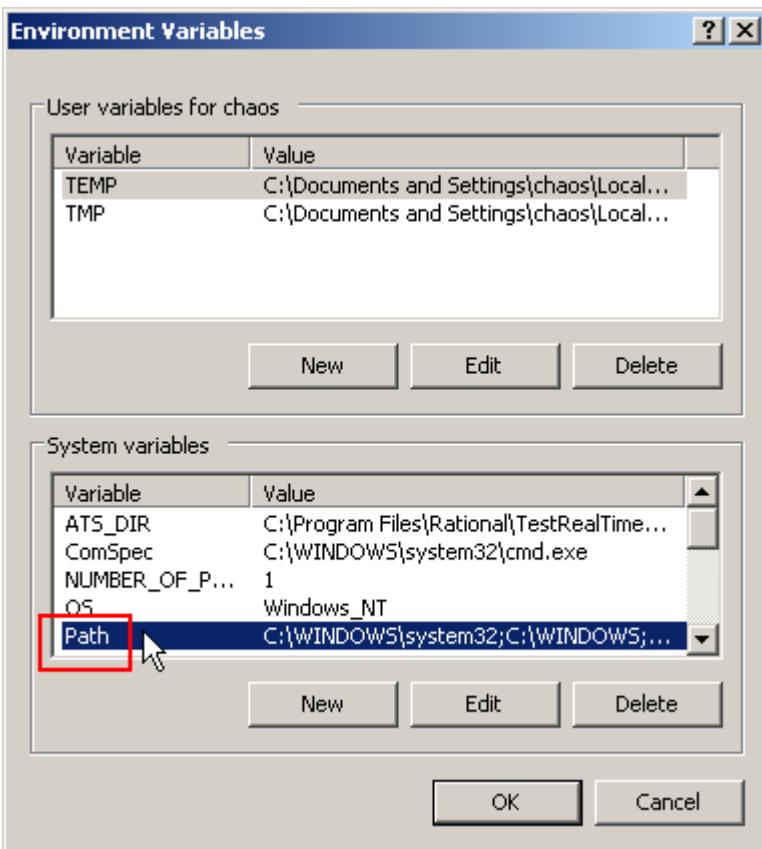


17. To properly configure Cygwin, first, go to **Control Panel > System**. (Note for Windows XP users: double-click **My Computer**, within "System Tasks", click **View System Information**.)

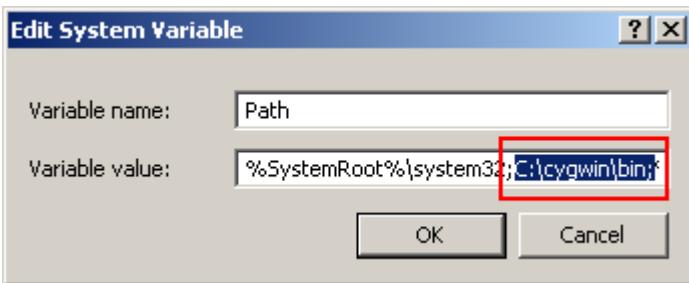
18. In the System Properties window, within the "Advanced" tab, click on the **Environment Variables** button.



19. In the Environment Variables window, within "System variables", double-click the **Path** variable.



20. In the Edit System Variable window, ensure the "Variable name:" is Path. At the end of "Variable value:", add **C:\cygwin\bin;**. Do not forget to include the semicolon (";"). (Note for Windows XP users: in "Variable value:", add **C:\cygwin\bin;** between **%SystemRoot%\system32** and **%SystemRoot%**. This should result in the following order: **%SystemRoot%\system32; C:\cygwin\bin; %SystemRoot%; ...**)

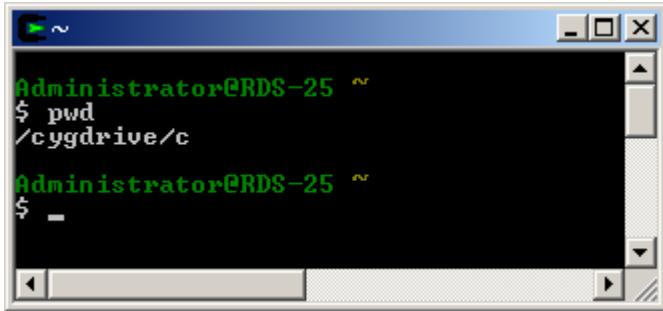


21. Click OK to close the Edit System Variable window. Click OK to close the Environment Variables window. Click OK to close the System Properties window.

Note: After this step is complete, it is advisable to log out and re-login.

22. Create a folder **C:\tmp** (not **C:\temp**), ie. from **C:**, choose **File > New > Folder**, and rename the folder **tmp** (not temp). Give read/write access to **C:\tmp**, ie. from **C:**, right-click **tmp**, select **Properties**, within the "General" tab, listed under "Attributes:", un-click **Read-only**.

23. Double-click the **Cygwin** icon on the desktop. Type **pwd**, and hit **Enter**. If
 24. ./cygdrive/c is output, *Cygwin should now be installed and configured*, so proceed to **STEP 4: Installing the CD++ Builder Plugin.**



```
Administrator@RDS-25 ~
$ pwd
/cygdrive/c
Administrator@RDS-25 ~
$ -
```

25. If /cygdrive/c is not output, open **C:\cygwin\etc\profile** file with a text editor, ie. from C:\cygwin\etc, double-click **profile**, in the Windows window, select "Select the program from a list", click **OK**, in the Open With window, select **WordPad**, click **OK**.
26. Under the line: **USER=`id -un`**
 type the following: **HOME="/cygdrive/c"**
export HOME
27. Save the modified **profile** file, ie. from the toolbar, select **File > Save** . Close all Cygwin windows. Go to step 12).
28. To finish up, and to make sure that every feature of the plugin works normally check that you have the file <**cat.exe**> in the folder <**C:\cygwin\bin**>(if cygwin is installed in **C:**) if not then go to:
<http://programming ccp14.ac.uk/ftp-mirror/programming/cygwin/pub/cygwin/latest/textutils/>
 and download the file called **textutils-2.0.21-1.tar.tar** (495 Kb in size).
29. Unpack the file you downloaded using WinRar and go to **user→bin→cat.exe** .
30. Copy **cat.exe** in to your cygwin bin folder. For instance, if cygwin has been installed in the C: drive, then copy **cat.exe** over to **C:\cygwin\bin** to completely install cygwin.

12 To finish up, and to make sure that every feature of the plugin works normally check that you have the file <**cat.exe**> in the folder <**C:\cygwin\bin**>(if cygwin is installed in **C:**) if not then go to:

<http://programming ccp14.ac.uk/ftp-mirror/programming/cygwin/pub/cygwin/latest/textutils/> and download the file called **textutils-2.0.21-1.tar.tar** (495 Kb in size).

13 Unpack the file you downloaded using WinRar and go to "**user→bin→cat.exe**".

14 Copy **cat.exe** in to your cygwin bin folder. For instance, if cygwin has been installed in the

C: drive, then copy **cat.exe** over to **C:\cygwin\bin** to completely install cygwin.

- Is zcat.exe the same as cat.exe?

- These steps (12 – 14) are not mentioned in the Installation manual. The software seems to be working fine without the cat.exe file.

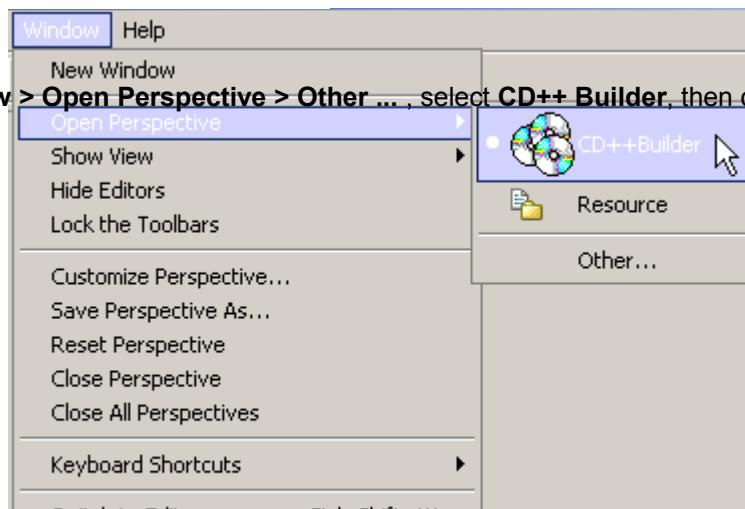
STEP 4: Installing the CD++ Builder Plugin

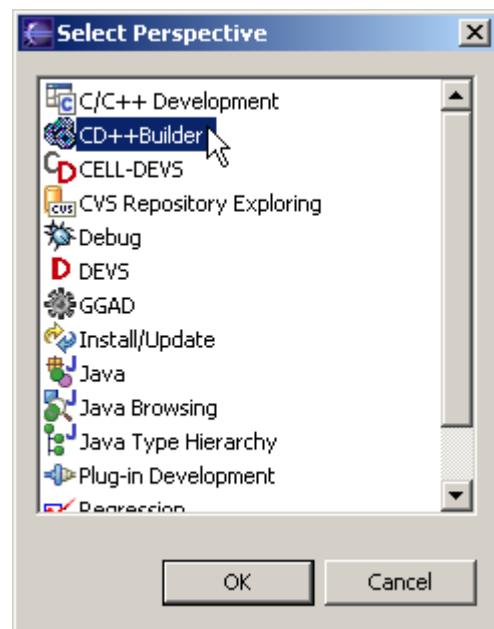
Note: Proceed with installing the CD++ Builder Plugin only if the Java JRE, Eclipse SDK, and Cygwin have been installed successfully.

- 1) To download the CD++ Builder Plugin, first go to <http://www.sce.carleton.ca/faculty/wainer/wbgraf/distrib/CD++builderV1.2.zip>, and save the specified .zip file to a known folder.
- 2) To install the CD++ Builder Plugin, first, from the known folder, double-click the .zip file. In the Winzip introductory window, click **I Agree**.
- 3) From the toolbar, select **Actions > Extract ...**. In the "Extract To:" drop-down menu, choose **C:\eclipse\plugins**. Select "**All Files**", as well as "**Use Folder Names**". Also select "**Overwrite Existing Files**". Click **Extract**. *The CD++ Builder plugin files should now be extracted.* Close Winzip.
- 4) Double-click the **Eclipse** icon on the desktop.

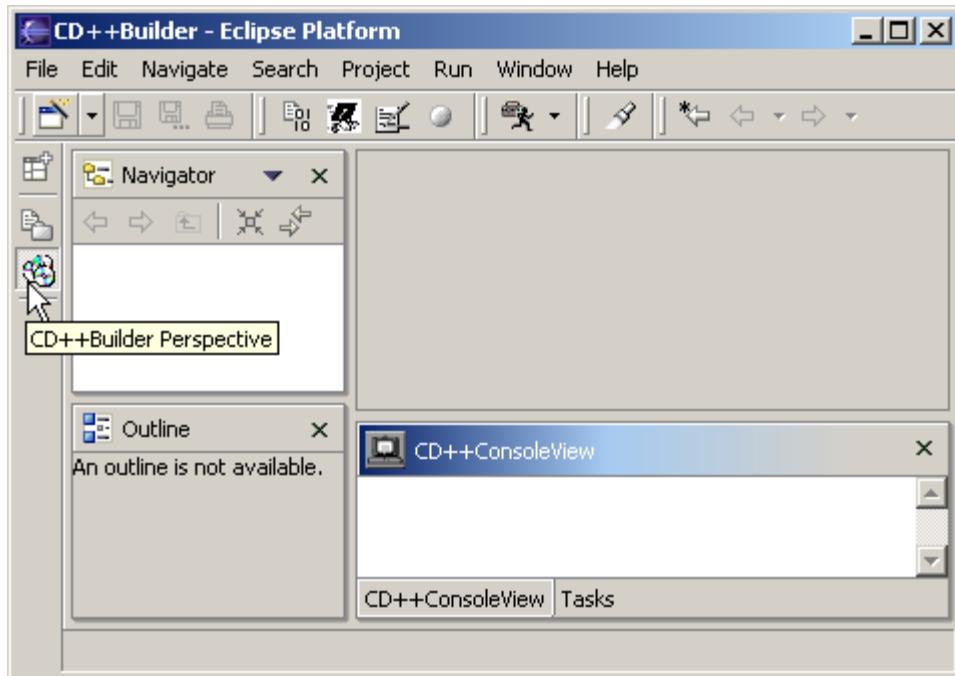
From the toolbar, select **Window > Open Perspective > CD++ Builder**.

Or, select **Window > Open Perspective > Other ...**, select **CD++ Builder**, then click **OK**.





The Eclipse perspective should resemble the following figure. *The CD++ Builder plugin files should now be installed.*



The next step involves installing the **CDT plugin** for Eclipse, which allows one to debug C++ code and for our purpose it would help in debugging CD++ models written in C++.

Download the CDT plugin from :

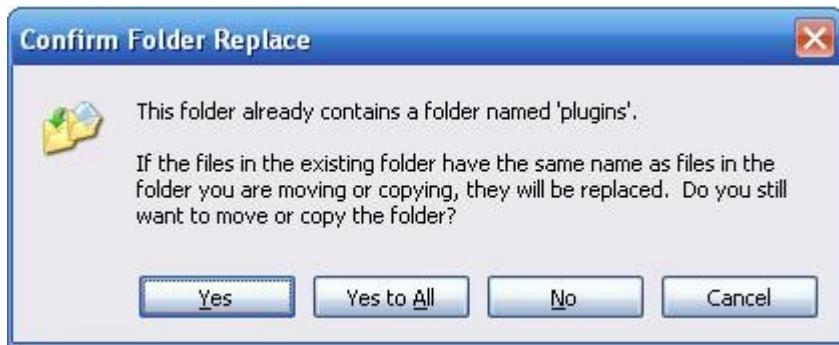
<http://download.eclipse.org/tools/cdt/releases/dist/cdt-full-1.2-win32.zip>

Unzip it. If, for e.g., CDT has been unzipped in the C: then follow the steps given below.

Copy the folder **C:\cdt-full-1.2-win32\eclipse\features** to **C:\eclipse** (if eclipse has been installed in C:). Click **Yes To All** when the following dialog box pops up:



Finally, copy the folder **C:\cdt-full-1.2-win32\eclipse\plugins** to **C:\eclipse** (if eclipse has been installed in C:). Click **Yes To All** when the following dialog box pops up:



The CDT should now be installed and ready to use.

Examples

From the first chapter of the user guide/manual, download and run the examples: life2D, and ATM.

10.1.5 Troubleshooting

10.1.5.1 Internal File Corrupted

To have a sucessful install, the ATM model representing Cell Devs should compile without errors (by pressing the build button). However this may not be the case as seen in the figure below (Figure 323):

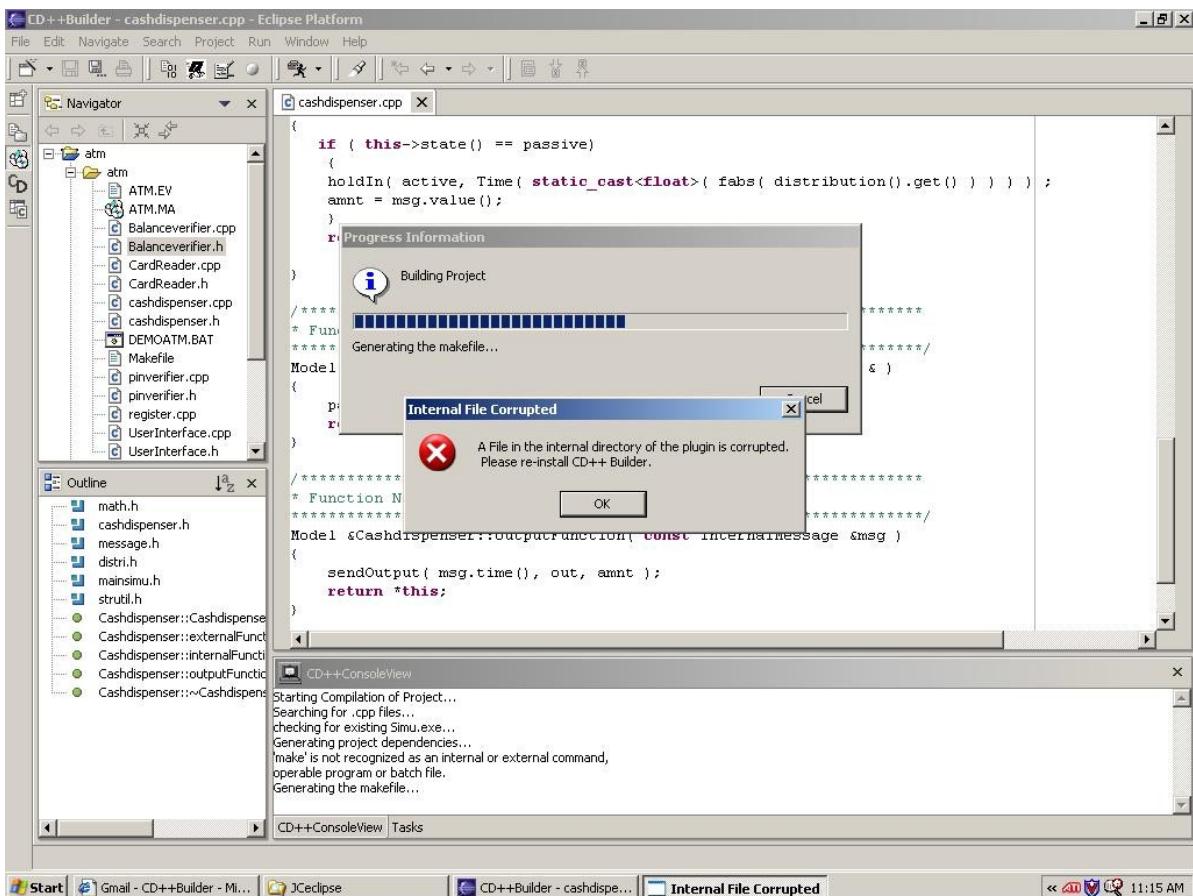


Figure 323: Error compiling ATM

It is noted that there's an error window stating "make" file is not recognized and that interal file is corrupted in the plugin. This means the version of CD++ plugin that you have installed is not installed properly or that it is buggy.

To fix this problem ensure that you have done the following:

1. Install cygwin correctly by following the steps in . Pay close attention to what packages of Cygwin to install; also, **importantly**, you must set the path correctly in Cygwin. This requires **ADMIN PRIVILEGES** on the computer in which you are installing Cygwin.
2. Reinstall the latest version of CD++. Go to the following webpage:

<http://www.sce.carleton.ca/faculty/wainer/wbgraf/distrib/>

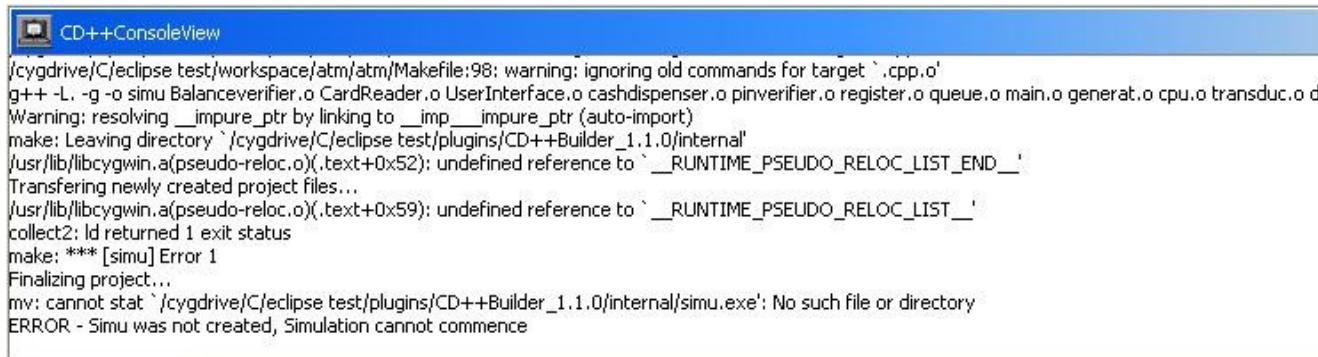
On this webpage you can find the version of CD++ plugin in working order.

Download the plugin by clicking on the link to download the **plugin only**.

Install the plugin by double clicking on it. Then click *next* and in the new window click browse to select the folder in which Eclipse is installed [path of main folder of eclipse should fill the dialog box]. Click *next* again to install.

10.1.5.2 ‘Simu.exe: No such file or directory; Simu was was not created!’

As mentioned above, both the Life2D and ATM model should execute properly to ensure that Eclipse and the plugin are working properly. One of the problems while running the ATM model might be of the nature shown in Figure 324 below.



```
CD++ConsoleView
/cygdrive/C/eclipse test/workspace/atm/atm/Makefile:98: warning: ignoring old commands for target ` .cpp.o'
g++ -L. -g -o simu BalanceVerifier.o CardReader.o UserInterface.o cashdispenser.o pinverifier.o register.o queue.o main.o generat.o cpu.o transduc.o d
Warning: resolving __impure_ptr by linking to __imp__impure_ptr (auto-import)
make: Leaving directory `/cygdrive/C/eclipse test/plugins/CD++Builder_1.1.0/internal'
/usr/lib/libcygwin.a(pseudo-reloc.o)(.text+0x52): undefined reference to `__RUNTIME_PSEUDO_RELOC_LIST_END_'
Transferring newly created project files...
/usr/lib/libcygwin.a(pseudo-reloc.o)(.text+0x59): undefined reference to `__RUNTIME_PSEUDO_RELOC_LIST_'
collect2: ld returned 1 exit status
make: *** [simu] Error 1
Finalizing project...
mv: cannot stat `/cygdrive/C/eclipse test/plugins/CD++Builder_1.1.0/internal/simu.exe': No such file or directory
ERROR - Simu was not created, Simulation cannot commence
```

Figure 324 Error when cygwin isn't installed properly

This basically says that “**Simu.exe not found, Simu was not created, cannot start simulations**”

This problem usually occurs if Cygwin is not installed properly and therefore it has to be reinstalled. Since Cygwin doesn't have an uninstaller, follow the steps below to uninstall cygwin from your computer:

Note: You need to have Admin Rights for the machine you're planning to do this on.

1. Go to **Start→Run** and then type “**regedit**” and hit “**Enter**” to open the Registry Editor. The following screen appears,

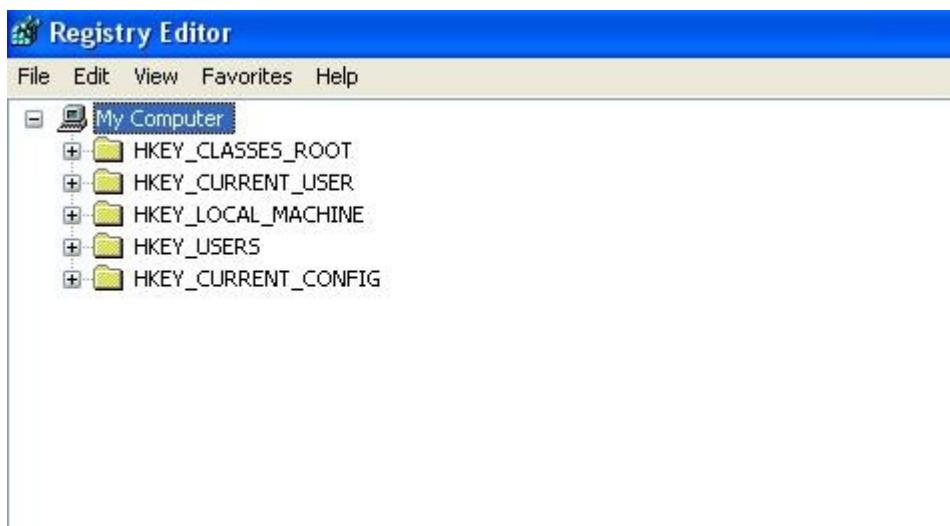


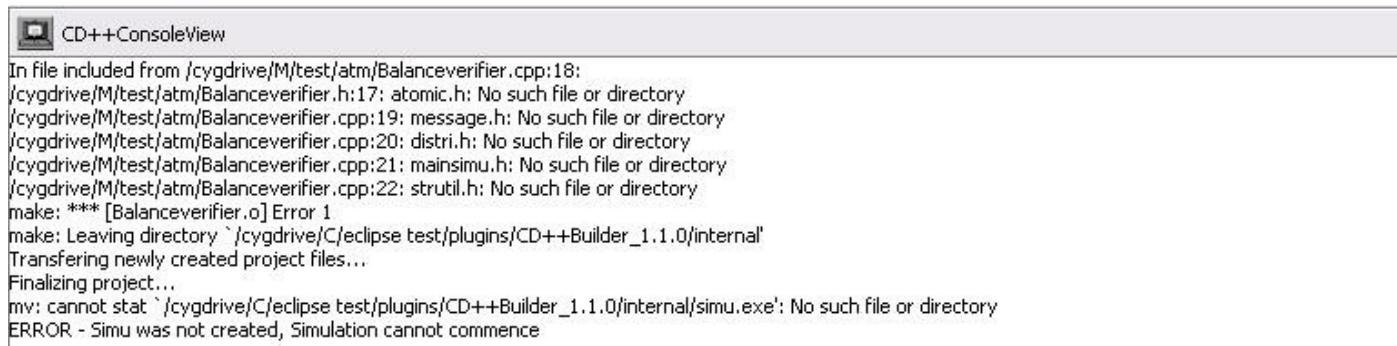
Figure 325 The Registry Editor

2. Now double-click the first folder below My-Computer that starts with [HKEY], and then double-click on a sub-folder named “**Software**”.

3. From the new sub-folders that open up, double-click the one that says “**Cygnus Solutions**” if it exists. If not then
4. Double-check to see if “**Cygwin**” appears as one of the sub-folders of “**Cygnus solutions**”, mentioned in step 3.
5. Click the minus (-) sign beside “**Cygnus solutions**” folder to close its subfolders.
6. Now right-click on the “**Cygnus Solutions**” folder and select “**delete**” to delete the entire folder along with its contents.
7. Repeat steps 2 through to 6 for all the folders (that start with HKEY) in the Registry Editor
8. After step 7 is done, all the registry keys associated with Cygwin would have been deleted.
9. Next, locate the folder where cygwin was previously installed, use the search utility in Windows Explorer to find occurrences of “**cygwin**”
10. Once the old cygwin folder is located, right-click it and select “**delete**” to delete it.
11. After that go to the desktop and delete any short-cuts to cygwin.
12. Lastly, go to the start menu by clicking “Start” and delete any start-menu shortcuts to Cygwin to complete the uninstallation.
13. Now, restart the machine by going to **Start→ShutDown** and selecting **Restart** from the drop-down menu.

Thereafter, reinstall cygwin as indicated in section .

Yet another reason for this problem is that the user has created a project in a non-default workspace (i.e. other than <.../eclipse/workspace/...>). If that is the case, then the error looks slightly different as shown in Figure 326.



```

CD++ConsoleView
In file included from /cygdrive/M/test/atm/Balanceverifier.cpp:18:
/cygdrive/M/test/atm/Balanceverifier.h:17: atomic.h: No such file or directory
/cygdrive/M/test/atm/Balanceverifier.cpp:19: message.h: No such file or directory
/cygdrive/M/test/atm/Balanceverifier.cpp:20: distri.h: No such file or directory
/cygdrive/M/test/atm/Balanceverifier.cpp:21: mainsimu.h: No such file or directory
/cygdrive/M/test/atm/Balanceverifier.cpp:22: strutil.h: No such file or directory
make: *** [Balanceverifier.o] Error 1
make: Leaving directory `/cygdrive/C/eclipse test/plugins/CD++Builder_1.1.0/internal'
Transferring newly created project files...
Finalizing project...
mv: cannot stat `/cygdrive/C/eclipse test/plugins/CD++Builder_1.1.0/internal/simu.exe': No such file or directory
ERROR - Simu was not created, Simulation cannot commence

```

Figure 326 Error if a project is created in a non-default workspace

To remedy this error, try creating the project in the default eclipse workspace instead, as shown in Figure 327 below.

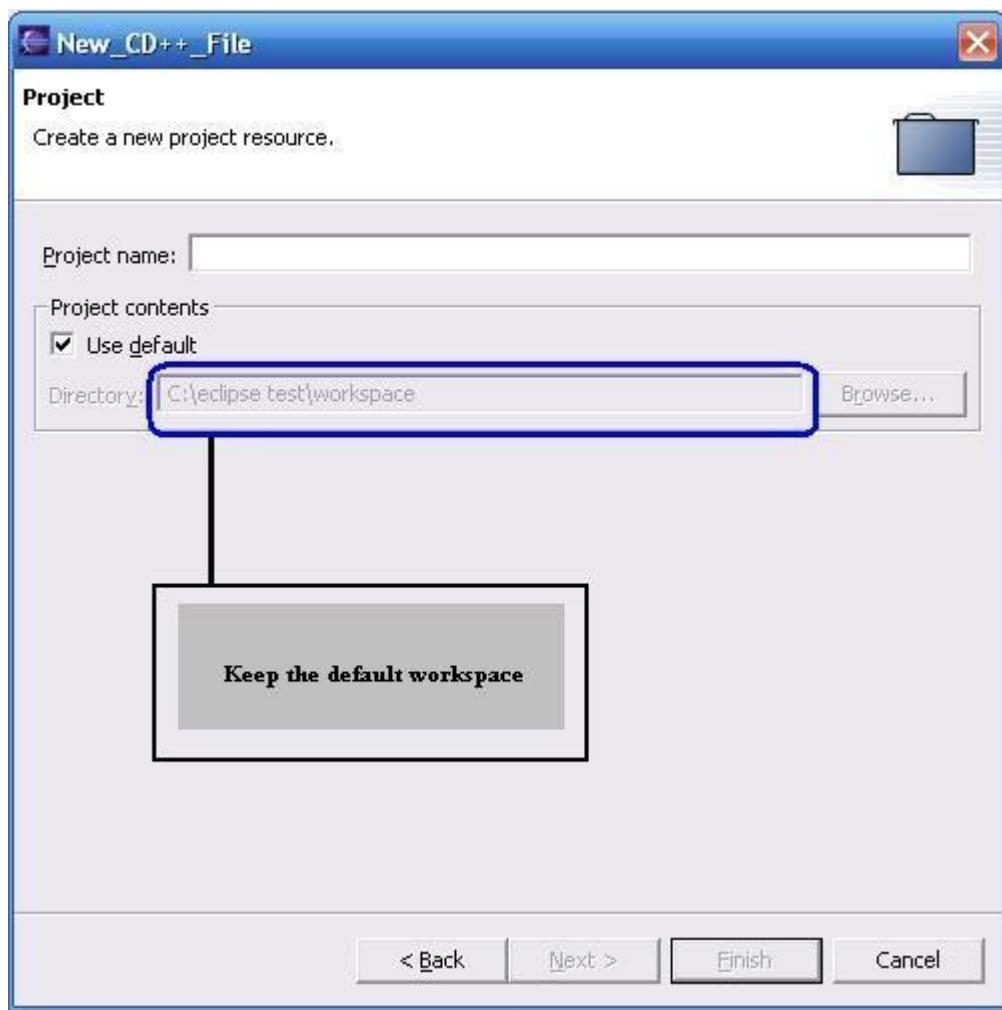


Figure 327 Use the default Eclipse workspace instead of any other

1. If you want to keep the project in a non-default workspace, then follow these steps:
 2. Close all Eclipse windows and re-open Eclipse again.
 3. Try recompiling the project again. If it still doesn't compile properly repeat steps 1 and 2. Two or three tries should do it.
- In registry editor, there is no subfolder called Cygwin or Cygnus Solutions under Software.
 - There is only two folders called Adobe and Microsoft.

10.1.5.3 ‘Deprecated JAVA methods’

A potential problem might be caused by using methods which are underlined as being

deprecated for e.g. consider the following,

Tasks (8 items)		
	✓ !	Description
		The method RunDrawlogInterfaceImpl.CommandRunThread.destroy() overrides a deprecated method from Thread

Figure 328 A warning generated by a deprecated method being overridden.

Tasks (8 items)		
	✓ !	Description
		The method show() from the type Dialog is deprecated

Figure 329 A warning generated by invoking a deprecated method

These warnings are generated because the methods are not implemented in some versions of the JRE. This can be fixed by making sure that the JRE(Java Runtime Environment) being used is the one mentioned in section i.e. < **J2SE v 1.4.2_05 JRE** > because the methods shown above are not deprecated in this older version of the JRE.

10.2 Command line installation

If you are planning to run the toolkit from the command line, the instructions are simple:

1. If you are using a Windows environment, refer to section 2.1, and download and install Cygwin
2. If you are using Linux, step 1 is not required.
3. Download CD++ toolkit from the toolkit website:

<http://www.sce.carleton.ca/faculty/wainer/wbgraf/distrib/CD++BuilderV1.2.zip>

CD++ toolkit files comes packed into a zip file (you will find different versions in the website). After having downloaded the toolkit, proceed to unzip the files to a directory. To execute a model, download it from <http://www.sce.carleton.ca/faculty/wainer/wbgraf> and unzip it in the same directory than the one where you included the toolkit. If it is a Cell-DEVS models, you just need to run the script (*.bat file) containing the execution commands for the model. If it is a DEVS model, you have to modify the Makefile (as explained in Section <> of the User Manual), and recompile the tool.

10.3 Installation for parallel simulation

Parallel CD++ was developed to run in UNIX and Windows NT environments that support the MPI library. It has been successfully tested in clusters of Linux machines running on Pentium processors. It supports both, parallel and standalone simulation.

The standalone version can also be compiled to run under Windows systems.

The CD++ distribution includes the following utilities:

- Drawlog: draws the evolution of a cellular model.
- Parlog: Counts the number of (*,t) messages received by each LP during each simulation cycle.
- Logbuffer: required by drawlog and parlog when parallel simulation is used. Sorts the log messages that are sent to standard output to ensure they are processed in the correct order.
- ToMap: creates the initial state cell map file from a .ma file.
- MakeRand: generates a random initial state cell map file.

The latest version of CD++ is distributed as a .tar.gz file and to install and compile CD++ the following utilities will be required:

- makedepend: current version released with X11R6 (part of X-windows software)
- GNU Make makefile utility (part of GNU software)
- g++: the GNU C++ compiler and accompanying libc, version 2.7.0 or later (part of GNU software)
- an implementation of MPI (e.g. MPICH) (for parallel simulation)
- GNU bison
- GNU flex

For parallel simulation, an implementation of MPI is required. If MPI is already installed in your system, find out if its includes and lib directories have been already added to the corresponding environment variables. Otherwise, take note of these directories because they will be required later on.

If MPI is not installed on your system, then it is recommended you install MPICH version 1.2.0, which can be downloaded from <http://www.mcs.anl.gov/home/lusk/mpich/index.html>. You can then install MPICH in a shared location (special permissions will be required) or in your home directory. Basic installation instructions will be provided.

The installation instructions here presented are based on personal experience installing in on Linux machines. If in doubt, please, check the mpich installation instructions found in **install.ps** in the /doc directory.

1. Uncompress the distribution files

```
gunzip -c mpich.tar.gz | tar xovf
```
2. Run

```
./configure
```

This script will try to set the optimum parameters for compilation on your system. If mpich will be installed in a shared location, then run (on your preferred location)

```
./configure --prefix= /usr/local/mpich-1.2.0.
```

3. Compile mpich by running

```
make >& make.log
```

This might take several minutes to an hour, depending on your system.

4. Edit the util/machines/machines.LINUX file and set the list of available machines in the cluster.
5. (Optional) Install mpich on a shared location
make install

If the default settings have not been changed, MPICH will use rsh to run the remote programs. For rsh to work properly, please check

1. Machine names are properly resolved, either using a DNS or the /etc/hosts file.
2. The inet services must be enabled in all the machines.
3. If you want to be able to run rsh without being prompted for a password, you will have to create a .rhosts file with the names of the machines in the cluster. The .rhost file must not have any group permissions enabled. Run chmod 600 .rhosts.
4. If the filesystem is not shared between all of the machines in the cluster, then a copy of CD++ and any model files will be required on each machine.

To install CD++, gunzip and untar the distribution file. On most Linux machines the command

```
gunzip -c pcd-3.x.x.tar.gz | tar xovf
```

will just do this.

The following directory structure will be created

```
CD++
+----- warped
      +----- TimeWarp
      +----- NoTime
      +----- Sequential
      +----- common
+----- models
      +----- net
      +----- airport
```

You must then edit Makefile.common and set the desired compilation options:

4. Set the source code location. If running parallel simulation, you will also need to indicate the location of the MPI include and lib files.

```
#CD++ Makefile.common
=====
#CD++ Directory Details
export MAINDIR=/USERDEFINEDPATH/CD++

=====
#MPI Directory Details
export MPIDIR=/USERDEFINEDPATH/mpich-1.2.0
export LDFLAGS +=-L$(MPIDIR)/lib/
export INCLUDES_CPP += -I$(MPIDIR)/include
=====
```

Figure 330. Makefile.common – Setting the source location

Specify whether parallel or stand alone simulation will be used. For stand alone simulation, the NoTime simulation kernel must be used. For parallel simulation, you can choose from the TimeWarp and NoTime kernel. If not sure, the NoTime kernel is recommended.

#If running parallel simulation, uncomment the following lines

```

export DEFINES_CPP += -DMPI
export LIBMPI = -lmpich
#=====
#=====
#WARPED CONFIGURATION
#=====
#Warped Directory Details
#For the TimeWarp kernel uncomment the following
#export DEFINES_CPP += -DKERNEL_TIMEWARP
#export TWDIR=$(MAINDIR)/warped/TimeWarp/src
#export PLIBS += -ITW -lm -lsl $(LIBMPI)
#export TWLIB = libTW.a

#For the NoTimeKernel, uncomment the following
export DEFINES_CPP += -DKERNEL_NOTIME
export TWDIR=$(MAINDIR)/warped/NoTime/src
export PLIBS += -lNoTime -lm -lsl $(LIBMPI)
export TWLIB = libNoTime.a
#=====
```

Figure 331. Makefile.common – Choosing the Warped kernel

- Decide which atomic models will be included by removing the necessary comments.

```

#####
#MODELS
#Let's define here which models we would like to include in our distribution
#Basic models
EXAMPLESOBJ=queue.o main.o generat.o cpu.o transduc.o distri.o com.o linpack.o register.o

#Uncomment these lines to include the airport models
#DEFINES_CPP += -DDEVS_AIRPORT
#INCLUDES_CPP += -I./models/airport
#LDFLAGS += -L./models/airport
#LIBS += -lairport

#Uncomment these lines to include the net models
#DEFINES_CPP += -DDEVS_NET
#INCLUDES_CPP += -I./models/net
#LDFLAGS += -L./models/net
#LIBS += -lnet
#####
```

Figure 332. Makefile.common – Model selection

After you have edited Makefile.common, you are ready to build CD++. To build CD++ and all the accompanying utilities, issue the following commands:

```
make depend
make
```

If you change any settings in Makefile.common you will need to rebuild CD++ again. To do this,

```
make clean
make
```

Appendix B - Local transition functions for Cell-DEVS models.

Local transition functions for cellular models are defined as groups in the .ma file. They are not tied to a particular model, so they can be used for more than one cellular model at the same time. A local transition is made of a set of rules of the form:

rule : result delay { condition }

A rule is composed of three elements: a *condition*, a *delay* and a *result*. To calculate the new value for a cell's state, the simulator takes each rule (in the order in that they were defined) and evaluates the condition clause. If the condition evaluates to true, then the result and delay clause are evaluated. The result will be the new cell state and will be sent as an output after the obtained delay. Whether the previous state values will be still sent as outputs or not will depend on the delay type of the cells. Inertial delay cells will preempt any scheduled outputs. On the other hand, transport delay cells will keep them.

Rules whose condition clause evaluates to false are skipped. If all the rules are evaluated without one having a true condition, then the simulation will be aborted. If there is more than one rule with a condition that evaluates to true, the first one will be the one that determines the new cell's state. If the delay clause of a cell evaluates to undefined, then the simulation will be automatically cancelled.

A grammar for writing the rules

The BNF for the grammar used for the rules is shown in Figure 333. Words written in bold lowercase represent terminals symbols, while those written in uppercase represent non terminals.

RULELIST	= RULE RULE RULELIST
RULE	= RESULT RESULT {BOOLEXP}
RESULT	= CONSTANT {REALEXP}
BOOLEXP	= BOOL (BOOLEXP) REALRELEXP not BOOLEXP BOOLEXP OP_BOOL BOOLEXP
OP_BOOL	= and or xor imp eqv
REALRELEXP	= REALEXP OP_REL REALEXP COND_REAL_FUNC(REALEXP)
REALEXP	= IDREF (REALEXP) REALEXP OPER REALEXP
IDREF	= CELLREF CONSTANT FUNCTION portValue(PORTNAME) send(PORTNAME, REALEXP) cellPos(REALEXP)
CONSTANT	= INT REAL CONSTFUNC ?
FUNCTION	= UNARY_FUNC(REALEXP) WITHOUT_PARAM_FUNC BINARY_FUNC(REALEXP, REALEXP) if(BOOLEXP, REALEXP, REALEXP) ifu(BOOLEXP, REALEXP, REALEXP, REALEXP)
CELLREF	= (INT, INT REST_TUPLE)
REST_TUPLE	= , INT REST_TUPLE)
BOOL	= t f ?
OP_REL	= != = > < >= <=
OPER	= + - * /
INT	= [SIGN] DIGIT {DIGIT}
REAL	= INT [SIGN] {DIGIT}.DIGIT {DIGIT}
SIGN	= + -
DIGIT	= 0 1 2 3 4 5 6 7 8 9
PORTNAME	= thisPort STRING
STRING	= LETTER {LETTER}

```

LETTER      = a | b | c | ... | z | A | B | C | ... | Z
CONSTFUNC   = pi | e | inf | grav | accel | light | planck | avogadro |
              faraday | rydberg | euler_gamma | bohr_radius | boltzmann |
              bohr_magneton | golden | catalan | amu | electron_charge |
              ideal_gas | stefan_boltzmann | proton_mass | electron_mass |
              neutron_mass | pem
WITHOUT_PARAM_FUNC = truecount | falsecount | undefcount | time | random |
                     randomSign
UNARY_FUNC   = abs | acos | acosh | asin | asinh | atan | atanh | cos | sec |
              sech | exp | cosh | fact | fractional | ln | log | round | cotan | cosec | cosech | sign | sin |
              sinh | statecount | sqrt | tan | tanh | trunc | truncUpper | poisson | exponential | randInt |
              chi | asec | acotan | asech | acosech | nextPrime | radToDeg | degToRad
              | nth_prime | acotanh | CtoF | CtoK | KtoC | KtoF | FtoC | FtoK

BINARY_FUNC  = comb | logn | max | min | power | remainder | root | beta | gamma |
              lcm | gcd | normal | f | uniform | binomial | rectToPolar_r | rectToPolar_angle |
              polarToRect_x | hip | polarToRect_y

COND_REAL_FUNC = even | odd | isInt | isPrime | isUndefined

```

Figure 333: Grammar used for the definition of a cell's local transition

Basically, a rule is made of three expressions: a result expression, a delay expression and a boolean expression. The result expression should evaluate to any real value. The delay expression should also evaluate to any real value that will be truncated to the smallest integer.

Precedence Order and Associativity of Operators

The precedence order indicates which operation will be solved first. For example if we have:

$$C + B * A$$

where * and + are the sum and multiplication operations for real numbers, and A , B and C are real constants, then since * has higher precedence than +, $B * A$ will be evaluated first. The sum will be evaluate in a second step. The result will be equivalent to solve $C + (B * A)$.

The associativity indicates which of two operations of same precedence will be evaluated first. Operators are either left associative or right associative. The logical operators *AND* and *OR* are left associative, so the *in* the expression

C and B or D

will be solved as (C and B) or D

Clauses that are not associative cannot be combined simultaneously without another operator of different precedence.

The table of precedence and associativities for the rule specification language follows:

<u>Order</u>	<u>Code</u>	<u>Associativity</u>
Lower	$\neg \neg p \equiv p$ $\neg(p \wedge q) \equiv \neg p \vee \neg q$ $\neg(p \vee q) \equiv \neg p \wedge \neg q$	\neg \wedge \vee

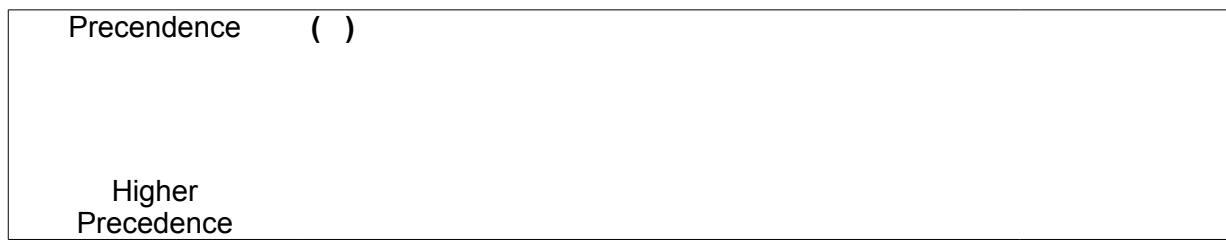


Figure 334: Precedence Order and Associativity used in CD++

Functions and Constants allowed by the language

Boolean Values

Boolean values in CD++ use trivalent logic.

The trivalent logic use the values **T** or **1** to represent to the value *TRUE*, **F** or **0** to represent the *FALSE*, and **?** to represent to the *UNDEFINED*.

Boolean Operators

Operator AND

The behavior of the operator **AND** is defined with the following table of truth:

AND	T	F	?
T	T	F	?
F	F	F	F

Figure 335: operator AND truthtable

Operator OR

The behavior of the operator OR is defined with the following table of truth:

OR	T	F	?
T	T	T	T
F	T	F	?

Figure 336: Operator OR truthtable

Operator NOT

The behavior of the operator NOT is defined with the following table of truth:

<i>NOT</i>	
T	F
F	T

Figure 337: Behavior of the boolean operator NOT

Operator XOR

The behavior of the operator XOR is defined with the following table of truth:

Figure 338: Operator XOR truthtable

<i>XOR</i>	T	F	?
T	F	T	?
F	T	F	?

Operator IMP

IMP represents the logic implication, and its behavior is defined with the following table of truth:

<i>IMP</i>	T	F	?
T	T	F	?
F	T	T	T

Figure 339: Operator IMP truthtable

Operator EQV

EQV represents the equivalence between trivalent logic values, and its behavior is defined with the following table of truth:

Figure 340: Operator EQV truthtable

<i>EQV</i>	T	F	?
T	T	F	F
F	F	T	F

Functions and Operations on Real Numbers

Relational Operators

The relational operators work on real numbers¹ and return a boolean value pertaining to the previously defined trivalent logic. The language used by CD++ allows the use of the operators ==, !=, >, <, >=, <= whose behavior is described next.

As opposed to the traditional definition of these operators, the introduction of an undefined value

¹ From here, when referring to the term “Real Number” a value in the set $R \cup \{ ? \}$ will be meant.

makes the definition of a total order impossible because the value ? is not comparable with any existing real number.

Operator =

The operator = is used to test for equality of two real numbers.

=	?	Real Number
?	T	?
Real Number	?	= of real number

Figure 341: Behavior of the Relational Operator =

Operator !=

The operator != is used to test if two real numbers are not equal. Its behavior is defined as follows:

!=	?	Real Number
?	F	?
Real Number	?	\neq of real number

Figure 342: Behavior of the Relational Operator !=

Operator >

The operator > is used to test if a real number is greater than another real number. Its behavior is defined as follows:

>	?	Real Number
?	F	?
Real Number	?	> of real number

Figure 343: Behavior of the Relational Operator >

Operator <

The operator `<` is used to test if a real number is less than another real number. Its behavior is

<code><</code>	?	Real Number
?	F	?
Real Number	?	<code><</code> of real number

defined as follows:

Figure 344: Behavior of the Relational Operator `<`

Operator `<=`

The operator `<=` is used to test if a real number is less or equal to another real number. Its behavior is defined as follows:

Figure 345: Behavior of the Relational Operator `<=`

<code><=</code>	?	Real Number
?	T	?
Real Number	?	<code>\leq</code> of real number

Operator `>=`

The operator `>=` is used to test if a real number is greater or equal to another real number. Its behavior is defined as follows:

<code>>=</code>	?	Real Number
?	T	?
Real Number	?	<code>\geq</code> of real number

Figure 346: Behavior of the Relational Operator `>=`

Arithmetic Operators

The traditional arithmetic operators are available. If any of the operands is undefined, then the result of the operation will be undefined. This is also valid for functions. If any of a function arguments is undefined, the result of evaluating the function will also be undefined.

The available operators are:

Division by zero will result to the undefined value.

<code>op1 + op2</code>	returns the sum of the operators.
<code>op1 - op2</code>	returns the difference between the operators.
<code>op1 / op2</code>	returns the value of the op1 divided by op2.
<code>op1 * op2</code>	returns the product of the operators

Figure 347: Arithmetic Operators

Functions on Real Numbers

Functions to Verify Properties of Real Numbers

The functions in this section allow to check for special properties of real numbers, such as parity, primality, etc.

Function Even

<u>Signature:</u>	<code>even : Real □ Bool</code>
<u>Description:</u>	Returns <i>True</i> if the value is integer and even. If the value is undefined returns <i>Undefined</i> . In any other case it returns <i>False</i> .
<u>Examples:</u>	$\text{even}(?) = \text{F}$ $\text{even}(3.14) = \text{F}$ $\text{even}(3) = \text{F}$ $\text{even}(2) = \text{T}$

Function Odd

<u>Signature:</u>	<code>odd : Real □ Bool</code>
<u>Description:</u>	Returns <i>True</i> if the value is integer and odd. If the value is undefined returns <i>Undefined</i> . In any other case it returns <i>False</i> .
<u>Examples:</u>	$\text{odd}(?) = \text{F}$ $\text{odd}(3.14) = \text{F}$ $\text{odd}(3) = \text{T}$ $\text{odd}(2) = \text{F}$

Function isInt

<u>Signature:</u>	<code>isInt : Real □ Bool</code>
<u>Description:</u>	Returns <i>True</i> if the value is integer and not undefined. Any other case returns <i>False</i> .
<u>Examples:</u>	$\text{isInt}(?) = \text{F}$ $\text{isInt}(3.14) = \text{F}$ $\text{isInt}(3) = \text{T}$

Function isPrime

<u>Signature:</u>	<code>isPrime : Real □ Bool</code>
<u>Description:</u>	Returns <i>True</i> if the value is a prime number. Any other case returns <i>False</i> .
<u>Examples:</u>	$\text{isPrime}(?) = \text{F}$ $\text{isPrime}(3.14) = \text{F}$ $\text{isPrime}(6) = \text{F}$ $\text{isPrime}(5) = \text{T}$

Function isUndefined

<u>Signature:</u>	<code>isUndefined : Real □ Bool</code>
-------------------	----------------------------------------

<u>Description:</u>	Returns <i>True</i> if the value is undefined, else returns <i>False</i> .
<u>Examples:</u>	<pre>isUndefined(?) = T isUndefined(4) = F</pre>

Mathematical Functions

This section describes commonly used mathematical functions.

Trigonometric Functions

Function tan

<u>Signature:</u>	tan : Real $a \square$ Real
<u>Description:</u>	Returns the tangent of a measured in radians. For the values near to $\pi/2$ radians, returns the constant <i>INF</i> . If a is undefined then return undefined.
<u>Examples:</u>	<pre>tan(PI / 2) = INF tan(?) = ? tan(PI) = 0</pre>

Function sin

<u>Signature:</u>	sin : Real $a \square$ Real
<u>Description:</u>	Returns the sine of a measured in radians. If a has the value <i>?</i> then returns <i>?</i> .

Function cos

<u>Signature:</u>	cos : Real $a \square$ Real
<u>Description:</u>	Returns the cosine of a measured in radians. If a has the value <i>?</i> then returns <i>?</i> .

Function sec

<u>Signature:</u>	sec : Real $a \square$ Real
<u>Description:</u>	Returns the secant of a measured in radians. If a has the value <i>?</i> then returns <i>?</i> . If the angle is of the form $\pi/2 + x\pi$, with x an integer number, then returns the constant <i>INF</i> .

Function cotan

<u>Signature:</u>	cotan : Real $a \square$ Real
<u>Description:</u>	Calculates the cotangent of a . If a has the value <i>?</i> Then returns <i>?</i> . If a is zero or multiple of π , then returns <i>INF</i> .

Function cosec

<u>Signature:</u>	cosec : Real $a \square$ Real
<u>Description:</u>	Calculates the cosecant of a . If a has the value <i>?</i> , then returns <i>?</i> . If a is zero or multiple of π , then returns <i>INF</i> .

Function atan

<u>Signature:</u>	atan : Real $a \square$ Real
<u>Description:</u>	Returns the arc tangent of a measured in radians, which is defined as the value b such $\tan(b) = a$. If a has the value <i>?</i> Then returns <i>?</i> .

Function asin

Signature:

Description:

asin : Real $a \square$ Real

Returns the arc sine of a measured in radians, which is defined as the value b such $\sin(b) = a$.

If a has the value ? or if $a \square [-1, 1]$, then returns ?.

Function acos

Signature:

Description:

acos : Real $a \square$ Real

Returns the arc cosine of a measured in radians, which is defined as the value b such $\cos(b) = a$.

If a has the value ? or if $a \square [-1, 1]$, then returns ?.

Function asec

Signature:

Description:

asec : Real $a \square$ Real

Returns the arc secant of a measured in radians, which is defined as the value b such $\sec(b) = a$.

If a is undefined (?) or if $|a| < 1$, then returns ?.

Function acotan

Signature:

Description:

acotan : Real $a \square$ Real

Returns the arc cotangent of a measured in radians, which is defined as the value b such $\cotan(b) = a$.

If a is undefined (?), then returns ?.

Function sinh

Signature:

Description:

sinh : Real $a \square$ Real

Returns the hyperbolic sine of a measured in radians.

If a has the value ?, then returns ?.

Function cosh

Signature:

Description:

cosh : Real $a \square$ Real

Returns the hyperbolic cosine of a measured in radians, which is defined as $\cosh(x) = (e^x + e^{-x}) / 2$.

If a has the value ?, then returns ?.

Function tanh

Signature:

Description:

tanh : Real $a \square$ Real

Returns the hyperbolic tangent of a measured in radians, which is defined as $\sinh(a) / \cosh(a)$.

If a has the value ?, then returns ?.

Function sech

Signature:

Description:

sech : Real $a \square$ Real

Returns the hyperbolic secant of a measured in radians, which is defined as $1 / \cosh(a)$

If a has the value ?, then returns ?.

Function cosech

Signature:

Description:

cosech : Real $a \square$ Real

Returns the hyperbolic cosecant of a measured in radians.

If a has the value ?, then returns ?.

Function atanh

Signature:

Description:

atanh : Real $a \square$ Real

Returns the hyperbolic arc tangent of a measured in radians, which is

defined as the value b such $\tanh(b) = a$.

If a has the value $?$, or if its absolute value is greater than 1 (i.e., $a \in [-1, 1]$), then returns $?$.

Function `asinh`

Signature:

Description:

asinh : *Real a* \rightarrow *Real*

Returns the hyperbolic arc sine of a measured in radians, which is defined as the value b such $\sinh(b) = a$.

If a has the value $?$, then returns $?$.

Function `acosh`

Signature:

Description:

acosh : *Real a* \rightarrow *Real*

Returns the hyperbolic arc cosine of a measured in radians, which is defined as the value b such $\cosh(b) = a$.

If a has the value $?$ or is less than 1, then returns $?$.

Function `asech`

Signature:

Description:

asech : *Real a* \rightarrow *Real*

Returns the hyperbolic arc secant of a measured in radians, which is defined as the value b such $\text{sech}(b) = a$.

If a is undefined, then return $?$. If it is zero, then returns the constant *INF*.

Function `acosech`

Signature:

Description:

acosech : *Real a* \rightarrow *Real*

Returns the hyperbolic arc cosec of a measured in radians, which is defined as the value b such $\text{cosech}(b) = a$.

If a is undefined, then returns $?$. If it is zero, then returns the constant *INF*.

Function `acotanh`

Signature:

Description:

acotanh : *Real a* \rightarrow *Real*

Returns the hyperbolic arc cotangent of a measured in radians, which is defined as the value b such $\text{cotanh}(b) = a$.

If a is undefined, then returns $?$. If it is 1 then returns the constant *INF*.

Function `hip`

Signature:

Description:

hip : *Real c1 x Real c2* \rightarrow *Real*

Calculates the hypotenuse of the triangle composed by the side $c1$ and $c2$. If $c1$ or $c2$ are undefined or negatives, then returns $?$.

Functions to calculate Roots, Powers and Logarithms.

Function `sqr`

Signature:

Description:

sqr : *Real a* \rightarrow *Real*

Returns the square root of a .

If a is undefined or negative, then returns $?$.

Examples :

$\text{sqr}(4) = 2$
 $\text{sqr}(2) = 1.41421$
 $\text{sqr}(0) = 0$
 $\text{sqr}(-2) = ?$
 $\text{sqr}(?) = ?$

Note:

$\text{sqr}(x)$ is equivalent to $\text{root}(x, 2)$

Function `exp`

Signature: **exp** : *Real* $x \square$ *Real*
Description: Returns the value of e^x .
If x is undefined, then return ?.
Examples: $\exp(?) = ?$
 $\exp(-2) = 0.135335$
 $\exp(1) = 2.71828$
 $\exp(0) = 1$

Function *In*

Signature: **In** : *Real* $a \square$ *Real*
Description: Returns the natural logarithm of a .
If a is undefined or is less or equal than zero, then returns ?.
Examples: $\ln(-2) = ?$
 $\ln(0) = ?$
 $\ln(1) = 0$
 $\ln(?) = ?$
Note: $\ln(x)$ is equivalent to **logn**(x , e) . x

Function *log*

Signature: **log** : *Real* $a \square$ *Real*
Description: Returns the logarithm in base 10 of a .
If a is undefined or less or equal to zero, then returns ?.
Examples: $\log(3) = 0.477121$
 $\log(-2) = ?$
 $\log(?) = ?$
 $\log(0) = ?$
Note: $\log(x)$ is equivalent to **logn**(x , 10) . x

Function *logn*

Signature: **logn** : *Real* $a \times$ *Real* $n \square$ *Real*
Description: Returns the logarithm in base n of the value a .
If a or n are undefined, negatives or zero, then returns ?.
Notes: $\logn(x, e)$ is equivalent to **In**(x) . x
 $\logn(x, 10)$ is equivalent to **log**(x) . x

Function *power*

Signature: **power** : *Real* $a \times$ *Real* $b \square$ *Real*
Description: Returns a^b .
If a or b are undefined or b is not an integer, then returns ?.

Function *root*

Signature: **root** : *Real* $a \times$ *Real* $n \square$ *Real*
Description: Returns the n -root of a .
If a or n are undefined, then returns ?. Also, returns this value if a is negative or n is zero.
Examples: $\text{root}(27, 3) = 3$
 $\text{root}(8, 2) = 3$
 $\text{root}(4, 2) = 2$
 $\text{root}(2, ?) = ?$
 $\text{root}(3, 0.5) = 9$
 $\text{root}(-2, 2) = ?$
 $\text{root}(0, 4) = 0$
 $\text{root}(1, 3) = 1$
 $\text{root}(4, 3) = 1.5874$
Note: $\text{root}(x, 2)$ is equivalent to **sqrt**(x) . x

Functions to calculate GCD, LCM and the Rest of the Numeric Division

Function LCM

Signature: **lcm** : Real a x Real b □ Real
Description: Returns the Less Common Multiplier between a and b.
If a or b are undefined or non-integers, then returns ?.
The value returned is always integer.

Function GCD

Signature: **gcd** : Real a x Real b □ Real
Description: Calculates the Greater Common Divisor between a and b.
If a or b are undefined or non-integers, then returns ?.
The value returned is always integer.

Function remainder

Signature: **remainder** : Real a x Real b □ Real
Description: Calculates the remainder of the division between a and b. The returned value is: $a - n * b$, where n is the quotient a/b rounded as an integer.
If a or b are undefined, then returns ?.
Examples:
remainder(12, 3) = 0
remainder(14, 3) = 2
remainder(4, 2) = 0
remainder(0, y) = 0 . y □ ?
remainder(x, 0) = x . x
remainder(1.25, 0.3) = 0.05
remainder(1.25, 0.25) = 0
remainder(?, 3) = ?
remainder(5, ?) = ?

Functions to Convert Real Values to Integers Values

This section presents functions available to convert real values to integers using the rounding and truncation techniques as detailed.

Function round

Signature: **round** : Real a □ Real
Description: Rounds the value a to the nearest integer.
If a is undefined ?, then returns ?.
Examples:
round(4) = 4
round(?) = ?
round(4.1) = 4
round(4.7) = 5
round(-3.6) = -4

Function trunc

Signature: **trunc** : Real x □ Real
Description: Returns the greater integer number less or equal than x.
If x is undefined, then returns ?.
Examples:
trunc(4) = 4
trunc(?) = ?
trunc(4.1) = 4
trunc(4.7) = 4

Function truncUpper

Signature: **truncUpper** : *Real* $x \square$ *Real*
Description: Returns the smallest integer number greater or equal than x .
If x is undefined, then returns **?**.
Examples: $\text{truncUpper}(4) = 4$
 $\text{truncUpper}(?) = ?$
 $\text{truncUpper}(4.1) = 5$
 $\text{truncUpper}(4.7) = 5$

Function fractional

Signature: **fractional** : *Real* $a \square$ *Real*
Description: Returns the fractional part of a , including the sign.
If a is undefined then returns **?**.
Examples: $\text{fractional}(4.15) = 0.15$
 $\text{fractional}(?) = ?$
 $\text{fractional}(-3.6) = -0.6$

Functions to manipulate the Sign of numerical values

Function abs

Signature: **abs** : *Real* $a \square$ *Real*
Description: Returns the absolute value of a .
If a is undefined then returns **?**.
Examples: $\text{abs}(4.15) = 4.15$
 $\text{abs}(?) = ?$
 $\text{abs}(-3.6) = 3.6$
 $\text{abs}(0) = 0$

Function sign

Signature: **sign** : *Real* $a \square$ *Real*
Description: Returns the sign of a in the following form:
If $a > 0$ then returns 1.
If $a < 0$ then returns -1.
If $a = 0$ then returns 0.
If $a = ?$ then returns **?**.

Function randomSign

See the section of Probability Functions.

Functions to manipulate Prime numbers

This functions are used to test for primality. Although they are available, they are quite complex and can require a lot of time to solve.

Function isPrime

See the section of Functions to Verify Properties of Real Numbers.

Function nextPrime

Signature: **nextPrime** : *Real* $r \square$ *Real*
Description: Returns the next prime number greater than r .
If r is undefined then returns **?**.
If an overflow occur when calculating the next prime number, the constant *INF* is returned.

Function nth_Prime

Signature: **nth_Prime** : Real $n \square$ Real
Description: Returns the n^{th} prime number, considering as the first prime number the value 2.
If n is undefined or non-integer then returns ?.
If an overflow occur when calculating the next prime number, the constant *INF* is returned.

Functions to calculate Minimum and Maximums

Function min

Signature: **min** : Real $a \times$ Real $b \square$ Real
Description: Return the minimum between a and b .
If a or b are undefined then returns ?.

Function max

Signature: **max** : Real $a \times$ Real $b \square$ Real
Description: Returns the maximum between a and b .
If a or b are undefined then returns ?.

Conditional Functions

The functions described in this section return a real value that depends on the evaluation of a specified logical condition.

Function if

Signature: **if** : Bool $c \times$ Real $t \times$ Real $f \square$ Real
Description: If the condition c is evaluated to *TRUE*, then returns the evaluation of t , else returns the evaluation of f .
The values of t and f can even come from the evaluation of any expression that returns a real value, including another *if* sentence.
Examples: If you wish to return the value 1.5 when the natural logarithm of the cell (0, 0) is zero or negative, or 2 in another case. In this case, it will be written:

$$\text{if}(\ln((0, 0)) = 0 \text{ or } (0, 0) < 0, 1.5, 2)$$
If you wants to return the value of the cells $(1, 1) + (2, 2)$ when the cell (0, 0) isn't zero; or the square root of (3, 3) in another case, it will be written:

$$\text{if}((0, 0) != 0, (1, 1) + (2, 2), \sqrt(3, 3))$$
It can also be used for the treatment of a numeric overflow. For example, if the factorial of the cell (0, 1) produces an overflows, then return -1, else return the obtained result. In this case, it will be written:

$$\text{if}(\text{fact}((0, 1)) = \text{INF}, -1, \text{fact}((0, 1)))$$

Function ifu

Signature: **ifu** : Bool $c \times$ Real $t \times$ Real $f \times$ Real $u \square$ Real
Description: If the condition c is evaluated to *TRUE*, then returns the evaluation of t . If it evaluates to *FALSE*, returns the evaluation of f . Else (i.e. is undefined), returns the evaluation of u .
Examples: If you wish to return the value of the cell (0, 0) if its value is distinct than zero and undefined, 1 if the value of the cell is 0, and \square if the cell has the undefined value. In this case, it will be invoked:

$$\text{ifu}((0, 0) != 0, (0, 0), 1, \text{PI})$$

Probabilistic Functions

Function randomSign

Signature: **randomSign** : \square Real
Description: Randomly returns a numerical value that represents a sign (+1 or -1), with equal probability for both values.

Function random

Signature: **random** : \square *Real*
Description: Returns a random real value pertaining to the interval (0, 1), with uniform distribution.

Note: *random* is equivalent to *uniform(0, 1)*.

Function chi

Signature: **chi** : *Real df* \square *Real*
Description: Returns a random real number with Chi-Squared distribution with *df* degree of freedom.

If *df* is undefined, negative or zero, then returns ?.

Function beta

Signature: **beta** : *Real a* \times *Real b* \square *Real*
Description: Returns a random real number with Beta distribution, with parameters *a* and *b*.

If *a* or *b* are undefined or less than 10^{-37} , then returns ?.

Function exponential

Signature: **exponential** : *Real av* \square *Real*
Description: Returns a random real number with Exponential distribution, with average *av*.
If *av* is undefined or negative, then returns ?.

Function f

Signature: **f** : *Real dfn* \times *Real dfd* \square *Real*
Description: Returns a random real number with F distribution, with *dfn* degree of freedom for the numerator, and *dfd* for the denominator.
If *dfn* or *dfd* are undefined, negatives or zero, then return ?.

Function gamma

Signature: **gamma** : *Real a* \times *Real b* \square *Real*
Description: Returns a random real number with Gamma distribution with parameters (*a*, *b*).
If *a* or *b* are undefined, negatives or zero, then returns ?.

Function normal

Signature: **normal** : *Real \bar{x}* \times *Real σ* \square \square *Real*
Description: Returns a random real number with Normal distribution (\bar{x} , σ), where \bar{x} is the average, and σ is the standard error.
If \bar{x} or σ are undefined, or σ is negative, returns ?.

Function uniform

Signature: **uniform** : *Real a* \times *Real b* \square *Real*
Description: Returns a random real number with uniform distribution, pertaining to the interval (*a*, *b*).
If *a* or *b* are undefined, or *a* > *b*, then returns ?.
uniform(0, 1) is equivalent to the function *random*.

Function binomial

Signature: **binomial** : *Real n* \times *Real p* \square *Real*
Description: Returns a random number with Binomial distribution, where *n* is the number of attempts, and *p* is the success probability of an event.
If *n* or *p* are undefined, *n* is not integer or negative, or *p* not pertain to the

interval [0, 1], then return ?.
The returned number is always an integer.

Function poisson

Signature:

Description:

poisson : Real $n \square$ Real

Return a random number with Poisson distribution, with average n .

If n is undefined or negative, then returns ?.

The returned number is always an integer.

Function randInt

Signature:

Description:

randInt : Real $n \square$ Real

Returns an integer random number contained in the interval [0, n], with uniform distribution.

If n is undefined, then returns ?.

Note: $\text{randInt}(n)$ is equivalent to $\text{round}(\text{uniform}(0, n))$

Functions to calculate Factorials and Combinatorial

Function fact

Signature:

Description:

fact : Real $a \square$ Real

Returns the factorial of a .

If a is undefined, negative or non-integer, then return ?.

If an overflow occur when calculating the next prime number, the constant *INF* is returned.

Examples:

fact(3) = 6

fact(0) = 1

fact(5) = 120

fact(13) = 1.93205e+09

fact(43) = *INF*

Function comb

Signature:

comb : Real $a \times$ Real $b \square$ Real

Description:

Returns the combinatory $\binom{a}{b}$

If a or b are undefined, negatives or zero, or non-integers, then returns ?.

This value is also returned if $a < b$.

If an overflow occur when calculating the next prime number, the constant *INF* is returned.

Functions for the Cells and his Neighborhood

This section details the functions that allow to count the quantity of cells belonging to the neighborhood whose state has certain value, as also the function *cellPos* that allows to project an element of the tuple that references to the cell.

Function stateCount

Signature:

stateCount : Real $a \square$ Real

Description:

Returns the quantity of neighbors of the cell whose state is equal to a .

Function trueCount

Signature:

trueCount : \square Real

Description:

Returns the quantity of neighbors of the cell whose state is 1.

This function is equivalent to *stateCount(1)* and it is not removed from the language to offer compatibility with CD++.

Function falseCount

Signature:

falseCount : \square Real

Description: Returns the quantity of neighbors of the cell whose state is 0.
This function is equivalent to *stateCount(1)* and it is not removed from the language to offer compatibility with *CD++*.

Function *undefCount*

Signature: **undefCount** : □ *Real*
Description: Returns the quantity of neighbors of the cell whose state is undefined (?).
This function is equivalent to *stateCount(1)* and it is not removed from the language to offer compatibility with *CD++*.

Function *cellPos*

Signature: **cellPos** : *Real* *i* □ *Real*
Description: Returns the *i*th position inside the tuple that references to the cell. That is to say, given the cell (x_0, x_1, \dots, x_n) , then *cellPos(i)* = x_i .

If the value of *i* is not integer, then it will be automatically truncated.

If $i \notin [0, n+1]$, where *n* is the dimension of the model, it will produce an error that will abort the simulation.

The value returned always will be an integer.

Examples:

Given the cell (4, 3, 10, 2):
cellPos(0) = 4
cellPos(3.99) = *cellPos(3)* = 2
cellPos(1.5) = *cellPos(1)* = 3
cellPos(-1) y *cellPos(4)* will generate an error.

Functions to Get the Simulation Time

Function *Time*

Signature: **time** : □ *Real*
Description: Returns the time of the simulation at the moment in that the rule this being evaluated, expressed in milliseconds.

Functions to Convert Values between different units

Functions to Convert Degrees to Radians

Function *radToDeg*

Signature: **radToDeg** : *Real* *r* □ *Real*
Description: Converts the value *r* from radians to degrees.
If *r* is undefined then returns ?.

Function *degToRad*

Signature: **degToRad** : *Real* *r* □ *Real*
Description: Converts the value *r* from degrees to radians.
If *r* is undefined then returns ?.

Functions to Convert Rectangular to Polar Coordinates

Function *rectToPolar_r*

Signature: **rectToPolar_r** : *Real* *x* x *Real* *y* □ *Real*
Description: Converts the Cartesian coordinate (*x*, *y*) to the polar form (*r*, θ), and returns *r*.
If *x* or *y* are undefined then return ?.

Function *rectToPolar_angle*

Signature: **rectToPolar_angle** : Real x x Real y \square Real
Description: Converts the Cartesian coordinate (x, y) to the polar form (r, θ), and returns \square .
If x or y are undefined then return ?.

Function polarToRect_x

Signature: **polarToRect_x** : Real r x Real \square Real
Description: Converts the polar coordinate (r, θ) to the Cartesian form (x, y), and returns x.
If r or θ are undefined, or r is negative, then returns ?.

Function polarToRect_y

Signature: **polarToRect_y** : Real r x Real \square Real
Description: Converts the polar coordinate (r, θ) to the Cartesian form (x, y), and returns y.
If r or θ are undefined, or r is negative, then returns ?.

Functions to Convert Temperatures between different units

Function CtoF

Signature: **CtoF** : Real \square Real
Description: Converts a value expressed in Centigrade degrees to Fahrenheit degrees.
If the value is undefined then returns ?.

Function CtoK

Signature: **CtoK** : Real \square Real
Description: Converts a value expressed in Centigrade degrees to Kelvin degrees.
If the value is undefined then returns ?.

Function KtoC

Signature: **KtoC** : Real \square Real
Description: Converts a value expressed in Kelvin degrees to Centigrade degrees.
If the value is undefined then returns ?.

Function KtoF

Signature: **KtoF** : Real \square Real
Description: Converts a value expressed in Kelvin degrees to Fahrenheit degrees.
If the value is undefined then returns ?.

Function FtoC

Signature: **FtoC** : Real \square Real
Description: Converts a value expressed in Fahrenheit degrees to Centigrade degrees.
If the value is undefined then returns ?.

Function FtoK

Signature: **FtoK** : Real \square Real
Description: Converts a value expressed in Fahrenheit degrees to Kelvin degrees.
If the value is undefined then returns ?.

Functions to manipulate the Values on the Input and Output Ports

Function portValue

Signature:

Description:

portValue : String $p \sqsubset \text{Real}$

Returns the last value arrived through the input port p of the cell of the cell being evaluated. This function will only be available for *PortInTransition* rules (see section 12.3). Other uses will generate an error.

If no message has arrived through port p before *portValue* is evaluated, an undefined value (?) will be returned. Otherwise, the last value received through the port will be returned.

When the string “**thisPort**” is used as the port name, the value received through the port associated with the current *PortInTransition* will be returned. For example:

The following model has two different *PortInTransitions*

```

PortInTransition: portA@cell(0,0)      functionA
PortInTransition: portB@cell(1,1)      functionB

[functionA]
rule: 10    100 { portValue(portA) > 10 }
rule: 0     100 { t }

[functionB]
rule: 10    100 { portValue(portB) > 10 }
rule: 0     100 { t }

```

Figure 348: Example of use of the function *portValue*

If we wanted to avoid repeating the same transition twice, we could either give the two ports the same name or use *thisPort* as shown next:

```

PortInTransition: portA@cell(0,0)      functionA
PortInTransition: portB@cell(1,1)      functionA

[functionA]
rule: 10    100 { portValue(thisPort) > 10 }
rule: 0     100 { t }

```

Figure 349: Example of use of the function *portValue* with *thisPort*

Section 12.3 shows an example where the *portInTransition* clause is used.

Function send

Signature:

Description:

send : String $p \times \text{Real} \sqsubset 0$

Sends the value x through the output port p .

If the output port p has not been defined, an error will be raised and the simulation will be aborted. This function is usually used to send values to other DEVS models.

send always returns 0. This makes it possible to include the function **send** in the *result* section of a rule without modifying the actual results.

```
{ (0,0) + send( port1, 15 * log(10) ) } 100 { (0,0) > 10 }
```

Note: **Send** is a function of the language that can be used in any expression, as for example, in the definition of a *condition*. However, this is not recommended because for every condition that is evaluated that includes the function send, a value will be sent. Instead, send should be used in the expression for the *delay* or the *value* of the cell.

Predefined Constants

The following constants frequently used in the domains of the physics and the chemistry are available.

Constant Pi

Returns 3.14159265358979323846, which represent the value of π , the relation between the circumference and the radius of the circle.

Constant e

Returns 2.7182818284590452353, which represent the value of the base of the natural logarithms.

Constant INF

This constant represents to the infinite value, although in fact it returns the maximum value valid for a *Double* number (in processors Intel 80x86, this number is 1.79769×10^{308}).

Note that if, for example, we make $x + INF - INF$, where x is any real value, we will get 0 as a result, because the operator + is associative to left, for that will be solved:

$$(x + INF) - INF = INF - INF = 0.$$

Note: When being generated a numeric overflows taken place by any operation, it is returned INF or -INF. For example: power(12333333, 78134577) = INF.

Constant electron_mass

Returns the mass of an electron, which is $9.1093898 \times 10^{-28}$ grams.

Constant proton_mass

Returns the mass of a proton, which is $1.6726231 \times 10^{-24}$ grams.

Constant neutron_mass

Returns the mass of a neutron, which is $1.6749286 \times 10^{-24}$ grams.

Constant Catalan

Returns the Catalan's constant, which is defined as $\sum_{k=0}^{\infty} (-1)^k \cdot (2^k + 1)^{-2}$, that is approximately 0.9159655941772.

Constant Rydberg

Returns the Rydberg's constant, which is defined as 10.973.731,534 / m.

Constant grav

Returns the gravitational constant, defined as $6,67259 \times 10^{-11} \text{ m}^3 / (\text{kg} \cdot \text{s}^2)$

Constant bohr_radius

Returns the Bohr's radius, defined as $0,529177249 \times 10^{-10}$ m.

Constant bohr_magneton

Returns the value of the Bohr's magneton, defined as $9,2740154 \times 10^{-24}$ joule / tesla.

Constant Boltzmann

Returns the value of the Boltzmann's constant, defined as $1,380658 \times 10^{-23}$ joule / $\square\text{K}$.

Constant accel

Returns the standard acceleration constant, defined as $9,80665$ m / sec².

Constant light

Returns the constant that represents the light speed in a vacuum, defined as $299.792.458$ m / sec.

Constant electron_charge

Returns the value of the electron charge, defined as $1,60217733 \times 10^{-19}$ coulomb.

Constant Planck

Returns the Planck's constant, defined as $6,6260755 \times 10^{-34}$ joule . sec.

Constant Avogadro

Returns the Avogadro's number, defined as $6,0221367 \times 10^{23}$ mols.

Constant amu

Returns the Atomic Mass Unit, defined as $1,6605402 \times 10^{-27}$ kg.

Constant pem

Returns the ratio between the proton and electron mass, defined as 1836,152701.

Constant ideal_gas

Returns the constant of the ideal gas, defined as 22,41410 litres / mols.

Constant Faraday

Returns the Faraday's constant, defined as 96485,309 coulomb / mol.

Constant Stefan_boltzmann

Returns the Stefan-Boltzmann's constant, defined as $5,67051 \times 10^{-8}$ Watt / (m² . $\square\text{K}^4$)

Constant golden

Returns the *Golden Ratio*, defined as $\frac{1+\sqrt{5}}{2}$.

Constant euler_gamma

Returns the value of the Euler's Gamma, defined as 0.5772156649015.

Techniques to Avoid the Repetition of Rules

This section describes different techniques that allow to avoid repeating rules. This helps to make models more readable.

Clause Else

When the clause **portInTransition** is used (see section12.3), it is possible to use the clause **else** to give an alternative rule in case that none of the rules evaluates to true.

Figure 11.19 shows a short example where the *Else* clause is used. The default local transition for the cells in this model is *default_rule*. In addition, cell (13,13) defines a special function to be used when an external event arrives through port *In*. If none of the conditions for the rules that make this functions is satisfied, then the *else* clause sets *default_rule* as the function to be evaluated.

```
[demoModel]
type: cell
...
link: in in@demoModel(13,13)
localTransition: default_rule
portInTransition: in@demoModel(13,13) another_rule

[default_rule]
rule: ...
...
rule: ...

[another_rule]
rule: 1 1000 { portValue(thisPort) = 0 }
...
else: default_rule
```

Figure 350: Example of the Else clause

The *Else* clause can point to any valid transition function. Care must be taken to avoid circular references, as in the example shown next.

```
[another_rule1]
rule: 1 1000 { portValue(thisPort) = 0 }
rule: 1.5 1000 { (0,0) = 5 }
rule: 3 1500 { (1,1) + (0,0) >= 1 }
else: another_rule2

[another_rule2]
rule: 1 1000 { (0,0) + portValue(thisPort) > 3 }
else: another_rule1
```

Figure 351: A circular reference produced by a bad use of the clause Else

CD++ will detect the special case shown in Figure 352, where the *else* clause references the same function being defined.

```
[another_rule]
rule: ...
rule: ...
else: another_rule
```

Figure 352: Example of a circular reference detected by the simulator

11.4.2 Preprocessor – Using Macros

CD++ has a preprocessor that will expand macros. If macros are not used, the preprocessor can be disabled using the command line argument **-b** to speed up model parsing.

Macros are usually defined in separate files that are included in the main .ma file by means of the preprocessor **#include** directive, which is of the form

```
#include(fileName)
```

where *fileName* is the name of the file that contains the definition of the macros. This file should be in the same directory where the main .ma file is.

More than one #include directive is allowed in the main .ma file, but no included files can have themselves the #include directive.

To define a macro, the directives **#BeginMacro** and **#EndMacro** are used.

A macro definition has the form:

```
#BeginMacro(macroName)
...
...definition of the macro...
...
#EndMacro
```

Figure 353: Definition of a macro

Macros can contain any valid text in any number of lines. The only restriction that applies is that they can not be used in the same file they are defined.

To expand a macro, the **#Macro** directive should be used in the place where the macro should be expanded. A #macro directive is of the form

```
#Macro(macroName)
```

An included file can contain any number of macro definitions. Any text in these files that is outside the macro definitions is ignored. If a required macro is not found, an error will be reported.

An *#include* directive can be placed at any line of the .ma file, as long as the macros therein defined are used after the *#include*.

A macro can not make use of another macro.

Within a .ma file, the preprocessor allows comments. Comments begin with a % . All text between the % and the end of the line is ignored.

```
% Here begins the rules
Rule : 1 100 { truecount > 1 or (0,0,1) = 2 } % Validate the existence
% of another individual.
```

Figure 354: A .ma file with comments

Section 12.5 shows a model where macros are used.

For special considerations regarding files created by the preprocessor, please see *Appendix C*.

11 Appendix C – Examples

The “Life Game”

The *Life Game* was presented in Scientific American by the well known mathematician Martin Gardner. In this game, living cells will live or die. The rules for life evolution are as follows:

- An active cell will remain in this state if it has two or three active neighbors.
- An inactive cell will pass to active state if it has two active neighbors exactly.
- In any other case, the cell will die

The implementation of this model in CD++ is as follows:

```
[top]
components : life

[life]
type : cell
width : 20
height : 20
delay : transport
border : wrapped
neighbors : life(-1,-1) life(-1,0) life(-1,1)
neighbors : life(0,-1) life(0,0) life(0,1)
neighbors : life(1,-1) life(1,0) life(1,1)
initialvalue : 0
initialrowvalue : 1 00010001111000000000
initialrowvalue : 2 00110111100010111100
initialrowvalue : 3 00110000011110000010
initialrowvalue : 4 00101111000111100011
initialrowvalue : 10 01111000111100011110
initialrowvalue : 11 00010001111000000000
localtransition : life-rule

[life-rule]
rule : 1 100 { (0,0) = 1 and (truecount = 3 or truecount = 4) }
rule : 1 100 { (0,0) = 0 and truecount = 2 }
rule : 0 100 { t }
```

Figure 355: Implementation of the Game of Life

A bouncing object

The following is the specification of a model that represents an object in movement that bounces against the borders of a room. This example is ideal to illustrate the use of a non toroidal cellular automata, where the cells of the border have different behavior to the rest of the cells.

For the representation of the problem, 5 different values are used for the states of each cell, these values are:

- 0 = represents an empty cell.
 1 = represents the object moving toward the south east.
 2 = represents the object moving toward the north east.
 3 = represents the object moving toward the south west.
 4 = represents the object moving toward the north west.

The specification of the model is:

```

[top]
components : rebound

[rebound]
type : cell
width : 20
height : 15
delay : transport
defaultDelayTime : 100
border : nowrapped
neighbors : rebound(-1,-1)      rebound(-1,1)
neighbors :      rebound(0,0)
neighbors : rebound(1,-1)      rebound(1,1)
initialvalue : 0
initialrowvalue : 13 00000000000000000010
localtransition : move-rule
zone : cornerUL-rule { (0,0) }
zone : cornerUR-rule { (0,19) }
zone : cornerDL-rule { (14,0) }
zone : cornerDR-rule { (14,19) }
zone : top-rule { (0,1)..(0,18) }
zone : bottom-rule { (14,1)..(14,18) }
zone : left-rule { (1,0)..(13,0) }
zone : right-rule { (1,19)..(13,19) }

[move-rule]
rule : 1 100 { (-1,-1) = 1 }
rule : 2 100 { (1,-1) = 2 }
rule : 3 100 { (-1,1) = 3 }
rule : 4 100 { (1,1) = 4 }
rule : 0 100 { t }

[top-rule]
rule : 3 100 { (1,1) = 4 }
rule : 1 100 { (1,-1) = 2 }
rule : 0 100 { t }

[bottom-rule]
rule : 4 100 { (-1,1) = 3 }
rule : 2 100 { (-1,-1) = 1 }
rule : 0 100 { t }

[left-rule]
rule : 1 100 { (-1,1) = 3 }
rule : 2 100 { (1,1) = 4 }
rule : 0 100 { t }

[right-rule]
rule : 3 100 { (-1,-1) = 1 }
rule : 4 100 { (1,-1) = 2 }
rule : 0 100 { t }

```

```

[cornerUL-rule]
rule : 1 100 { (1,1) = 4 }
rule : 0 100 { t }

[cornerUR-rule]
rule : 3 100 { (1,-1) = 2 }
rule : 0 100 { t }

[cornerDL-rule]
rule : 2 100 { (-1,1) = 3 }
rule : 0 100 { t }

[cornerUR-rule]
rule : 4 100 { (-1,-1) = 1 }
rule : 0 100 { t }

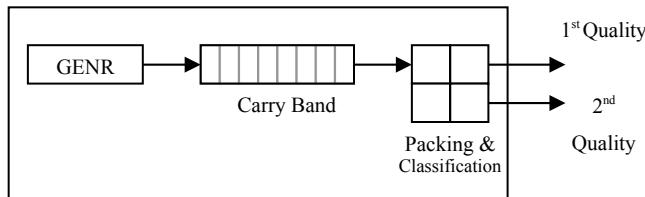
```

Figure 356: Implementation of the Rebound of an Object

Classification of raw materials

The aim of this example is to show the use of special behavior that can be given to a cell when an external event arrives through an input port. We have a model that represents the packing and classification of certain raw material that contains 30% of carbon approximately. The model is made of a machine that loads 100 grams fractions of that substance in a carrying band. One a fraction reaches the end of the band, it is processed by a packager that takes these fractions until a kilogram is obtained. Then, the packed substance is classified. If each packet contains 30 \square 1 % of carbon, it is classified as of first quality; otherwise, it will be of second quality.

The model uses the atomic model *Generator* that generates values (in this case always the value 1) each x seconds (where x has an Exponential distribution with average 3). These values are passed to the carry band, represented by a cellular mode. At the end of the band, another cellular model



makes the packaging and selection.

Figure 357: Coupling structure for the Classification of Substances

The following is the specification of the model:

```

[top]
components : genSubstances@Generator queue packing
out : outFirstQuality outSecondQuality
link : out@genSubstances in@queue
link : out@queue in@packing
link : out1@packing outFirstQuality
link : out2@packing outSecondQuality

```

```

[genSubstances]
distribution : exponential
mean : 3
initial : 1
increment : 0

[queue]
type : cell
width : 6
height : 1
delay : transport
defaultDelayTime : 1
border : nowrapped
neighbors : queue(0,-1) queue(0,0) queue(0,1)
initialvalue : 0
in : in
out : out
link : in in@queue(0,0)
link : out@queue(0,5) out
localtransition : queue-rule
portInTransition : in@queue(0,0) setSubstance

[queue-rule]
rule : 0      1 { (0,0) != 0 and (0,1) = 0 }
rule : { (0,-1) } 1 { (0,0) = 0 and (0,-1) != 0 and not isUndefined((0,-1)) }
rule : 0      3000 { (0,0) != 0 and isUndefined((0,1)) }
rule : { (0,0) } 1 { t }

[setSubstance]
rule : { 30 + normal(0,2) } 1000 { t }

[packing]
type : cell
width : 2
height : 2
delay : transport
defaultDelayTime : 1000
border : nowrapped
neighbors : packing(-1,-1) packing(-1,0) packing(-1,1)
neighbors : packing(0,-1) packing(0,0) packing(0,1)
neighbors : packing(1,-1) packing(1,0) packing(1,1)
in : in
out : out1 out2
initialvalue : 0
initialrowvalue : 00
initialrowvalue : 100
link : in in@ packing(0,0)
link : in in@ packing(1,0)
link : out@ packing(0,1) out1
link : out@ packing(1,1) out2
localtransition : packing-rule
portInTransition : in@packing(0,0) add-rule
portInTransition : in@packing(1,0) incQuantity-rule

[packing-rule]
rule : 0 1000 { isUndefined((1,0)) and isUndefined((0,-1)) and (0,0) = 10 }
rule : 0 1000 { isUndefined((-1,0)) and isUndefined((0,-1)) and (1,0) = 10 }

```

```

rule : { (0,-1) / (1,-1) } 1000 { isUndefined((-1,0)) and
isUndefined((0,1))
                                and (1,-1) = 10 and abs( (0,-1) / (1,-1) - 30 ) <=
1 }
rule : { (-1,-1) / (0,-1) } 1000 { isUndefined((1,0)) and
isUndefined((0,1))
                                and (0,-1) = 10 and abs( (-1,-1) / (0,-1) - 30 ) >
1 }
rule : { (0,0) } 1000 { t }

[add-rule]
rule : { portValue(thisPort) + (0,0) } 1000 { portValue(thisPort) != 0 }
rule : { (0,0) } 1000 { t }

[incQuantity-rule]
rule : { 1 + (0,0) } 1000 { portValue(thisPort) != 0 }
rule : { (0,0) } 1000 { t }

```

Figure 358: Implementation of the Model to Classify Substances

The cellular model **queue** that represents the carry band makes use of the **portInTransition** clause. As it was mentioned earlier, this clause is used to set the rule that will be evaluated when an external event is received by the cell through the specified port. This clause is then used again in the definition of the model *Packing* set the behavior of the cells upon the reception of a raw material from the carry band.

Life Game – 3D

The next example is an adaptation of the *Game of the Life* to a three dimensional space.

Figure 359 shows the model definition and Figure 360 lists the contents of file “3d-life.val” that contains the initial values for the cell.

```

[top]
components : 3d-life

[3d-life]
type : cell
dim : (7,7,3)
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : 3d-life(-1,-1,-1) 3d-life(-1,0,-1) 3d-life(-1,1,-1)
neighbors : 3d-life(0,-1,-1) 3d-life(0,0,-1) 3d-life(0,1,-1)
neighbors : 3d-life(1,-1,-1) 3d-life(1,0,-1) 3d-life(1,1,-1)
neighbors : 3d-life(-1,-1,0) 3d-life(-1,0,0) 3d-life(-1,1,0)
neighbors : 3d-life(0,-1,0) 3d-life(0,0,0) 3d-life(0,1,0)
neighbors : 3d-life(1,-1,0) 3d-life(1,0,0) 3d-life(1,1,0)
neighbors : 3d-life(-1,-1,1) 3d-life(-1,0,1) 3d-life(-1,1,1)
neighbors : 3d-life(0,-1,1) 3d-life(0,0,1) 3d-life(0,1,1)
neighbors : 3d-life(1,-1,1) 3d-life(1,0,1) 3d-life(1,1,1)
initialvalue : 0
initialCellsValue : 3d-life.val
localtransition : 3d-life-rule

```

```
[3d-life-rule]
rule : 1 100 { (0,0,0) = 1 and (truecount = 8 or truecount = 10) }
rule : 1 100 { (0,0,0) = 0 and truecount >= 10 }
rule : 0 100 { t }
```

Figure 359. Implementation of the Game of Life – 3D

(0,0,0) = 1	(2,4,1) = 1	(5,1,2) = 1
(0,0,2) = 1	(2,4,2) = 1	(5,2,0) = 1
(1,0,0) = 1	(2,5,0) = 1	(5,2,2) = 1
(1,0,1) = 1	(2,6,1) = 1	(5,3,0) = 1
(1,1,1) = 1	(3,2,1) = 1	(5,3,1) = 1
(1,2,0) = 1	(3,5,1) = 1	(5,5,1) = 1
(1,2,2) = 1	(3,5,2) = 1	(5,5,2) = 1
(1,3,2) = 1	(3,6,1) = 1	(5,6,0) = 1
(1,4,2) = 1	(3,6,2) = 1	(6,0,0) = 1
(1,5,0) = 1	(4,1,2) = 1	(6,1,1) = 1
(1,5,1) = 1	(4,2,0) = 1	(6,1,2) = 1
(1,6,0) = 1	(4,2,1) = 1	(6,3,0) = 1
(1,6,1) = 1	(4,4,1) = 1	(6,3,2) = 1
(2,1,2) = 1	(4,5,0) = 1	(6,4,2) = 1
(2,1,0) = 1	(4,5,2) = 1	(6,5,1) = 1
(2,3,1) = 1	(4,6,0) = 1	(6,6,0) = 1
(2,3,2) = 1	(4,6,2) = 1	(6,6,2) = 1

Figure 360: Initial values for the cells of the Game of Life – 3D

Use of Macros

The following example shows how macros can be used to write a new version of the *Game of the Life* for a 4 dimensional space. Macros can be defined in external files that are included in the main .ma file. More than one macro definition is may be included per file, but no macro can make use of an existing macro. A macro is defined between the #BeginMacro and a #EndMacro directives. All other text is ignored. The next figures show the contents of the four files that are used to completely define the new model.

```
#include(life.inc)
#include(life-1.inc)

[top]
components : life

[life]
type : cell
dim : (2,10,3,4)
delay : transport
defaultDelayTime : 100
border : wrapped
neighbors : life(-1,-1,0,0) life(-1,0,0,0) life(-1,1,0,0)
neighbors : life(0,-8,0,0) life(0,-1,0,0) life(0,0,0,0) life(0,1,0,0)
neighbors : life(1,-1,0,0) life(1,0,0,0) life(1,1,0,0)
initialvalue : 0
initialCellsValue : life.val
localtransition : life-rule
```

```
[life-rule]
% Comment: Here starts the definition of rules
rule : 1          100 { #macro(Heat) or #macro(Rain) }
rule : 0          100 { (0,0,0,0) = ? OR (0,0,0,0) = 2 }
#macro(rule1)    % Another comment: A macro is invoked
rule : 1          100 { (0,0,0,0) = (1,0,0,0) AND (0,0,0,0) > 1 }
#macro(rule2)
```

Figure 361. Implementation of the Game of Life with 4 dimensions and using macros

```
(0,0,0,0) = ?
(1,0,0,0) = 25
(0,0,1,0) = 21
(0,1,2,2) = 28
(1, 4, 1,2) = 17
(1, 3, 2,1) = 15.44
```

Figure 362. File life.val that contains the initial values for the Game of Life in 4D

```
This is a comment: The macro Rule3 assigns the value 0 if the cell's value
is 3, and 4 if the cell's value is negative.

#BeginMacro(rule3)
rule : 0 100 { (0,0,0,0) = 3 }
rule : 4 100 { (0,0,0,0) < 0 }
#EndMacro

#BeginMacro(rule1)
rule : 0 100 { (0,0,0,0) + (1,0,0,0) + (1,1,0,0) + (0,-8,0,0) = 11 }
#EndMacro

#BeginMacro(Heat)
(0,0,0,0) > 30
#EndMacro
```

Figure 363. File life.inc that contains some macros used in the Game of Life 4D

```
#BeginMacro(Rule2)
rule : 0 100 { (0,0,0,0) = 7 }
rule : { (0,0,0,0) + 2 } 100 { t }
#EndMacro

#BeginMacro(Rain)
(0,-8,0,0) > 25
#EndMacro
```

Figure 364. File life-1.inc that contains the remaining macros for the Game of Life 4D

Appendix D– The preprocessor and temporary files.

When the preprocessor is used to resolve macros (by default the preprocessor is enabled), it will create a temporary file for the model with all macros expanded and all the comments erased. This temporary file is then passed to the simulator for its interpretation. If the use of the preprocessor with the parameter **-b** is disabled and macros are used, the model will not be processed correctly.

The name of the temporary file is the value returned by the instruction *tmpnam* of the GCC. The directory where the temporary files are located will be selected according to the following criteria:

1. When CD++ is compiled, the name of directory defined by *P_tmpdir* <stdio.h> will be used, unless this is the root directory.

In Linux this variable usually has the value: “/TMP”, while in the version of the GCC 2.8.1 for Windows–32 bits, this variable references to the root directory of the disk unit that is in use.

2. If *P_tmpdir* points to the root directory, then the name defined by the environment variable **TEMP** will be used.
3. If no **TEMP** variable is defined, then the value of the environment variable **TMP** will be used.
4. Finally, if the **TMP** is neither defined, the current directory will be used.

Manual of the VRML GUI for the CD++ Tool

This manual describes in detail how to use the VRML GUI for the visualization of the results of the Cell-DEVS models. This GUI is developed with VRML and Java, so it is platform portable, and can be run on various environment. JDK1.1 and above is necessary to run this GUI.

Install CosmoPlayer

To run this java applet GUI, virtual reality software, for example CosmoPlayer, should be installed in the computer. This GUI uses **cosmo_win95nt_eng.exe**, which is free and can be download from the Internet. This version is for both MS Internet Explorer and Netscape, and compatible with MS Windows 95, 98, 98SE, and NT. It does not work on Windows ME, 2000 and XP.

cosmo_win95nt_eng.exe can be found in many web sites, such as,
<ftp://ftp.cai.com/pub/marketing/comosoftware/> **cosmo_win95nt_eng.exe**, if you use ftp, or
<http://endeavor.med.nyu.edu/download/win95/>, if you use http.

To install CosmoPlayer:

- (1) Download the file **cosmo_win95nt_eng.exe** to your hard disk in any favorite directory you want.
- (2) Find the **cosmo_win95nt_eng.exe** file in the directory and double-click it to begin installation, then follow the instructions on the screen. You can use all the default choices.
- (3) Close your browser completely after installation and reboot your system. Now your

system can view the VRML files and open this VRML GUI.

To run the GUI, decompress the **vrmlgui.zip** file under Windows environments. All the source code and some necessary files will be in the directory **vrmlgui**; all the necessary classes for compiling and running the GUI will be in directory **vrmlgui/netscape** and **vrmlgui/vrml**. For instance, we will consider that you will install **vrmlgui** just under **C:**. To use this tool, you should set up class path. For example, in MS-DOS, use the following command line:

```
set classpath=.;C:\vrmlgui
```

We recommend to add this line in **Autoexec.bat** file, then the computer will set up this class path automatically every time when you start or re-boot your system.

In **C:\vrmlgui** directory, double click the HTML file **vrml3dgui.html**, and the GUI will be show on the screen.

Introduction to CosmoPlayer VRML browser [1]

Every VRML browser provides a built-in GUI for the user to navigate in the Virtual World. The following Fig. 1 is the CosmoPlayer VRML browser. There is a built-in GUI (controlpanel) in the bottom. The controlpanel contains several buttons (indicated by the arrows). Clicking on these buttons in this VRML GUI with the mouse allows you to perform the following operations:

1. Select viewpoints for the VRML scene
2. GO: Click and then drag to move in any direction. It can be used as ZOOM button. Dragging the mouse to the top can enlarge the scene, and dragging the mouse to the bottom can reduce the scene.
3. ROTATION: Click and then drag to see the scene up or down or from side to side, can rotate the scene and get any viewing direction.
4. TRANSLATION: Click and then drag to slide straight up and down or to slide right or left.
5. UNDO: go back to the previous scene.
6. REDO: go forward to the next scene.
7. For more information about the CosmoPlayer, click button 7, a HTML page will be shown. This page gives a detailed description of the CosmoPlayer.

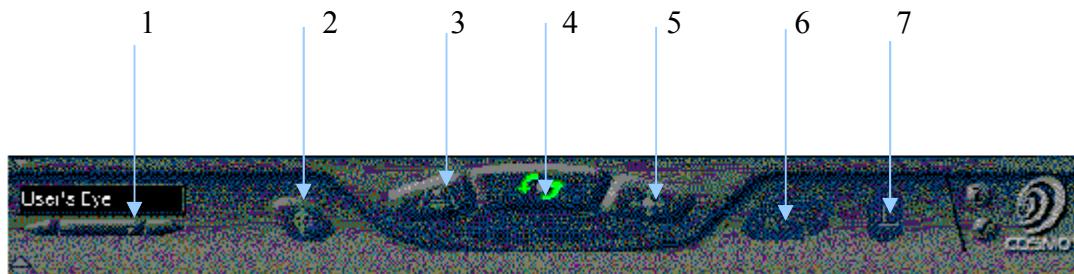


Figure 365: The cosmo Player

When you pass the mouse pointer over an active object, the pointer will change to a starburst to indicate that the node can be selected.



The node can be selected

VRML GUI

The VRML GUI is a 3D GUI, which can be used for the visualization of Cell-DEVS execution results. The following sections explore the functions of the GUI with examples. The GUI looks like the following Figure 366 when it is first open.

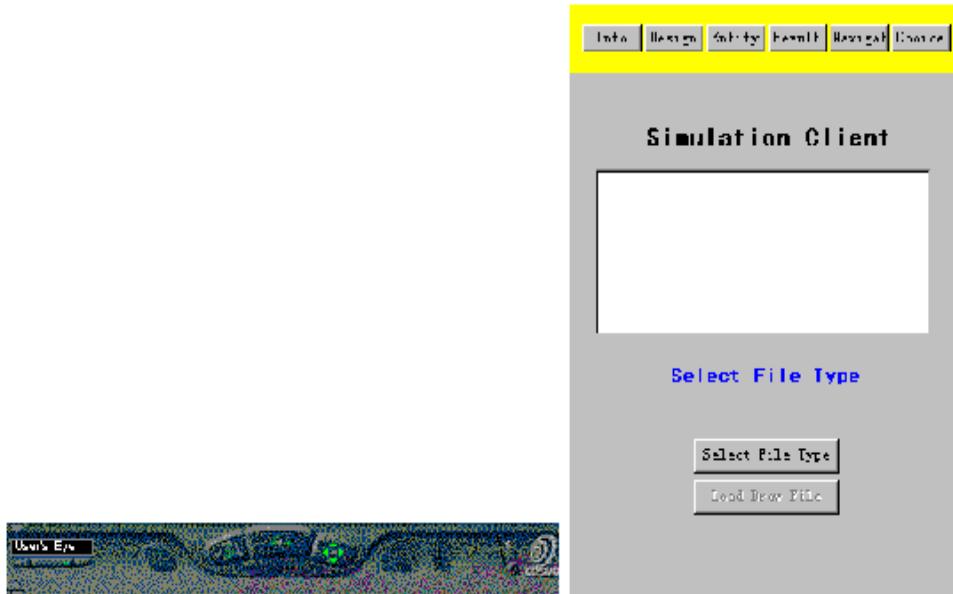


Figure 366: The VRML GUI

The left is the VRML scene area, and the right are the function panels for the user. The message "Select File Type" is displayed.

Info Panel

When the GUI is started, the InfoPanel in Figure 366 is presented, the user can load a result file for visualization. The result file records the state of each cell in the model [2]. The InfoPanel is shown when the GUI is first open, or by clicking the "Info" button on the top at any time.

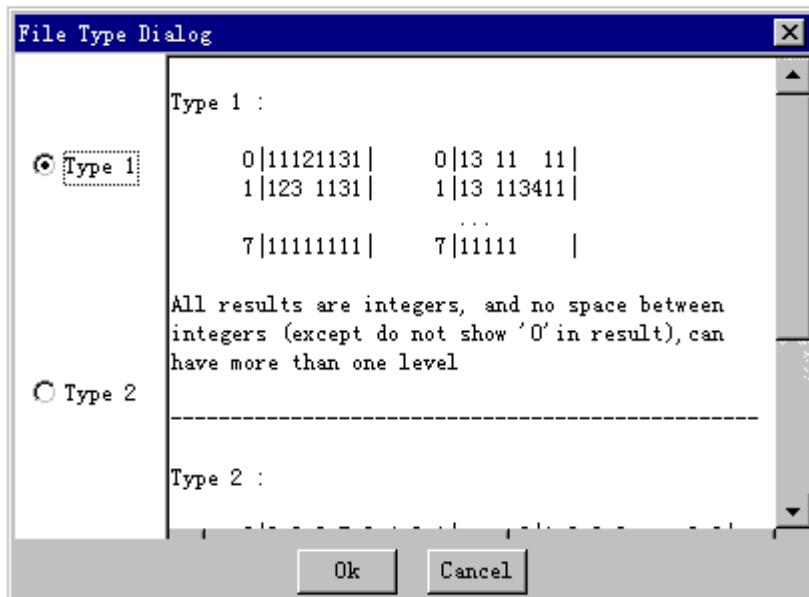


Figure 367: The File Type Dialog Box

- Click the “Select the file type” button, a dialog will be shown like Figure 367. Select “Type 1” or “Type 2” according to the file type, and click “OK” button.
- Click “Load Drw File” button, a load file dialog or a file name input dialog will be shown. Select a file with the file dialog, or input the file name in the file name input dialog, then click the “Load” or “OK” button

Then a message “Please Wait” will be displayed, the GUI begin to load the file and initiate the scene. When finished, the message will disappear, and you can switch to other panels by clicking the buttons on the top.

11.1.1 Color Selection Dialog

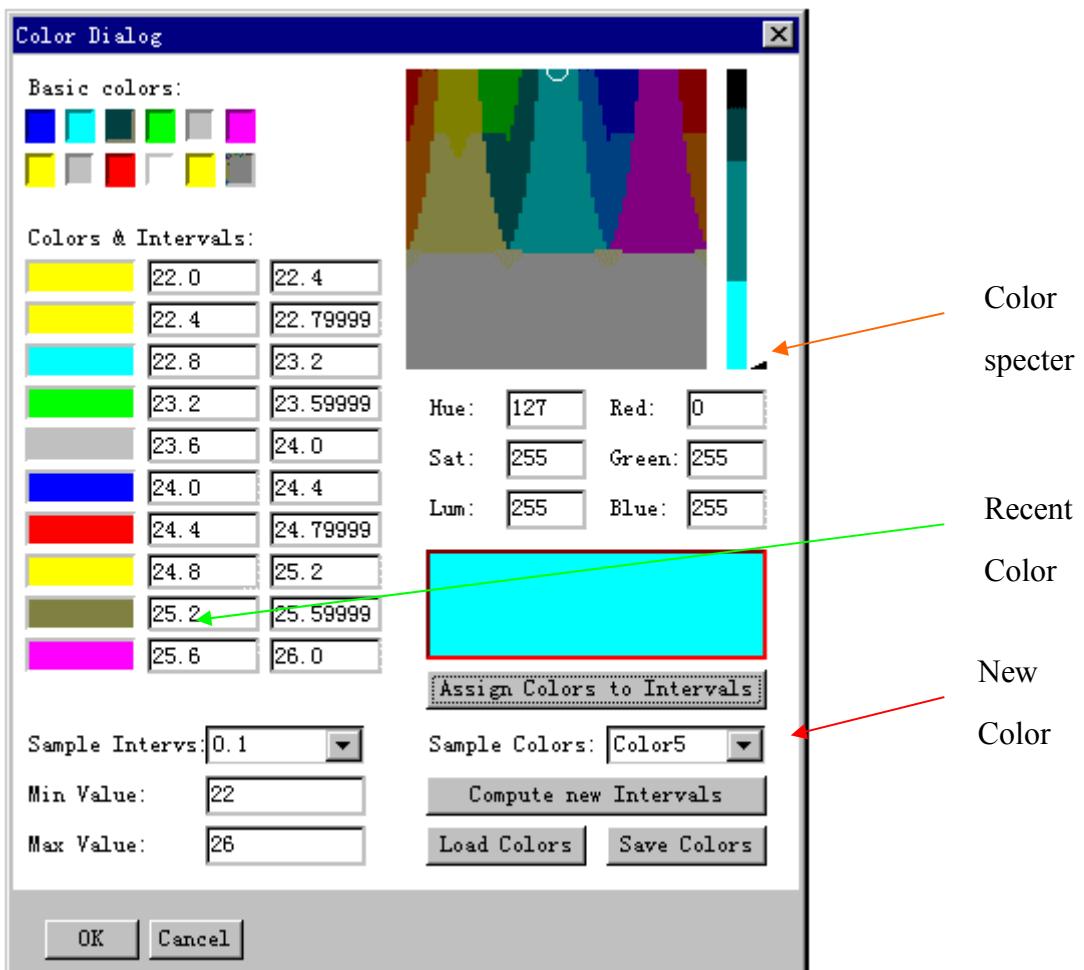


Figure 368: Selecting Colors

Select color for each value range with Color Dialog in Figure 368. This dialog can be brought up with "Select Colors" button in "Navigate" panel.

Set Colors for the Value Ranges

- **Set value ranges** according the values in the result file. There are several methods:
 - (1) Input value in "Min Value" and "Max Value" fields, then click "Compute new Intervals" button. The program will compute the interval, with the minimum value and the interval, all the value ranges will be calculated.
 - (2) Input value in "Min Value" and select an interval value in "Sample Intervals", then click "Compute new Intervals" button. With the minimum value and the interval, all the value ranges will be calculated.
 - (3) Input value in "Max Value" and select an interval value in "Sample Intervals", then click "Compute new Intervals" button. With maximum value and interval, the minimum value can be computer, then all the value ranges will be calculated.

- (4) Select a value in “Sample Intervals”. At this time, the minimum value is 0.0. With the minimum value and the interval, all the value ranges will be calculated.

- **Assign colors** to the value ranges, the steps:

- (1) Assign colors to all the 10 value ranges

Select one of the colors in “Sample Colors”, all the 10 value ranges will be assigned their own colors automatically.

- (2) Assign color to a specific value range

- Click the recent color of the specific value range
- Click one of the basic colors, or click on the “Color specter” panel to select new color, and the new color will be on the “New Color” panel.
- Click “Assign Color to Intervals” button, the new color will be assigned to the specific value range.

Don't display some nodes with specific values

Sometimes it is necessary not to display some nodes with values in specific value ranges.

Assign “White” color in Color Selection Dialog in Figure 368 for these specific value ranges, the nodes with values within these specific value ranges will not be displayed, as in Fig. 18 for color selection, and Figure 370 for result display.

If you want to display those not-displayed nodes, just assign other colors to these specific value ranges with Color Selection Dialog.

Save defined Colors and Intervals

The defined colors and Intervals can be saved to a *color* file.

Click “Save Colors” button, because the applet usually can not access the local file, the color file usually can not be saved. To ensure the file can be saved, the applet should be signed [3].

The *color* file has three parts as the following:

- (1) A string “VALIDSAVEDFILE”.
- (2) The colors, such as, 175,95,71, three integers are used to represent a color.
- (3) The intervals, such as, 0.0,0.4

The following file is the corresponding color file for example Figure 368. The first line “VALIDSAVEDFILE” is used to identify a valid *color* file. When the color file is loaded, this string is checked first to confirm it is a valid color file.

```
VALIDSAVEDFILE
255,255,0
255,255,0
0,255,255
0,255,0
215,175,163
0,0,255
255,0,0
255,200,0
175,95,71
255,0,255
22.0,22.4
```

22.4,22.79999999999997
22.8,23.2
23.2,23.5999999999998
23.6,24.0
24.0,24.4
24.4,24.7999999999997
24.8,25.2
25.2,25.5999999999998
25.6,26.0

Color File Example

Load color file

Click “Load Colors” button, because the applet can not access the local file, the local color file usually can not be loaded. However, because this applet is also loaded from the local disk, we can use an URL path to open the file and read it. A file name input dialog will be shown and the *color* file name should be input. If the applet is signed [3], a normal file load dialog will open, and the user can select the *color* file. Then the color values and their value ranges are read and assigned to the corresponding panels and text fields in the Color Selection Dialog in Figure 368. If the above color file is loaded, the Color Selection Dialog will looks like Figure 368.

Navigation Panel

In this panel, the “Color Selection Panel” in Figure 368 can be shown by clicking “Select Color” button. Other functions in this panel are as follows, the changes have effect on all the nodes in the scene.

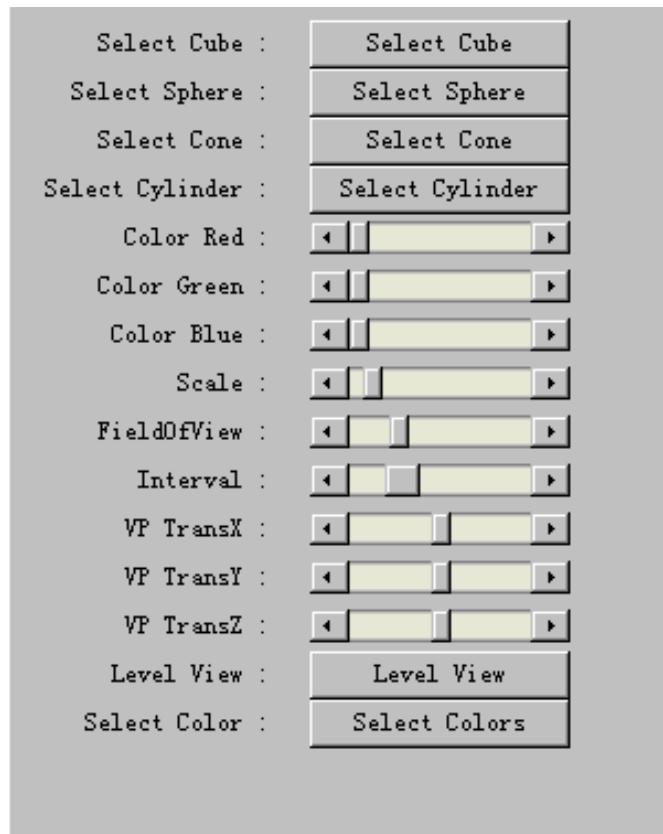


Figure 369: The Navigation Panel

Change the geometry

- Click “Select Cube” button, to select box as the nodes’ geometry in the scene as in Figure 369.
- Click “Select Sphere” button, to select sphere as the nodes’ geometry in the scene when the scene generation is finished, as in Figure 369.
- Click “Select Cone” button, to select cone as the nodes’ geometry in the scene when the scene generation is finished, as in Figure 369.
- Click “Select Cylinder” button, to select cylinder as the nodes’ geometry in the scene when the scene generation is finished, as in Figure 369.

This geometry selection affects all the nodes in the scene.

Change the color, use the three sliders “Color Red”, “Color Green” and “Color Blue” to change the colors of all the nodes in the scene.

Scale the Node, use the slider “Scale” to change the size of all the nodes in the scene. When the interval between the nodes reaches a minimum value, the nodes can’t be scaled further.

Move the Viewpoint, use three sliders “VP TransX”, “VP TransY” and “VP TransZ” to change the position of the viewpoint when “User’s Eye” viewpoint is active.

Change the Viewpoint, use built-in VIEPOINT button, as in Figure 369.

Change the Viewing Angle, use the slider “FieldOfView” to change the Viewing Angle of the scene.

Result Panel

Navigate the visualization. The “Output Files” text field displays the recent result. The “Time” list lists all the times displayed before according to their sequence, and the recent time is selected. The “Level” list lists all the levels in the 3D scene.

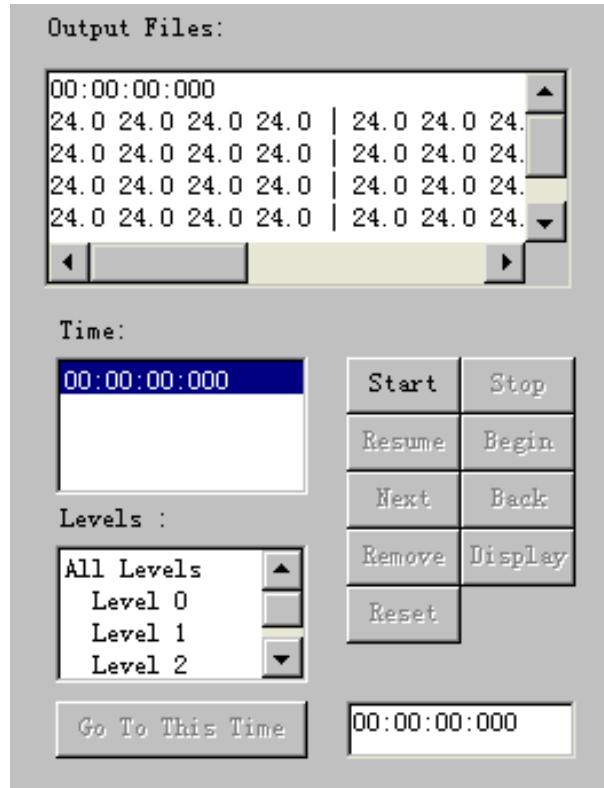
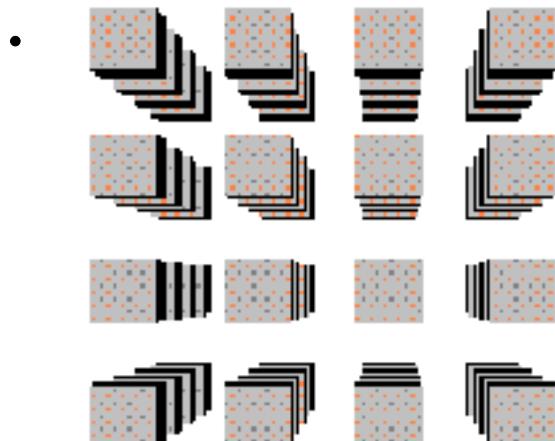


Figure 370: The Result Panel

At first, the “Result” panel looks like above Figure 370, and the scene looks like , showing the result at time 00:00:000. Only “Start” button is enabled.

- **Start the Visualization**, click “Start” button, the scene begin to display the result according to the time sequence continuously. The button status will look like as Figure 372. You can only stop the display.



- **Stop the display**, click “Stop” button when the scene generation is completed, the button status will look like as Figure 373.

Figure 371: Resulting scene at time 00:00:00:000 when only start button is enabled

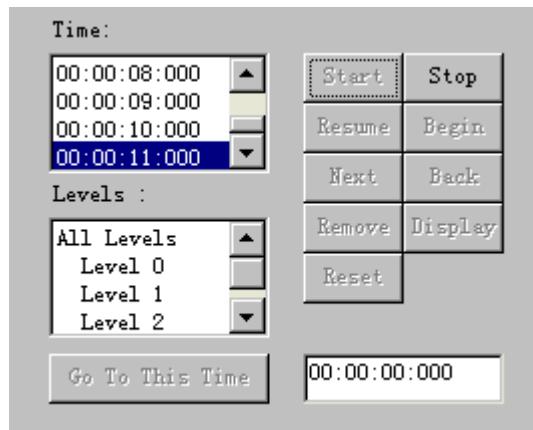


Figure 372: Result Panel when start button is pressed.

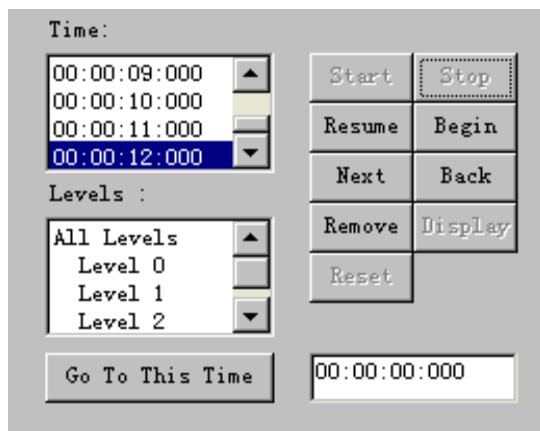


Figure 373: Result Panel when stop button is pushed.

- **Continue to display the result according to the time sequence when stop**, click “Resume” button, and the button status will go back to Figure 372.
- **Display the result in the next time**, click “Next” button.
- **Display the result in the last time**, click “Back” button.
- **Remove the layer**

- Select the layer needed to removed in “Level” list
- Click “Remove” button

The button status will look like as Fig. 10. ‘D’ is shown before the level name, indicating that this layer is removed.

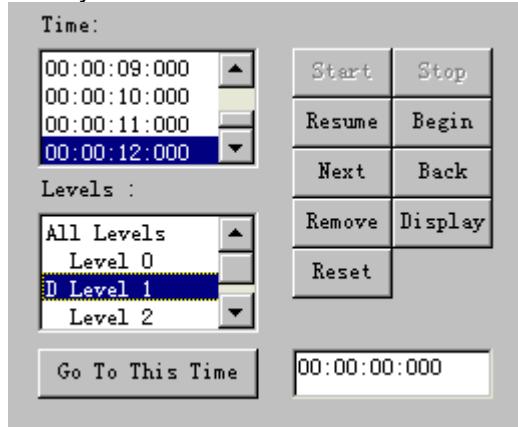


Figure 374: Remove layer panel.

- **Re-display the removed layers**
 - Select the removed layer, as in Figure 374
 - Click “Display” button

‘D’ will not be shown before the level name, as in Figure 374
- **Reset the Scene**, click “Reset” button, all the layers will be displayed, including the removed layers
- **Come back to any time point**, when stop, click an item in “Time” list, then the display will be at this time. The display will begin at this time.
- **Navigate in the scene**, use the built-in GUI of the browser
- **Go to any time displayed before**, when stop, click a time in “Time” list, or input a time in the text field, then press “Return / Enter” key, or click the “Go to This Time” button. The display will begin at this time.
- **View the result from the beginning**, when stop, click “Begin” button, the display will come to the first result, and begin the display from there.

Entity Panel

Edit any node in the scene. The “Entity” list lists all the nodes in the scene. The node should be selected first to be edited.

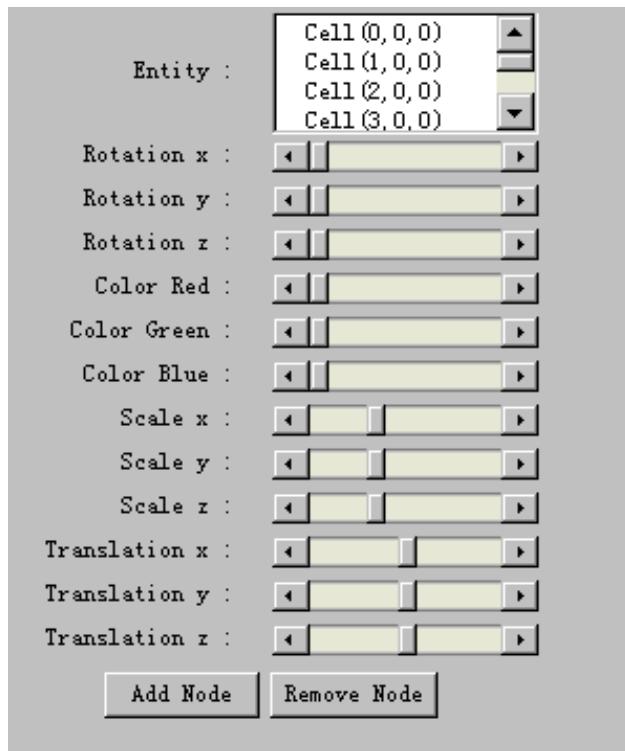


Figure 375: The Entity Panel

- **Select a node**, click an item in the “Entity” list, or move the mouse pointer onto the node, when the pointer changes to a starburst, click it.
- **Rotate the selected node**, use three sliders “Rotation x”, “Rotation y” and “Rotation z” to rotate the selected node, as in Figure 375.
- **Change the color**, use three sliders “Color Red”, “Color Green” and “Color Blue” to change the color of the selected node, as in Figure 375.
- **Scale the selected Node**, use three sliders “Scale x”, “Scale y” and “Scale z” to change the size of the selected node, as in Figure 375.
- **Remove or Display the selected node**, click “Remove Node” button, or “Add Node” button to remove the selected node, or display the removed node, as in Figure 375.

Examples

All the examples will use the following fragment of a result file.

Line : 166 - Time: 00:00:00:000

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0	1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0	2 24.0 24.0 24.0 24.0 2 24.0 24.0 24.0 24.0 2 24.0 24.0 24.0 24.0 2 24.0 24.0 24.0 24.0 2 24.0 24.0 24.0 24.0	3 24.0 24.0 24.0 24.0 3 24.0 24.0 24.0 24.0 3 24.0 24.0 24.0 24.0 3 24.0 24.0 24.0 24.0 3 24.0 24.0 24.0 24.0	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+			

Line : 340 - Time: 00:00:01:000

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0 0 24.0 24.0 24.0 24.0	1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0 1 24.0 24.0 24.0 24.0	2 24.0 24.0 24.0 24.0 2 24.0 24.0 71.4 24.0 2 24.0 24.0 24.0 24.0 2 24.0 24.0 24.0 24.0 2 24.0 24.0 24.0 24.0	3 24.0 24.0 24.0 53.5 3 24.0 24.0 24.0 24.0 3 24.0 24.0 24.0 24.0 3 24.0 24.0 24.0 -1.6 3 24.0 24.0 24.0 24.0	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+				

Line : 644 - Time: 00:00:02:000

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
0 25.2 24.0 25.2 23.7 0 24.0 24.0 27.8 24.2 0 23.0 24.0 23.0 21.5 0 22.5 24.0 22.5 22.7	1 22.5 24.0 24.4 24.9 1 24.0 25.9 25.9 23.0 1 22.5 24.0 24.4 20.5 1 22.5 21.1 22.5 22.5	2 25.2 25.9 27.1 25.6 2 27.8 25.9 25.9 26.1 2 23.0 25.9 24.9 23.4 2 22.5 24.0 26.3 22.7	3 25.2 26.4 27.1 23.1 3 24.2 25.9 26.1 26.1 3 23.0 22.0 24.9 25.3 3 24.2 24.0 24.2 21.2	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+				

Line : 968 - Time: 00:00:03:000

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
0 24.1 24.5 24.7 24.0 0 24.5 24.7 24.6 24.2 0 23.4 23.8 24.0 23.5 0 23.1 23.6 24.0 23.3	1 24.0 24.6 24.7 23.8 1 24.2 24.5 24.6 24.4 1 23.3 23.5 24.0 23.6 1 23.2 23.8 23.8 22.8	2 24.7 25.1 25.1 24.6 2 24.6 25.1 25.9 24.7 2 24.0 24.4 24.4 24.1 2 24.0 24.2 24.2 23.9	3 24.5 24.6 25.1 24.9 3 24.7 25.3 25.2 24.6 3 23.9 24.4 24.5 23.4 3 23.6 24.0 24.2 23.9	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+				

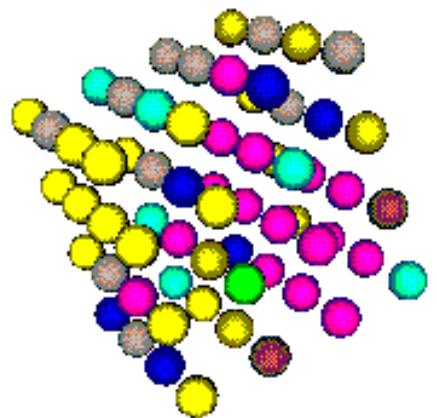
Line : 1292 - Time: 00:00:04:000

0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+
0 24.1 24.3 24.4 24.2 0 24.2 24.4 24.5 24.2 0 23.8 24.1 24.1 23.9 0 23.8 24.0 24.0 23.8	1 24.1 24.3 24.4 24.2 1 24.2 24.5 24.5 24.2 1 23.9 24.1 24.1 23.8 1 23.8 23.9 24.0 23.8	2 24.4 24.6 24.6 24.4 2 24.5 24.6 24.7 24.5 2 24.1 24.3 24.3 24.1 2 24.0 24.2 24.3 24.0	3 24.4 24.6 24.6 24.3 3 24.4 24.6 24.7 24.5 3 24.1 24.2 24.3 24.1 3 24.0 24.2 24.2 24.0	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+	+-----+				

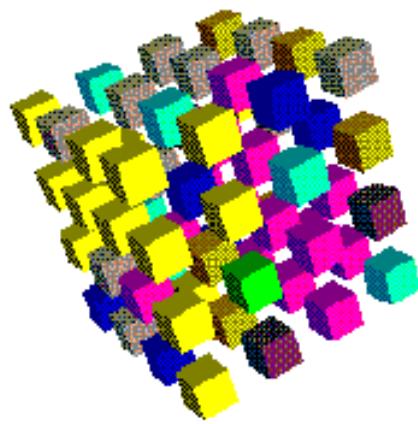
Color Selection

After color selection, the nodes will be displayed with the colors corresponding to their value, seen in Figure 376.

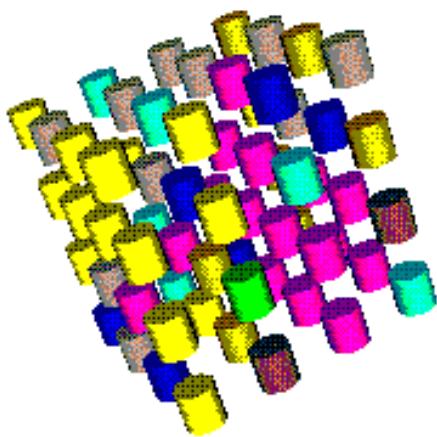
Geometry Selection



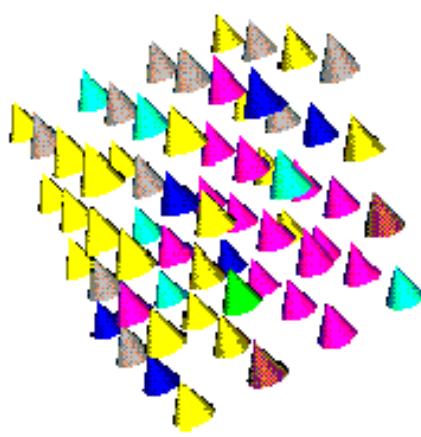
(1) Sphere



(2) Box



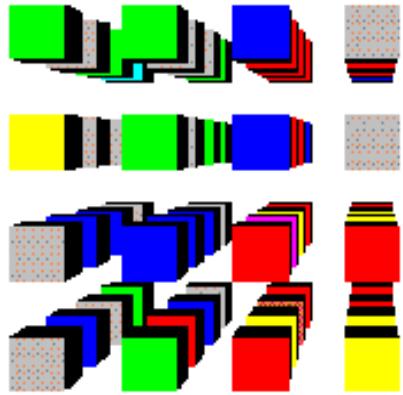
(3) Cylinder



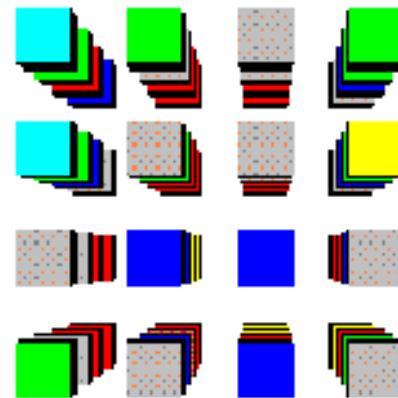
(4) Cone

Figure 376: *Geometry Selection*.

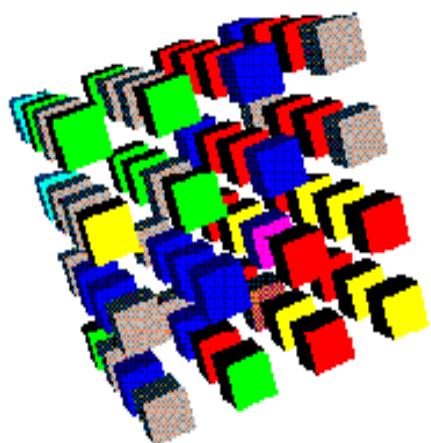
Different Viewpoint



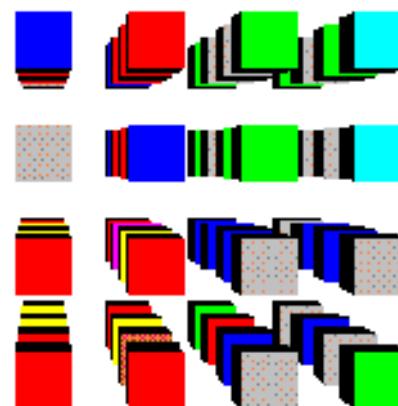
(1) User's Eye



(2) Side view 1



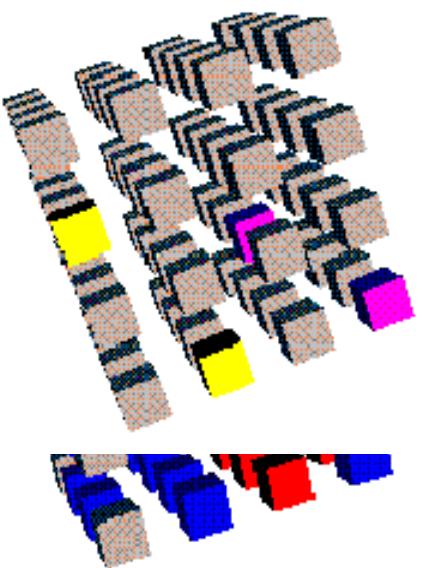
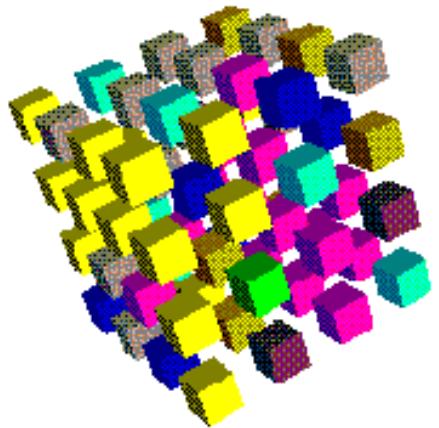
(3) Side view 2



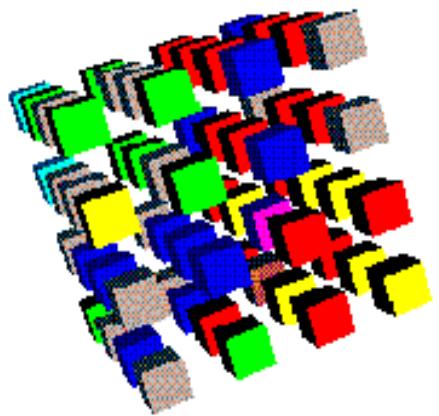
(4) Random viewpoint

Figure 377: Different Viewpoints

Continues display



(2)



(1) Time: 00:00:04:000

(2) Time: 00:00:05:000

Figure 378: Display Continued

Edit a Node in the Scene

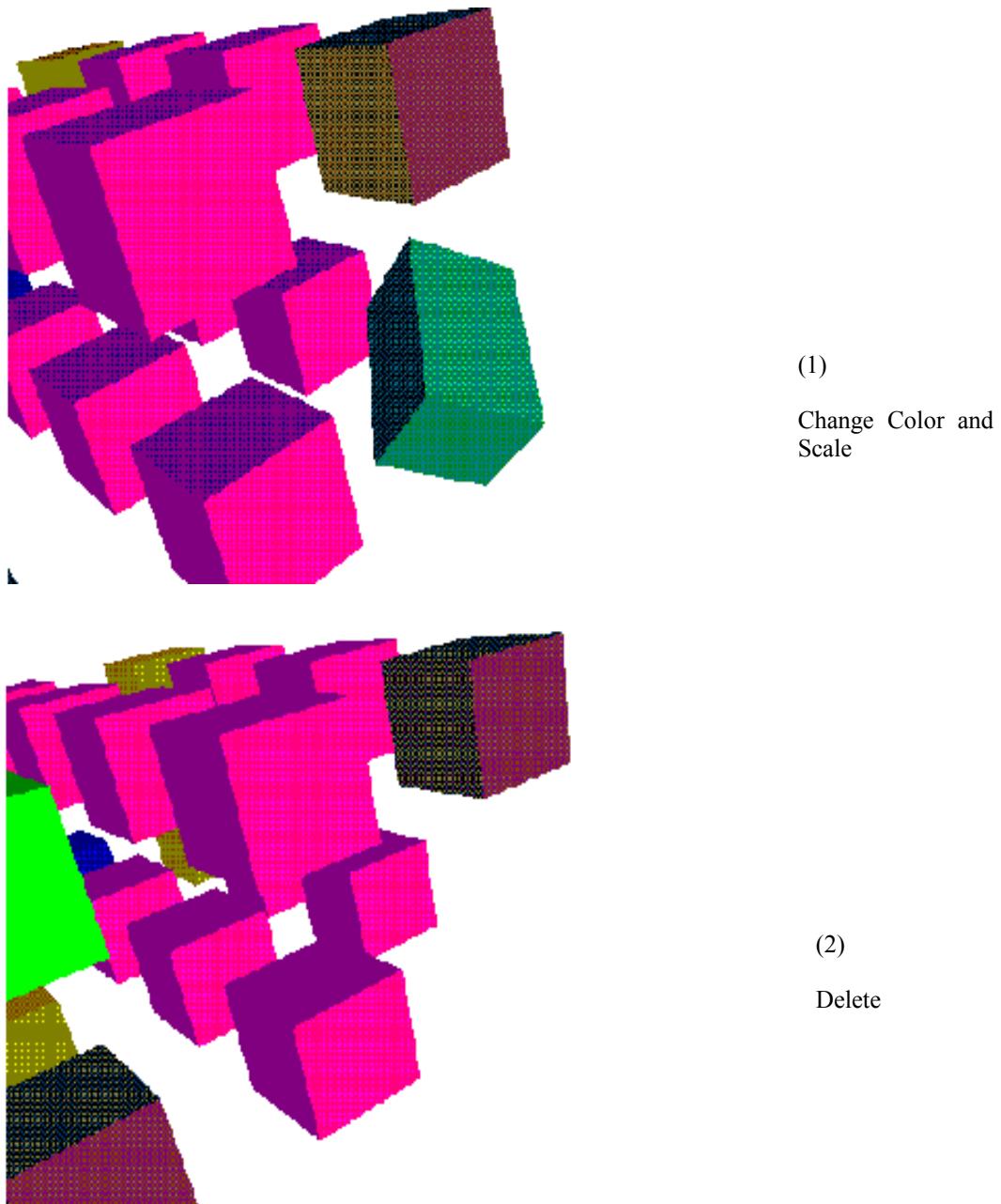


Figure 379: *Edit a node in the scene*

Delete Layer

Remove layer 1 in Figure 378(1)

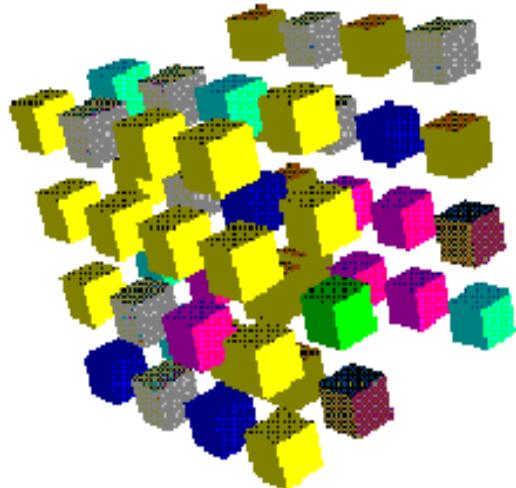


Figure 16

Scale Nodes

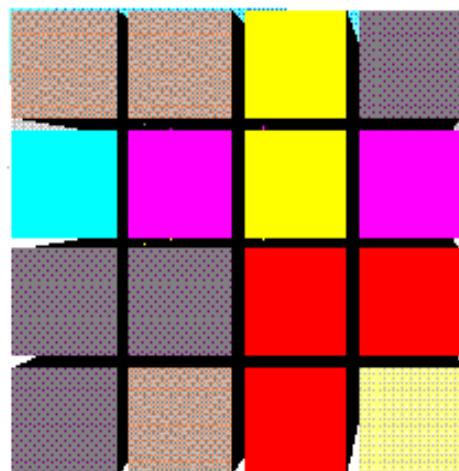
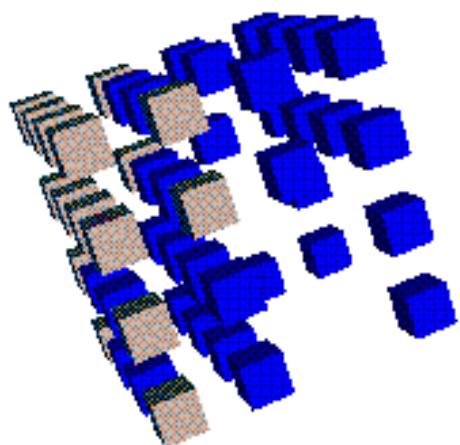


Figure 17

Transparent Display

Colors & Intervals:

	22.0	22.4
	22.4	22.79999
	22.8	23.2
	23.2	23.59999
	23.6	24.0
	24.0	24.4
	24.4	24.79999
	24.8	25.2
	25.2	25.59999
	25.6	26.0



Time: 00:00:05:000

Figure 18

Figure 19