CD++Modeler: a graphical toolkit to develop DEVS models

Chiril Chidisiuc Gabriel A. Wainer
Carleton University
1125 Colonel By. Ottawa, ON. K1S 5B6. Canada.
1-613-520-2600
gwainer@sce.carleton.ca

ABSTRACT

Modeling and simulation tools have been used for helping in the early stages of hardware/software systems design. The DEVS formalism is a technique that enables hierarchical description of discrete event models that can be used for this task. CD++Modeler is an application that permits defining DEVS models graphically. We present the design and main features of the application.

1. INTRODUCTION

In recent years, Real-Time embedded applications have grown tremendously, both in the number of existing systems and in the complexity of the tasks they execute. Current advances in computing technology have made it possible to automate tasks at a level of complexity unknown previously. The continuing trend in growth and increased complexity complicates the development of Real-Time software, where concerns for functionality, predictability and reliability must be addressed. Current methods for Real-Time software construction are either hard to scale up for large systems, or require a difficult testing effort with no guarantee for a bug free software product. Modeling and Simulation (M&S) techniques and tools for analyzing testing scenarios have proven to be able to provide product s that are of better quality and have a reduced lifecycle cost, due to improved testability and maintainability. Despite the net gains, most project managers are reluctant to use the techniques because they require extra initial resources in the construction of simulation models that will not be part of the delivered application. Similarly, in the early stage of the design of embedded systems, software and hardware are designed independently. The software development team is waiting for the hardware prototypes to be available; however, the hardware development team is waiting for the software environment for hardware prototype verification and testing. An M&S-based design approach allows the user to test the functionality of the hardware in a very early stage. This is economically efficient, and shortens the product development cycle and time-to-market period.

We have built an Eclipse-based platform called *CD++Builder* to enable the development of real-time applications using a model-based approach. The toolkit is based on a mathematical theory called DEVS (Discrete Event System Specifications), an increasingly accepted framework for understanding and supporting the activities of M&S [1]. The models, created in CD++Builder, can be programmed in C++ language and incorporated in one of the modules. Alternatively, models can be created graphically, using *CD++Modeler* – a stand-alone application, included in CD++Builder toolkit. DEVS theory provides an abstract, yet quite intuitive way of modeling, which is independent of any underlying runtime system, hardware, and middleware. CD++ [2] is an engine that enables users to define and execute DEVS models. *CD++Modeler* provides an alternative method to create DEVS models without the need to use programming languages. In this paper we show how the *CD++Modeler* was designed, and its relationship to CD++Builder toolkit.

2. BACKGROUND

The DEVS formalism for modeling and simulation [1] provides a framework for the construction of hierarchical models in a modular fashion, allowing model reuse, reducing development and testing time. The hierarchical and discrete event nature of DEVS makes it a good choice to achieve an efficient product development test. DEVS are timed models, which also enables us to define timing properties for the models under development. Each DEVS model can be built as a behavioral (atomic) or a structural (coupled) model. A DEVS atomic model is described as:

$$\mathbf{M = <X, S, Y, \delta_{int}, \delta_{ext}, \lambda, D>}$$

In the absence of external events, the model will remain in state $s \in S$ during $ta(s)$. Transitions that occur due to the expiration of $ta(s)$ are called internal transitions. When an internal transition takes place, the system outputs the value $\lambda(s) \in Y$, and changes to the state defined by $\delta_{int}(s)$. Upon reception of an external event, $\delta_{ext}(s, e, x)$ is activated using the input value $x \in X$, the current state s and the time elapsed since the last transition e. Coupled models are defined as:

$$\mathbf{CM = < X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, select >}$$

They consist of a set of basic models (*Mi*, atomic or coupled) connected through their interfaces. Component identifications are stored into an index (*D*). A translation function (*Zij*) is defined by using an index of influencees created for each model (*Ii*). The function defines which outputs of model *Mi* are connected to inputs in model *Mj*.

CD++ [2] implements DEVS theory. The toolkit has been built as a set of independent software pieces, each of them independent of the operating environment chosen. The defined models are built as a class hierarchy, and each of

them is related with a simulation entity that is activated whenever the model needs to be executed. New models can be incorporated into this class hierarchy by writing DEVS models in C++, overloading the basic methods representing DEVS specifications: external transitions, internal transitions and output functions. CD++ employs a virtual time simulation approach, which allows skipping periods of inactivity. A real-time engine enables simulation advancing based on the wall-clock.

3. CD++MODELER

CD++Modeler is part of the CD++ toolkit, created for designing and executing DEVS models using a graphical notation. The main components of the model and their relationships are shown in Figure 1. *CD++Modeler* provides the tools to animate the results of model simulation, and the user can create advanced models using a built-in specification tool.

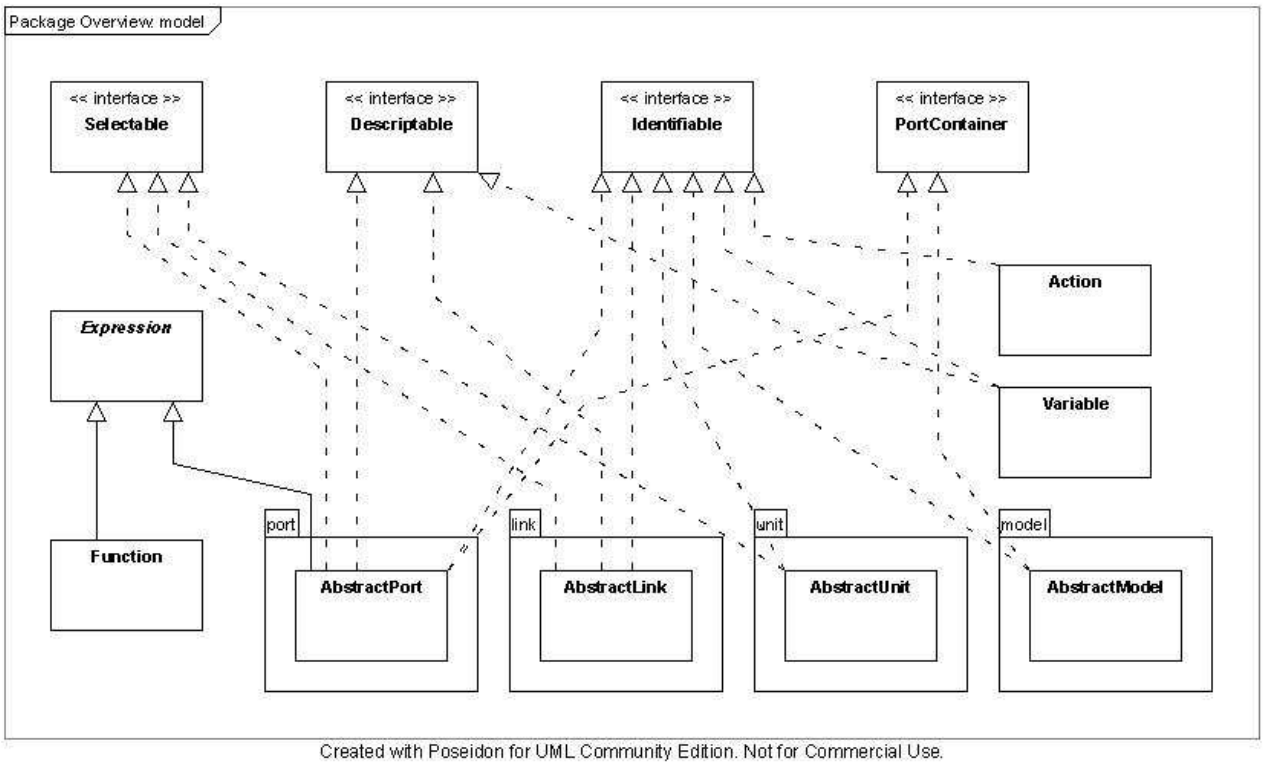


Figure 1. Design of CD++Modeler.

4. CD++MODELER DESIGN OVERVIEW

The Graphical User Interface (GUI) of *CD++Modeler* is shown in Figure 2. Some basic functions include the creation of atomic and coupled models, retrieving the parent class of the coupled model, run external commands, simulate the model and animate the simulation results. The application also includes a simple text editor.

When *CD++Modeler* is launched, method `main()` of `MainDEVS` class is called, to create an instance of `MainDEVS` type. The `MainDEVS` inherited layout and behavior of class `JApplet` of javax.swing package. Extending the `JApplet`, allowed the `MainDEVS` class to support the JFC/Swing component architecture [3].

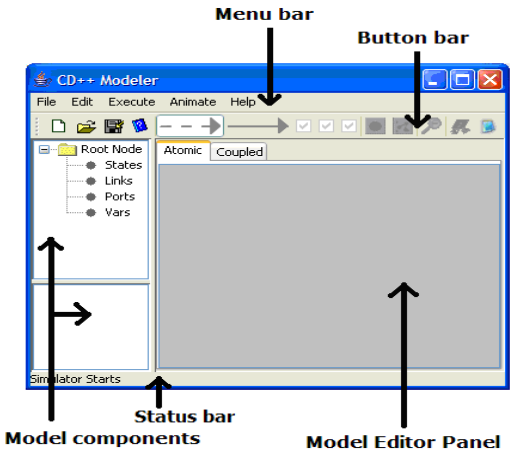


Figure 2. CD++Modeler window

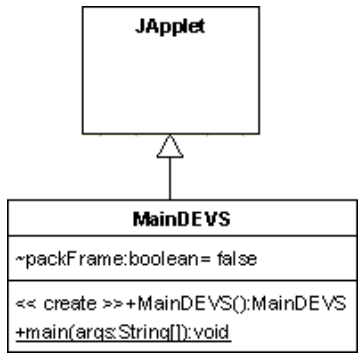
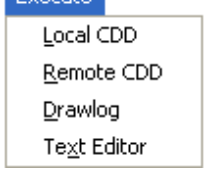


Figure 3. MainDEVS class diagram

`MainDEVS` instantiates the `MainFrame` class, which is responsible for *CD++Modeler* frame GUI. `MainFrame` class extends the `JFrame` class. The role of `MainFrame` is to collect and assemble the components, which later appear in the *CD++Modeler* window. Some of the main components include menu bar, button bar, model editor panel, model components panel, and the status bar. The menu bar appears at the top of the *CD++Modeler* window. The menus, menu items, and menu bar itself are created in `MainFrame` class using objects of `JMenu` and `JMEnuItem` type of the javax.swing package. When a menu item is selected, the corresponding action is performed. To make connection between a menu item and an action, each menu item is associated with a class that implements `ActionListener` interface. The relationships between menu, menu items, and classes responsible for menu item actions are shown in Table 1.

Menu	Menu Item	Action class	Description of action
File 	New	`NewActionListener`	Creates new project (atomic or coupled,)
	Open	`OpenActionListener`	Opens project
	Save	`saveActionListener`	Saves project
	Save As	`saveAsActionListener`	Saves project with user-specified name
	Close	`CloseActionListener`	Closes current project and prompts to save, if needed
	Export	`FileExportActionListener`	Exports model with new extension
	Save and Export	`FileExportActionListener` `saveActionListener`	Performs two operations: 1. Save (see save) 2. Export (see export)
	Import	`FileImportActionListener`	(works only for coupled models)
	Image Repository Directory	`ImageRepositoryActionListener`	Sets image source
	Exit	`actionListener`	Closes the project and terminates CD++Modeler
Edit 	Undo	N/A	
	Redo	N/A	
	Copy	N/A	
	Paste	N/A	
	Delete	`toolbarDeleteActionListener`	Removes selected object in model editor panel
Execute 	Local CDD	`LocalCDDActionEventHandler`	Simulate coupled model on local machine
	Remote CDD	`RemoteCDDActionEventHandler`	Simulate coupled model on remote machine
	Drawlog	`DrawlogActionEventHandler`	draws the evolution of a cellular model
	Text Editor	`textEditorActionListen`	Allows user to edit models

		`er`	in text mode
Animate Animate Cell-DEVS animation AtomicAnimate CoupledAnimate	Cell-DEVS animation	`cellAnimateActionListener`	visualize the simulation results of atomic Cell-DEVS models
	AtomicAnimate	`atomicAnimateActionListener`	visualize the simulation results of atomic-DEVS models
	CoupledAnimate	`coupledAnimateActionListener`	visualize the simulation results of coupled-DEVS model

Table 1. CD++Modeler menu structure

The menu "Edit", of *CD++Modeler* toolkit, provides basic editing tools, which include: undo and redo the changes, copy, paste, and delete elements of the model. Menu "Execute" provides tools to simulate the model, draw simulation results and edit files, associated with the model, in text form. To establish the connection between buttons and objects of the text editor dialog window with their corresponding actions, external classes, which implement ActionListener interface, were created.

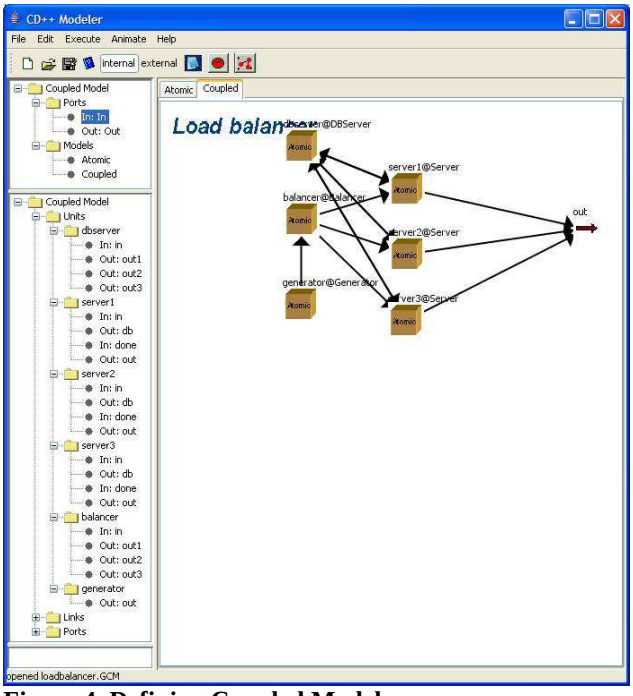


Figure 4. Defining Coupled Models.

The "Animate" menu allows client to animate the results of model simulation. However, unlike the Drawlog, all types of models (Cell-DEVS, Atomic, and Coupled) can be processed for animation. When model is simulated, the results are recorded to a file with "log" extension. The information stored in the "log" file is used to animate the results.

To select the type of model for animation, the client must choose one of the following menu items of the "Animate" menu: "Cell-DEVS animation" (cell-DEVS model), "AtomicAnimate" (atomic model), or "CoupledAnimate" (coupled model). Upon selection of one of the three menu items, a corresponding dialog window appears on the screen.

In order to link the menu items with their corresponding actions, inner classes `cellAnimateActionListener`, `atomicAnimateActionListener`, and `coupledAnimateActionListener` were created within `MainFrame` class. All three inner classes implemented `ActionListener` interface. To design the GUI of Cell-DEVS animation, layout and behavior of `CellAnimateIf` interface was used. The concrete implementation of `CellAnimateIf` methods was done in `CellAnimate` class. The `CellAnimate` class, which extended class JFrame class of javax.swing package, additionally was able to define the GUI for a pop-up window that appears, when Cell-DEVS animation is in use.

Using similar design of Cell-DEVS animation mechanism, the animation feature for atomic and coupled models was created. In summary, the `MainFrame`'s inner classes, `atomicAnimateActionListener` and `coupledAnimateActionListener`, implemented the `ActionListener` interface and defined the behavior of *CD++Modeler* in case of selecting `AtomicAnimate` or `CoupledAnimate` menu items respectively. Once any of the two inner classes are instantiated, their `actionPerformed()` methods are called. Unlike the Cell-DEVS animation, a dialog window, prompting the client to input details about the model, appears on the screen in case of atomic and coupled models. The window, containing tools to animate either atomic or coupled model, follows the dialog window after the details about the model were provided.

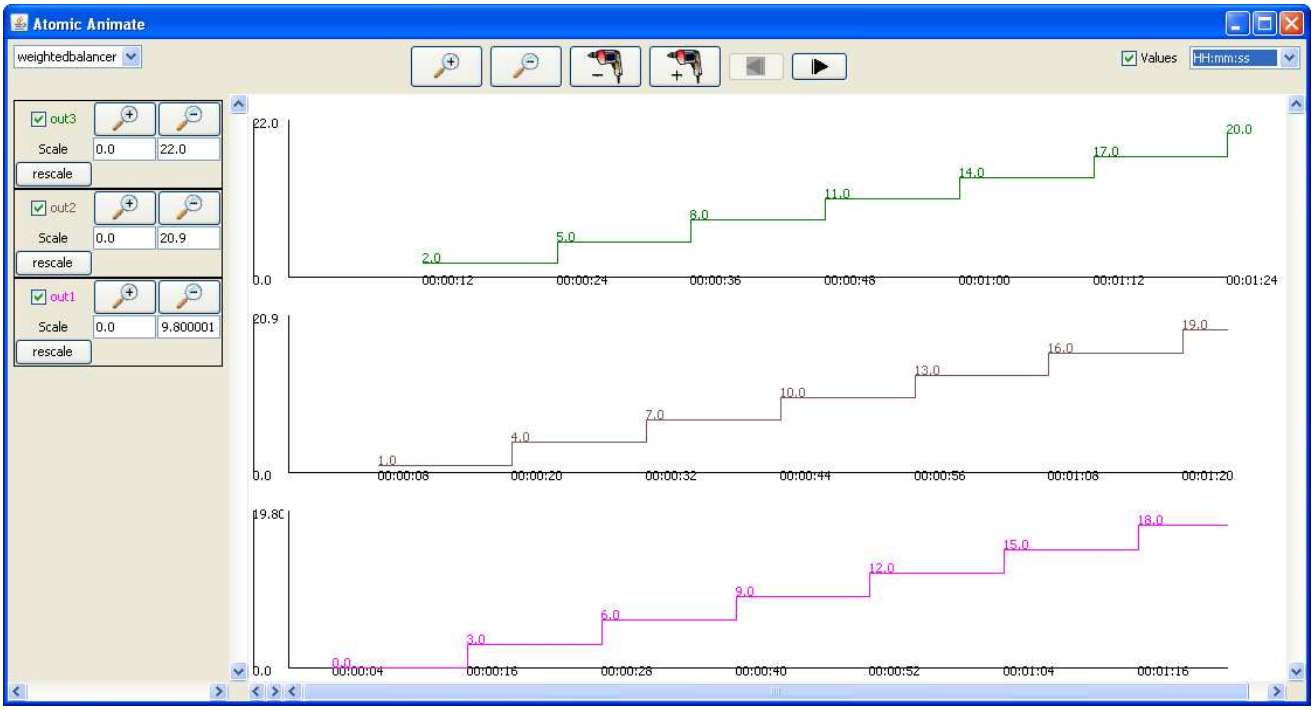


Figure 5. Classes that implement ActionListener interface for components of TextEditFrame class

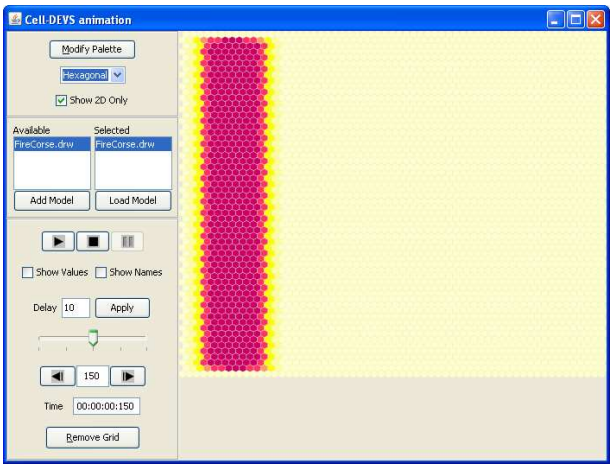


Figure 6. Cell-DEVS animate GUI

Class `AtomicAnimateDialog` defines the GUI of the atomic dialog box, whereas the GUI for the coupled dialog box was constructed in class `CoupledAnimateDialog`, both of which inherited the behavior of class `OkCancelJDialog`.

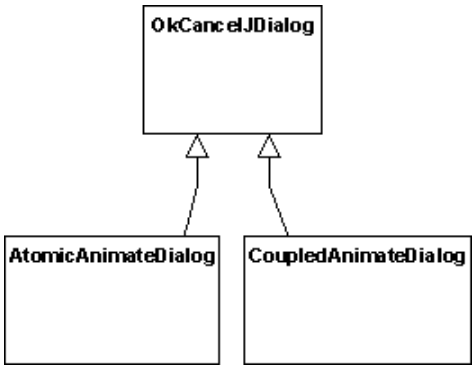


Figure 7. Atomic and Coupled animate dialog window relationship

Once model details were entered at the prompt of the dialog box, the animation window (specific to atomic and coupled models) appears on the screen. Class `AtomicAnimate`, responsible for the GUI of atomic animate window, or class `CoupledAnimate`, responsible for GUI of coupled animate window, is instantiated, depending if the atomic or coupled model were selected. Classes `CoupledAnimate` and `AtomicAnimate` provide the concrete implementation of `CoupledAnimateIf` and `AtomicAnimateIf` interfaces respectively.

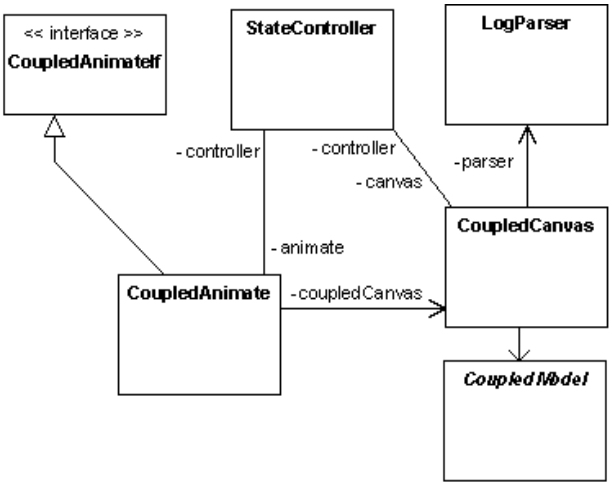


Figure 8. Coupled Animate class diagram

To increase the efficiency and simplify usage of *CD++Modeler*, the tools that are accessed the most often were placed in the button bar, located in the main window of *CD++Modeler*.

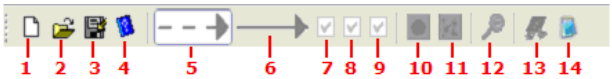


Figure 9. CD++Modeler button bar (see Table 2 for further details)

To create the bar, containing tools and buttons of *CD++Modeler*, `JToolBar` class of javax.swing package was used due to its ability to present the tools, actions, and controls in a user-friendly manner [4]. Instance of `JToolBar` was added to *CD++Modeler* window using `BorderLayout` layout manager. To perform actions, each button of the toolbar is associated with a class that implements the `ActionListener` interface. The summary of classes that provide the `ActionListener` interface to the components of *CD++Modeler* toolbar is given in Table 2. Action classes, listed in Table 2, are inner classes of `MainFrame` class, except for the `getAtomicGraphEditor()` method, which returns an instance of the external class, called `AtomicModelEditor` class.

Index	Emblem	Button	Action class	Description
1		*New*	`NewActionListener`	new project
2		*Open File*	`OpenActionListener`	open project
3		*Save File*	`saveActionListener`	save current activity
4		*Help*	`HelpActionListener`	opens the help dialog
5		*Internal Link*	`getAtomicGraphEditor(): AtomicModelEditor`	Places internal link between units of a model
6		*External Link*	`getAtomicGraphEditor(): AtomicModelEditor`	Places external link between units of model
7		*Show Link Expression*	`getAtomicGraphEditor(): AtomicModelEditor`	Shows expression in text form beside all links (internal external), when checked
8		*Show Link Actions*	`getAtomicGraphEditor(): AtomicModelEditor`	Shows action in text form beside all links (internal external), when checked
9		*Show Link Ports*	`getCoupledModelEditor() :CoupledModelEditor`	Shows ports in text form beside all links (internal external), when checked
10		*Add new Atomic Model Unit*	`AddAtomicUnitActionListener`	Adds atomic unit to a coupled model
11		*Add new Coupled Model Unit*	`AddCoupledUnitActionListener`	Adds a coupled unit to a coupled model
12		*Close Exploded Unit*	`ExitActionListener`	Close exploded unit (only for coupled model)
13		*Local Simulator*	`LocalSimulatorActionListener`	simulates the model (only coupled model)
14		*Editor*	`EditorActionListener`	provides a simple text editor application

Table 2. CD++Modeler toolbar

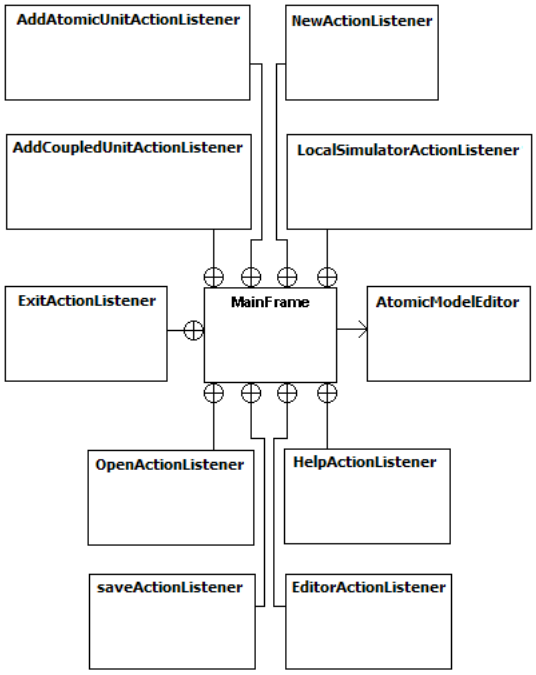


Figure 10. Action classes of CD++Modeler toolbar.

Five components, included in the toolbar, share the same functionalities as some menu items, to provide an alternative way to access widely used features of *CD++Modeler*: buttons "New", "Open File", "Save File", "Local Simulator", and "Editor" with menu items "New", "Open", "Save", "Local CDD", and "Text Editor" respectively. Buttons, listed above, perform the same actions as their menu counterparts. To explore further about design details of actions, performed by these buttons, see the design description of the corresponding menu item in section 4.2 of this document. Buttons, that do not have a menu counterpart are: "Help", "Internal Link", "External Link", "Show Link Expression", "Show Link Actions", "Show Link Ports", "Add new Atomic Model Unit", "Add new Coupled Model Unit", and "Close Exploded Unit" (refer to Table 2).

`MainFrame`'s inner class `HelpActionListener` dictates the response of "Help" button. Method `actionPerformed()` of `HelpActionListener` invokes `showHelp()` method of `MainFrame` class, which instantiates `HelpInfo` and `HelpLoader` classes. Class `HelpLoader` creates the GUI's help window. To create the default layout for "Help" feature of *CD++Modeler*, class `HelpLoader` used `HelpSet` class from javax.help package. `HelpSet` object represents a collection of information about content, structure, and layout of the "Help" content: `HelpSet` file, table of contents (TOC), index, topic files, and Map file [5]. To represent the `HelpSet` object visually, a `HelpBroker` object was used.

To create a pop-up window that offers client to open the *CD++Modeler* manual, class `HelpInfo` extended `JFrame` class of javax.swing package. Manual can be opened in either format: "doc", "html", or both. Each format of the manual is opened in native system application, responsible for that format (e.g. "*.html" opens in internet browser), which was made possible by running Windows command line "`cmd /c start + [file_path]`".

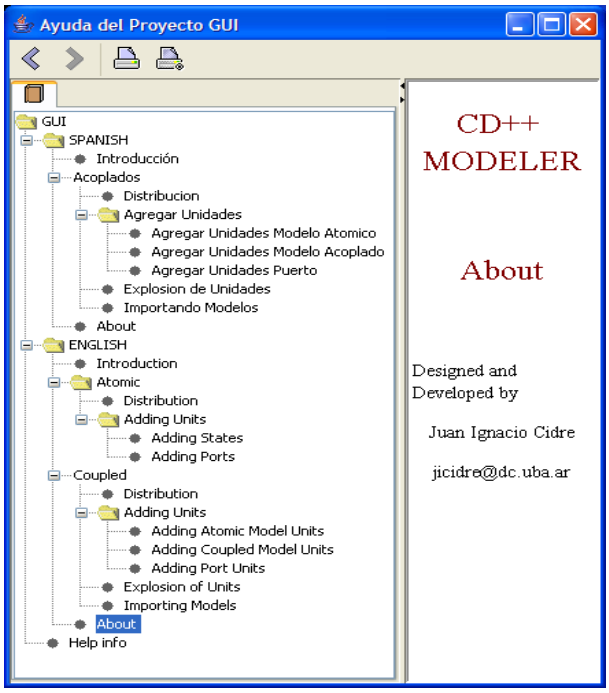


Figure 11. CD++Modeler Help window

Button "Internal Link" (see Table 2) is used to edit DEVS models in *CD++Modeler*: it allows client to link components of an atomic DEVS model by connecting their internal ports. Class `AtomicModelEditor` provides the `ActionListener` interface to "Internal Link" button. The attributes of the internal link are used, when drawn in the model editor panel of the *CD++Modeler*. To link external ports, the "External Link" button is used (see Table 2). The button uses `AtomicModelEditor` class, which provides the `ActionListener` interface. To create the GUI of the "Internal Link" and "External Link" buttons, `JToggleButton` class of javax.swing package was used. The `JToggleButton` class provides the behavior of a two-state button and is a superclass to `JRadioButton` and `JCheckBox` classes. Since only one type of link can be created at a time in *CD++Modeler*, the two-state button, provided by `JToggleButton` class, was used.

A set of properties is associated with each link in *CD++Modeler*. To show or hide the properties in the Model Editor Panel of *CD++Modeler* main window, checkboxes "Show Link Expression", "Show Link

Actions", and "Show Link Ports" were placed in the toolbar near the "Internal Link" and "External Link" buttons. Checkboxes "Show Link Expression" and "Show Link Actions" are attributes of Atomic model only, therefore, it was designed such that AtomicModelEditor provided implementation of ActionListener interface for these two checkboxes. Checkbox "Show Link Ports" appears in coupled models only, thus, making it reasonable to make CoupledModelEditor implement ActionListener interface for this checkbox. To get default layout and properties, all three checkboxes extended "JCheckBox" class of javax.swing package and were instantiated in `MainFrame` class.

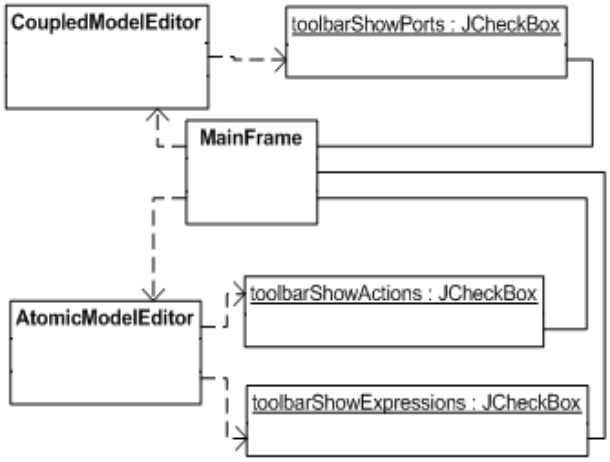


Figure 12. Composite design diagram of toolbar checkboxes

Since coupled model by definition is a system of atomic and coupled DEVS models, it requires more editing than atomic model. To improve the efficiency of editing a coupled model, buttons "Add new Atomic Model Unit" and "Add new Coupled Model Unit" were added to *CD++Modeler* toolbar, as described in Table 2. The buttons are enabled only when tab "Coupled" of Model Editor Window is selected. Both buttons were created using default layout and behavior of java library class `JButton` (javax.swing package) and are instantiated in `MainFrame` class. Inner classes `AddAtomicUnitActionListener` and `AddCoupledUnitActionListener` provide implementation of `ActionListener` interface for "Add new Atomic Model Unit" and "Add new Coupled Model Unit" buttons respectively. Class `CoupledModelEditor` adds both atomic and coupled units to the model.

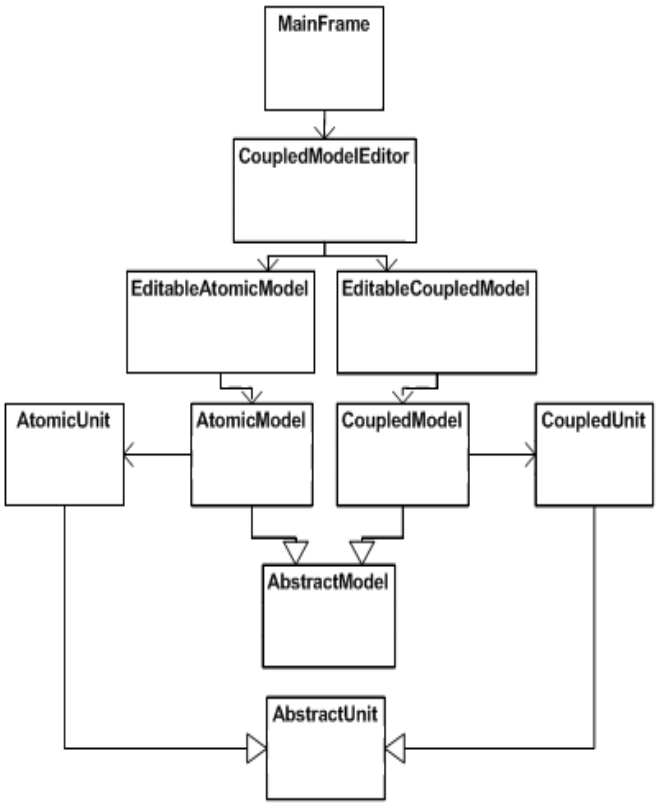


Figure 13. Adding atomic/coupled unit to coupled model

To edit units of a coupled model, the units can be opened ("exploded") in a modified *CD++Modeler* window, where only either atomic or coupled models can be edited, i.e. Model Editor Window contains either atomic or coupled tab, depending on the type of model that was exploded.

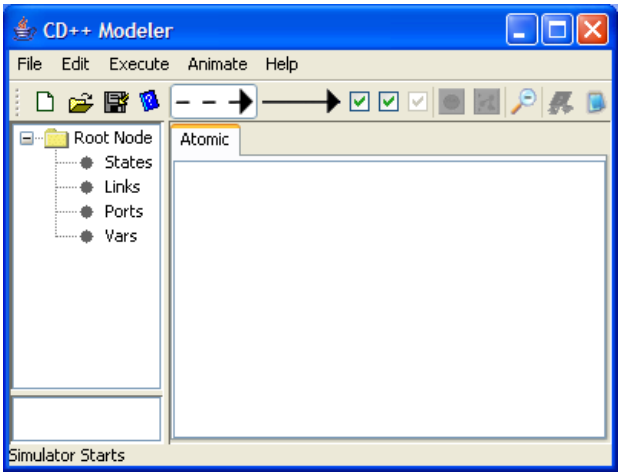


Figure 14. Exploded atomic unit

To return to the original *CD++Modeler*, once the "exploded" unit was edited, button "*Close Exploded Unit*" is used. This button was created in `MainFrame` class and used class `JButton` of javax.swing package to define its

default layout and behavior. Inner class `ExitActionListener` of `MainFrame` class implements `ActionListener` interface for "*Close Exploded Unit*" button. The response is identical to the response of menu item "Exit" of "File" menu (see Section 4.2.1).

The workspace of *CD++Modeler* was designed using split pane layout. The split pane is a pane, which can be split into sections either vertically or horizontally. To create a split pane, class `JSplitPane` of javax.swing package was used. The Split Pane was then partitioned horizontally into two major sections (Section 1 and Section 2) that were placed horizontally one beside another using a vertical divider. The split pane was inserted in *CD++Modeler* frame using java layout manager `BorderLayout` of java.awt package.

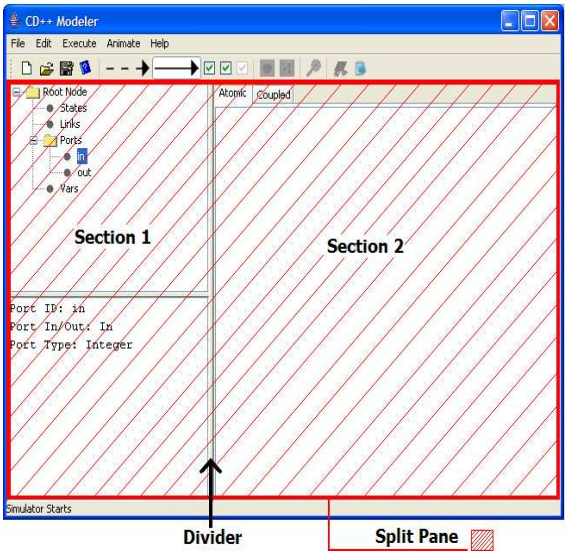


Figure 15. Split pane - component of CD++Modeler

The Model Editor Panel is a component of the *CD++Modeler* split pane (labeled as "Section 2" in the figure). To create GUI for Model Editor Panel, instance of `JTabbedPane` class (javax.swing package) was added to right-most section of *CD++Modeler* split pane. The class `JTabbedPane` provided tabbed layout of the Model Editor panel. Upon instantiation of the `JTabbedPane` object, two tabs were added "Atomic" and "Coupled". Tab "Atomic" was designed to provide a graphical model editor for atomic DEVS model, whereas "Coupled" provided graphical model editor for coupled DEVS model. Both "Atomic" and "Coupled" tabs placed an instance of `JScrollPane` (javax.swing package) in its content to provide the area for a graphical model editor. The actual graphical editors were provided by `AtomicModelEditor` and `CoupledModelEditor` classes. The atomic graphical model editor was placed in scroll pane of "Atomic" tab, while graphical editor for coupled models was placed in scroll pane of "Coupled" tab.

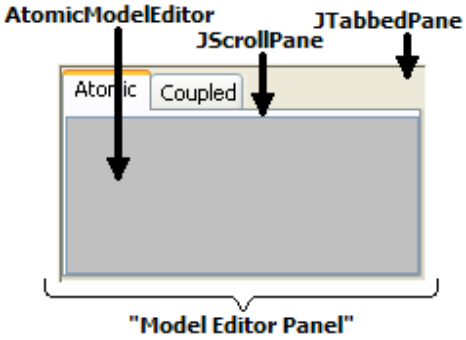


Figure 16. Class components of Model Editor Panel

Upon selection of one of the tabs, the other tab becomes invisible, which was made possible by using `ActionListener` interface implemented by inner class `TabbedPaneChangedListener` of `MainFrame` class.

To start editing a model, the user must select the appropriate tab and choose menu item "New" of File menu or button "New" in the toolbar. As the result, an editable model is instantiated and is associated with model editor of the selected tab. For example, when tab "Atomic" is selected, class `EditableAtomicEditor` is instantiated and associated with `AtomicModelEditor`. The relationship of classes that contribute to model editor design is shown following.

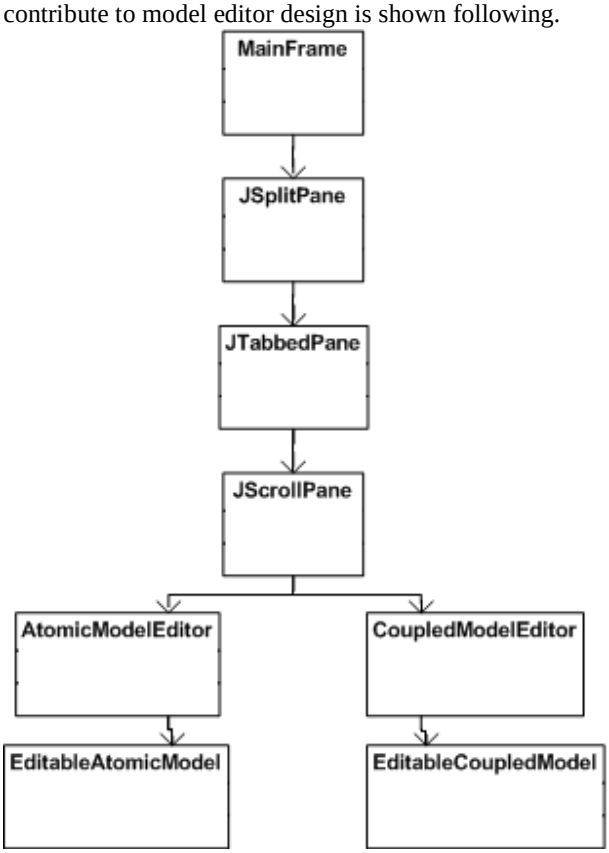


Figure 17, Design of model editor panel

CD++Modeler provided Model Components Panel to view, access, and use components of a model (labeled as "Section 1" in the figure). The components panel, which was created using `JPanel` class of javax.swing package, is located in left-most partition of *CD++Modeler* Split pane. The objects, included in the components panel are categorized by their functionality. To categorize objects, the components panel was split into sections, using default layout and functionalities of `JSplitPane` class of javax.swing package. Instance of class `JScrollPane` was added to each partition of the pane. The contents of the Components Panel were added to the scroll panes directly.

The GUI of components panel is different for Atomic and Coupled models, since contents of the two differ.

Upon starting the *CD++Modeler*, by default, tab "Atomic" of model editor panel is selected and its corresponding components panel is shown. To create GUI for components panel of atomic model, a split pane, divided vertically in two blocks (top and bottom), was added to the panel. Instance of `JScrollPane` class was added to each block of the split pane. Instance of `AtomicUnitsTree` class was added to the scroll pane of the top partition of split pane and instance of `DescriptableDataPanel` class was added to the scroll pane of bottom portion of split pane. As the result, panel, shown in the following figure, was created.

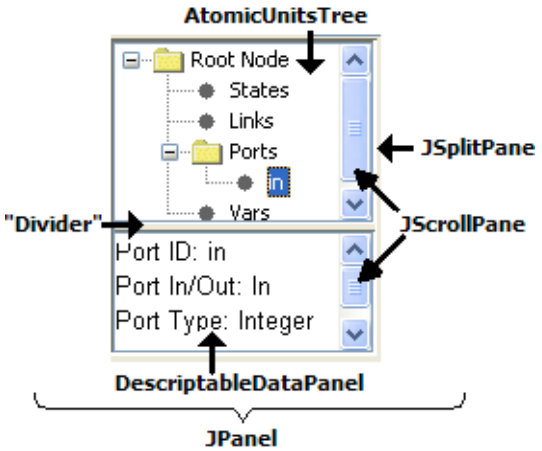


Figure 18. Component panel design

The relationship of classes that contributed to the design of atomic components panel is shown following. To create components panel for coupled model, different GUI is created. Similarly to atomic components panel, instance of `JPanel` class of javax.swing package was inserted in left-most partition of *CD++Modeler* split pane (labeled as "Section 1" in the figure). To create the GUI, an instance of vertically split `JSplitPane`, with another instance of `JSplitPane` in its top partition, was added to the panel.

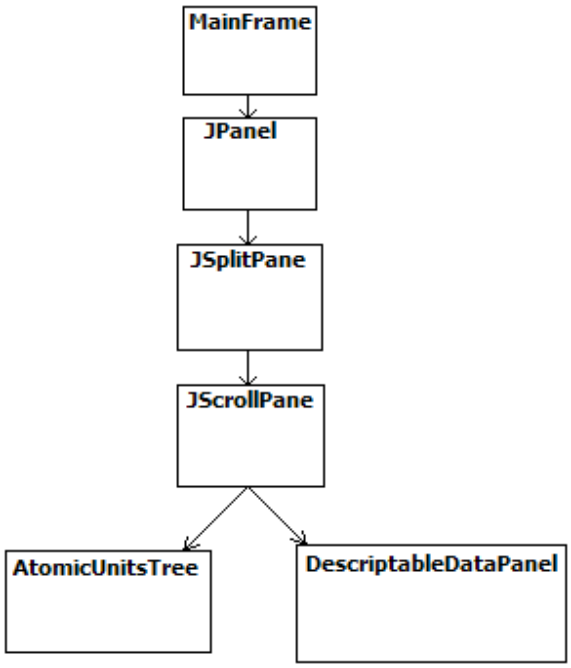


Figure 19. Classes incorporated in components panel

Such layout provided three sections of the panel, available for further development. An instance of `JScrollPane` class was added to each of the partitions. Class `CoupledClassesTree` provided view for scroll pane in the top section, `CoupledUnitsTree` in the middle, and `DescriptableDataPanel` in the bottom section respectively.

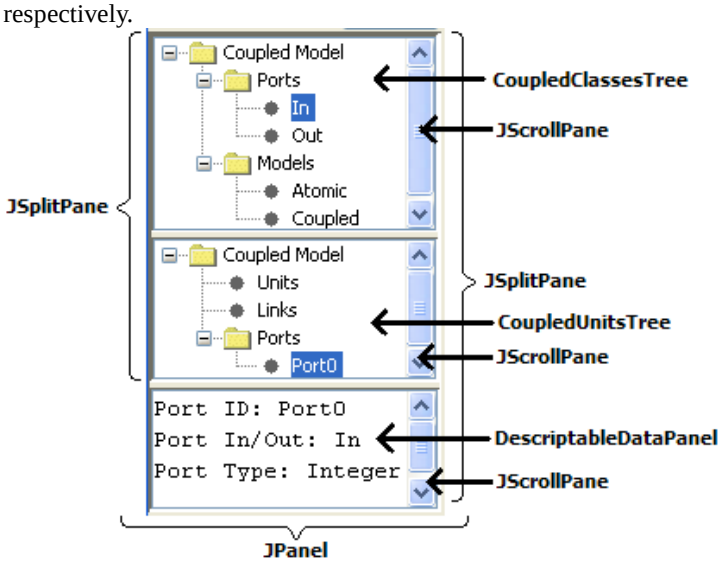


Figure 20. Coupled model components panel structure

The Status bar was designed to display description of current activities within *CD++Modeler* environment. To create status bar for *CD++Modeler*, an instance of `JLabel` class was created in class `MainFrame`. Using `BorderLayout` layout manger, the label was added to the south region of *CD++Modeler* frame, defined by class `MainFrame`. To display messages in the status bar,

method `setText()` of `JLabel` class was used to set the text of label.

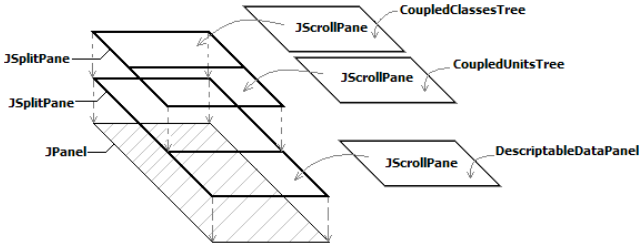


Figure 21. Design of coupled components panel

5. CONCLUSION

We presented the design of *CD++Modeler*, where DEVS models can be graphically built and edited. The *CD++Modeler* combines CD++Builder functionalities to create and simulate the models with unique features of its own, such as animation of the model simulation results. The CD++Builder plug-in requires installation of Eclipse platform, Cygwin, and Java Runtime Environment (JRE) 5.0 in order to properly operate, whereas *CD++Modeler* requires only JRE 5.0 to be available on the machine and can be run as a stand-alone application. Therefore, *CD++Modeler* provides a versatile and simple way to create, edit, and simulate DEVS models, as well as animate simulation results. The graphical nature of *CD++Modeler* environment permits users with various levels of experience in programming to develop DEVS models. In the future, *CD++Modeler* will be further developed in the area of finishing the existing and introducing new components, hence, increasing the efficiency of the application.

REFERENCES

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

[2] "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

[3] Sun Microsystems, Inc., "Class JApplet", [Online document] 2004, [2006 Sep. 15], Available at HTTP: http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JApplet.html

[4] Sun Microsystems, Inc., "Class JToolBar", [Online document] 2004, [2006 Sep. 17], Available at HTTP: http://java.sun.com/j2se/1.5.0/docs/api/javax/swing/JToolBar.html

[5] Sun Microsystems, Inc., "Class JToolBar", [Online document] 2004, [2006 Sep. 17], Available at HTTP: http://supportweb.cs.bham.ac.uk/docs/java/stdex/javahelp/api/javax/help/HelpSet.html