

CD++Builder: An Eclipse-Based IDE For DEVS Modeling

Chiril Chidisiuc

Gabriel A. Wainer

Carleton University

Dept. of Systems and Computer Engineering
1125 Colonel By. Ottawa, ON. K1S 5B6. Canada.
1-613-520-2600

cchidisi@connect.carleton.ca

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++Builder is an Eclipse plugin that permits defining these applications. We present the main features of the toolkit and its relation to Eclipse libraries and extension points.

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 products 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. A M&S-based design approach allows the user to test the functionality of the hardware in a very early stage. This is economically effi-

cient, 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]. 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++Builder provides a runtime system associated with Eclipse plug-ins to support our model-centered methodology. In this paper we show how the toolkit was defined, and its relationship to Eclipse libraries and extension points.

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:

$$\mathbf{M} = \langle \mathbf{X}, \mathbf{S}, \mathbf{Y}, \mathbf{d}_{\text{int}}, \mathbf{d}_{\text{ext}}, \mathbf{l}, \mathbf{D} \rangle$$

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 $\mathbf{I}(s) \in Y$, and changes to the state defined by $\mathbf{d}_{\text{int}}(s)$. Upon reception of an external event, $\mathbf{d}_{\text{ext}}(s, e, x)$ 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:

$$CM = \langle X, Y, D, \{M_i\}, \{I_i\}, \{Z_{ij}\}, \text{select} \rangle$$

They consist of a set of basic models (M_i , atomic or coupled) connected through their interfaces. Component identifications are stored into an index (D). A translation function (Z_{ij}) is defined by using an index of influences created for each model (I_i). The function defines which outputs of model M_i are connected to inputs in model M_j .

CD++ [2][2] implements DEVS theory. The tool-kit 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++BUILDER

CD++Builder packages all the potential tools that can be used in a CD++ project from the existing CD++ toolbox, the Eclipse workbench, and its own tool set. A default perspective for CD++Builder is given to support the CD++ environment. CD++ Builder was designed and developed as a plug-in for the Eclipse platform. The latest version of the plug-in is compatible with Eclipse 3.1.x release. The final product is required to provide the following functionalities:

- Create and configure a project
- Edit model files, including reserved word and key word highlighting
- Display models and execution results
- Display buttons and perform operation with buttons

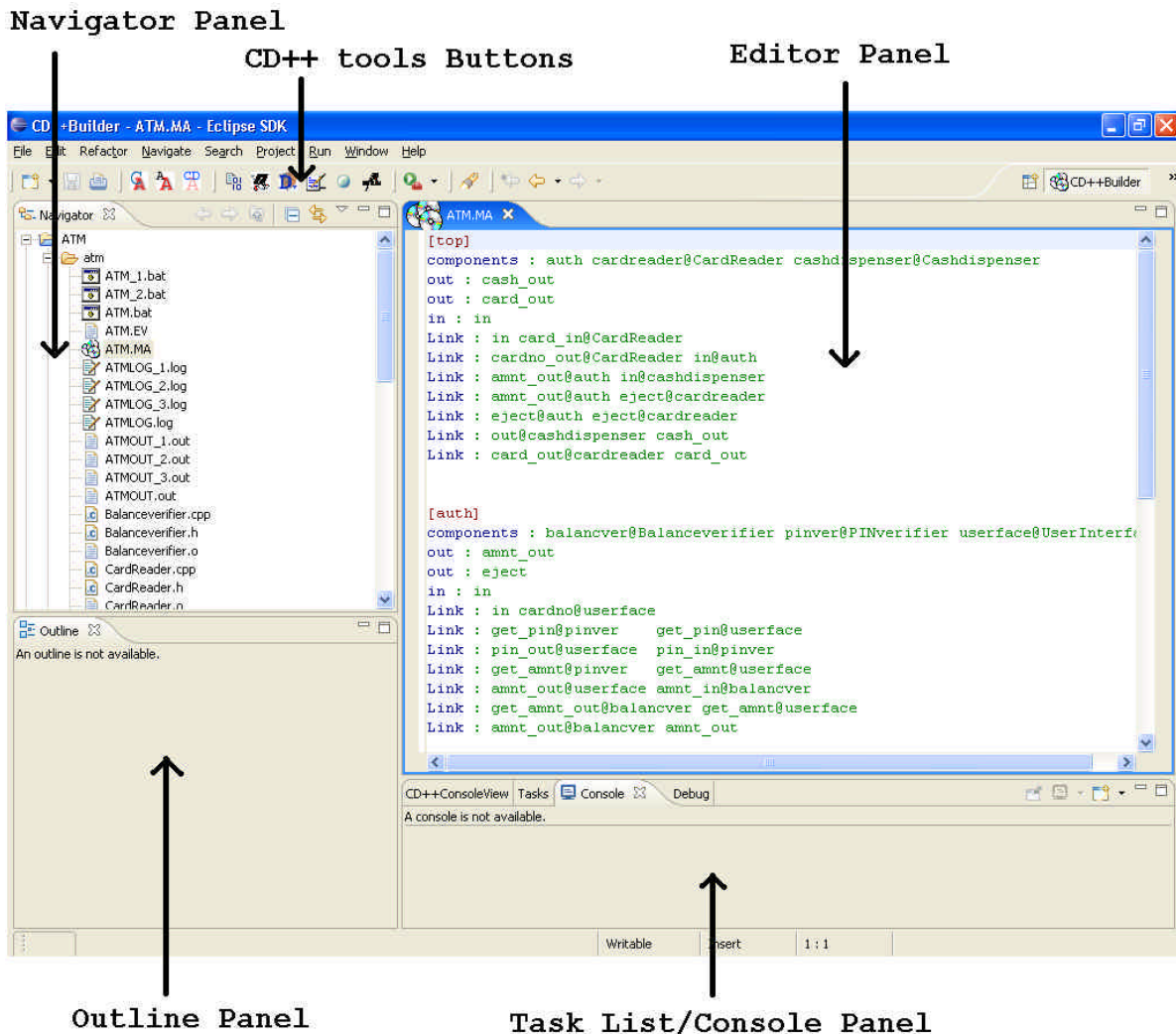


Figure 1. CD++Builder main window.

In accordance with the Eclipse 3.1 platform specifications, the components of CD++ Builder plug-in are defined in the manifest file, which contains declarations of dependencies on other plug-ins, visibility of plug-in public classes to other plug-ins, declaration of its functionalities, and implementation of other plug-ins' extension points [5]. The plug-in was designed by extending the Eclipse framework extension points to meet the above requirements. The extension points are used as entry points that allow plug-ins to offer new services or contribute to the behaviour of the existing components and services. The requirements of CD++ Builder plug-in were implemented by extending the interfaces provided through the following extension points: `org.eclipse.ui.ide.resourceFilters`, `org.eclipse.ui.editors`, `org.eclipse.ui.editors.documentProviders`, `org.eclipse.ui.views`, `org.eclipse.ui.newWizards`, `org.eclipse.ui.actionSets`, `org.eclipse.ui.popupMenus`, `org.eclipse.ui.perspectives` and `org.eclipse.ui.preferencePages`. The following figure shows the main components of CD++Builder main Window.

As we can see, there are some basic components:

- **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.

4. DESIGN OVERVIEW

The CD++ Builder plug-in is organized as a perspective, defined at the "org.eclipse.ui.perspectives" extension point. It also provides a project wizard (`org.eclipse.ui.newWizards`), a customized text editor, designed to work with model files (`org.eclipse.ui.editors` and `org.eclipse.ui.editors.documentProviders`), and a Console View Window to display the results of a running model as well as current system status (`org.eclipse.ui.views`).

The perspective implements the inherited methods from the `IPerspectiveFactory` interface to define the initial page layout and visible components (i.e. tabs, panels, and action sets), using the inherited method `createInitialLayout`. CD++Builder has a variety of

components to execute DEVS models and to analyze simulation results.

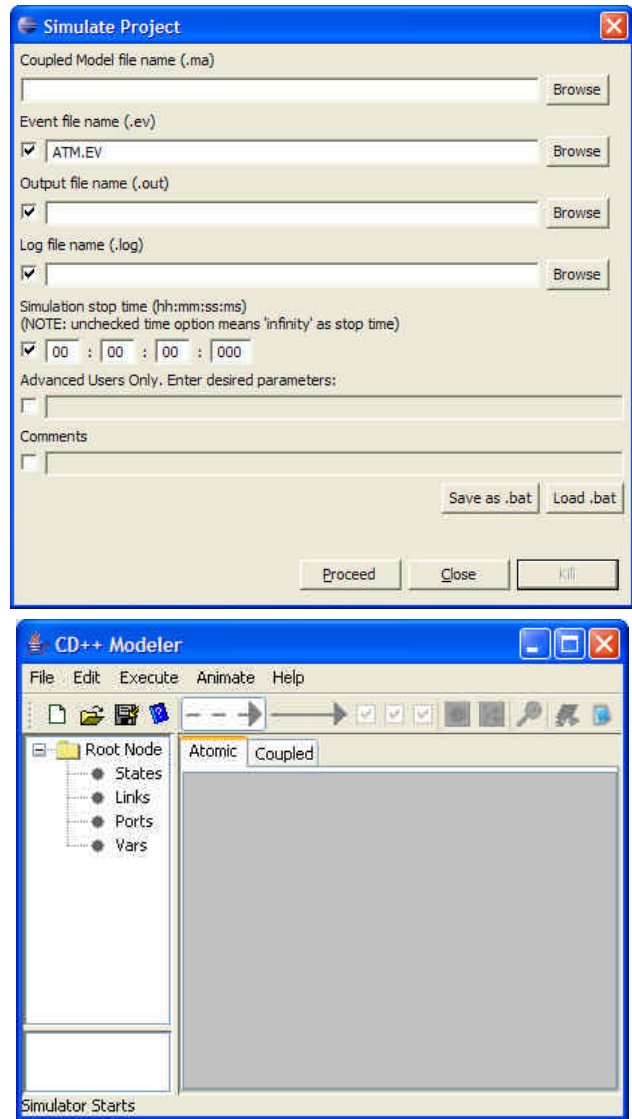


Figure 2. CD++ Builder button actions

The main components are integrated as buttons located in the top toolbar:

- **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 model.
- **Execute:** This button activates the CD++ engine. This executable represents a project-specific program that will execute a DEVS model.
- **Drawlog:** This button generates a file for easier visualization of the execution of a model.
- **CD++Modeler:** This button loads the CD++ Modeler program, a graphical tool for designing and executing

DEVS models. In this application you can design atomic/coupled models as well as animate the simulation results.

Each button has a corresponding class file, which defines the action for it. The association between a button and its action class was done using the **IworkbenchWindow ActionDelegate** interface. This interface was provided by extending the "org.eclipse.ui" extension point. Each action class implemented the inherited methods, one of which is the **run** method, inherited by **Iworkbench-**

Window ActionDelegate interface from **IAction-Delegate** interface. The only method implemented for each action class is **run**, thus making the response of each button unique, which is analogous to **actionPerformed** of Java **ActionListener** interface. Figure 2 shows the description of the action taken when buttons are pressed: **CDBuilder.buttons.SimuAction** launches a dialog window to run the models, and the **CDBuilder.buttons.BubbleEditorAction** starts **CD++Modeler**.

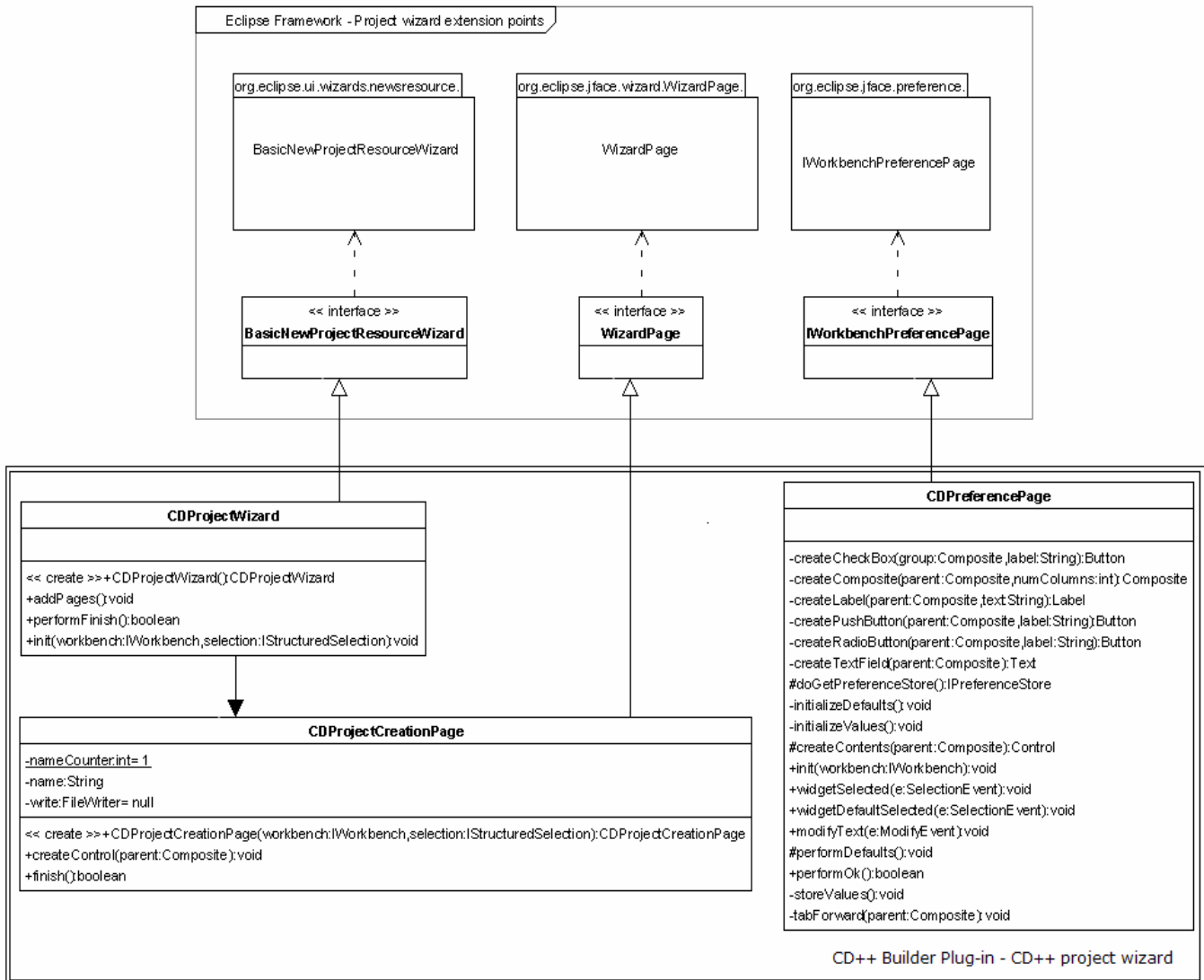


Figure 3. CD++ Builder project wizard class diagram

To create a model, CD++ Builder includes a project wizard. Its functionalities were implemented by extending the "org.eclipse.ui.newWizards" interfaces: **BasicNewProjectResourceWizard**, **WizardPage**, and **IWorkbenchPreferencePage**. The relationship between the interfaces and the classes that extend them is

shown in Figure 3. The **BasicNewProjectResourceWizard** interface is a workbench wizard that creates a new project resource in the workspace. The class **CDProjectWizard** implements the interface. A page was added to the interface-predefined basic structure of the wizard, where client's name must be entered, by us-

ing the `CDProjectCreationPage` class, which implemented the `WizardPage` interface. `CDProjectWizard` class, which inherited from interface `BasicNewProjectResourceWizard` includes the method `addPage`, added the page to the wizard. When the client finishes entering data into CD++ project wizard pages and clicks button *Finish*, the method `performFinish` of `CDProjectWizard` is called, which creates a project resource with user-entered name. After the model is, the next step is to define the components of the model. The components of the model are listed in the model file with extension. The `cseEditor` editor was set to be the default text editor for CD++Builder plug-in (Figure 1). It inherited the `TextEditor` interface provided by "org.eclipse.ui.text.TextEditor". `cseEditor` uses `cseDocumentProvider`, which implemented methods of the `FileDocumentProvider` interface, provided by

"org.eclipse.ui.text.FileDocumentProvider". The `cseDocumentProvider` allows the plug-in to create and save the files by implementing the methods of `FileDocumentProvider` interface. To define the behaviour of the syntax editor, a configuration class must be created and associated with the coupling syntax editor (cse). The configuration class for the coupling syntax editor `cseEditor` was chosen to be the `cseConfiguration`, which implemented the `Source Viewer Configuration` interface provided by the "org.eclipse.jface.text.source.Source ViewerConfiguration" extension point. To create database of reserved words for highlighting, a `cseScanner` and `cseWordDetector` classes implemented interfaces `RuleBasedScanner` and `IWordDetector` respectively from the "org.eclipse.jface.text.rules" extension point.

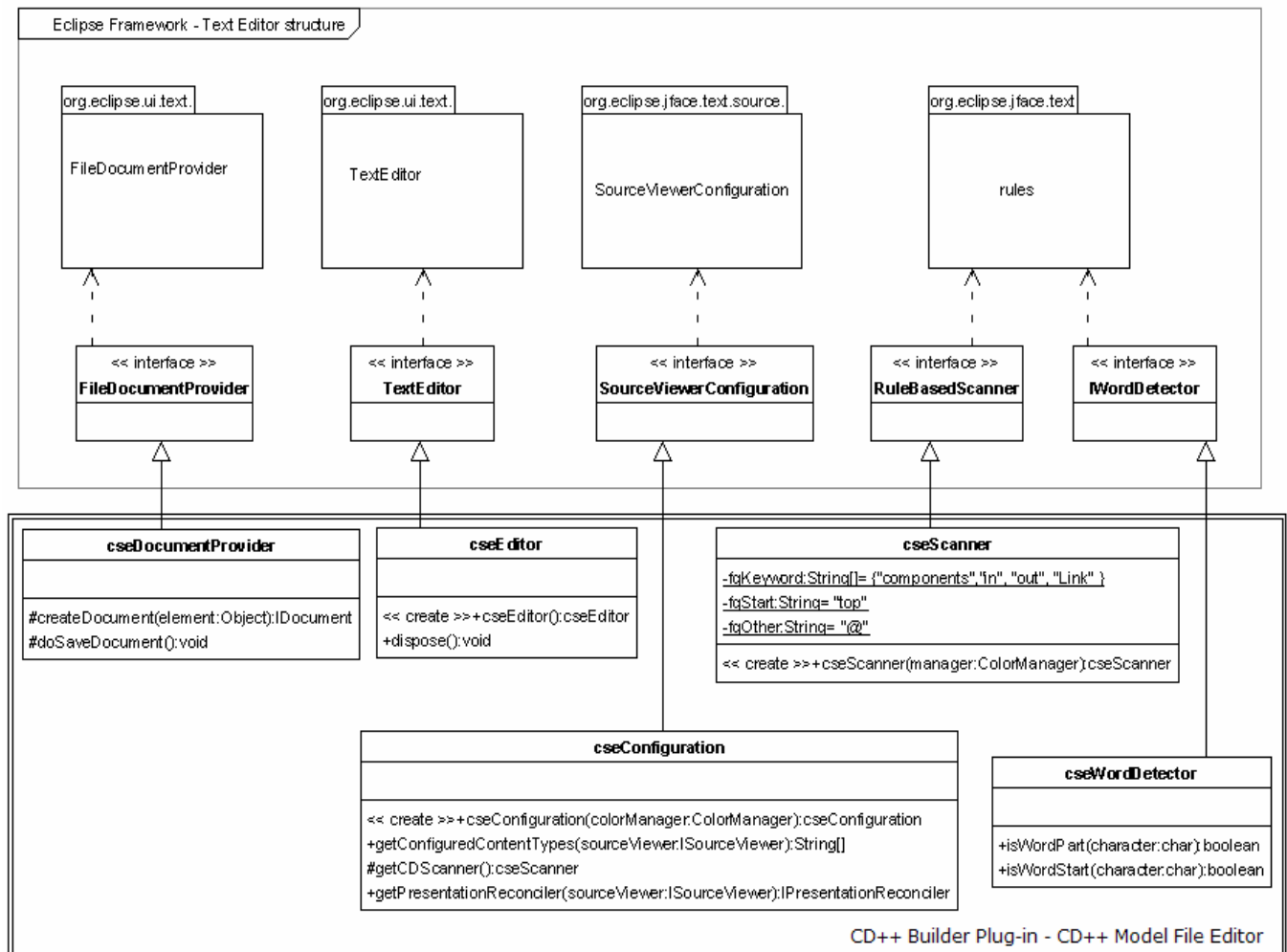
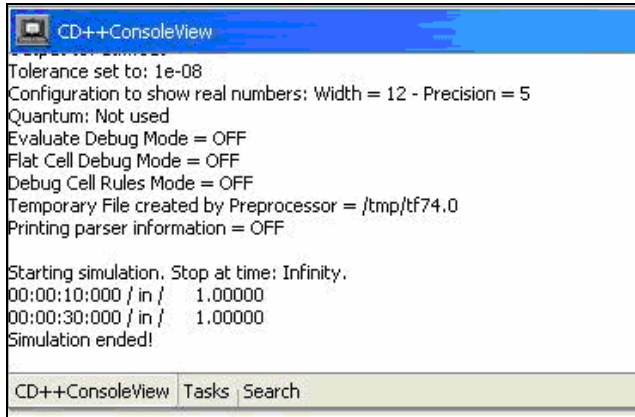


Figure 4. CD++ Builder text editor class diagram

The results of simulation and error reporting can be directly viewed in CD++ Builder perspectives in the console panel (Figure 5). The "org.eclipse.ui.views" extension point was used to create the panel. By implement-

tion point was used to create the panel. By implementing the `ViewPart` interface methods, the `ConsoleView` class handles the display of messages in the console. The text,

displayed in the console window, was managed by the `consoleDocument` class, which implemented the `AbstractDocument` interface. To store the text, displayed in the console window, the `ConsoleOutputTextStore` class was used, which extended the `ITextStore` interface.



```
CD++ConsoleView
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/tf74.0
Printing parser information = OFF

Starting simulation. Stop at time: Infinity.
00:00:10:000 / in / 1.00000
00:00:30:000 / in / 1.00000
Simulation ended!

CD++ConsoleView Tasks Search
```

Figure 5. CD++ Builder Console View

5. CONCLUSION

We presented the design of CD++Builder, an environment for building models based on the DEVS formalism. In the future, this environment will allow us to study

models in a simulated environment, and to execute them in a hardware surrogate. The hierarchical nature of DEVS permitted to do this without modifying the original models, providing the base for enhanced system development in embedded platforms. The environment permits an enhanced experience in the development of the applications.

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] "Modeling State-Based DEVS Models in CD++". G. Christen, A. Dobniewski, G. Wainer. In Proceedings of MGA, Advanced Simulation Technologies Conference 2004 (ASTC'04). Arlington, VA. U.S.A. 2004.
- [4] Glinsky, E. and G. Wainer. 2002a. Definition of Real-Time simulation in the CD++ toolkit. In Proceedings of the 2002 Summer Computer Simulation Conference. San Diego, USA.
- [5] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T.J. Grose, Eclipse Modeling Framework, 1 ed. , Addison-Wesley Professional, Aug. 200