Proceedings of ASME:
Computers in Engineering Conference
Atlanta, Georgia September 16-18, 1998

Put Paper Number Here

JAVA-BASED TOOLS FOR DEVELOPMENT AND DIAGNOSIS OF REAL-TIME CONTROL SYSTEMS

Will Shackleford/ NIST

Fredrick M. Proctor/NIST

ABSTRACT

The National Institute of Standards and Technology (NIST) has been using the *Real-time Control System* (RCS) Reference Model Architecture for building control systems based on a hierarchy of cyclically executing control modules. This paper describes the work done to build Java tools that allow developers to lay out their hierarchy according to RCS tenets and view or change the inputs and outputs of each module at run-time.

INTRODUCTION

This paper describes the Java tools built for RCS and their relationship to the *Real-time Control System* (RCS) Reference Model Architecture. An overview of the other software tools and libraries created for developing RCS applications is presented. Several of the unique challenges and strategies encountered in building tools of this sort in Java are also presented.[1]

BACKGROUND

Early work by Albus [1,2] and Barbera [3] in the Automated Manufacturing Research Facility (AMRF) led to the first definition of the Real-Time Control Systems engineering approach focusing primarily on software design. Later, software templates were created to make it easier to use RCS, encourage consistency both between modules in an application and between applications, and provide better structure (see Quintero [4]). Libraries of software were also created. These include:

- Neutral Manufacturing Language (NML) for communications
- Portable Application Programming Interfaces (API) for accessing operating systems services
- Functions for coordinate system transformations
- A base class for NML control modules.

`Makefile` definitions and directory structures were developed to allow multiple programmers to work together to build applications which run on multiple platforms. Before the tools were developed, control systems were built by copying and renaming several C++ source files, doing several search and replace operations and then populating the source files with the appropriate functionality for that particular application. Information on how the control system was behaving was gathered by running source code debuggers and/or by watching displays of custom built graphical interfaces. This information was used to diagnose problems and suggest improvement.

GOALS

Goals for developing the tools are as follows:

1. Allow current users of the RCS source files to quickly and inexpensively develop applications of higher quality.
2. Provide means for initiating new RCS developers with a much shorter learning curve.
3. Allow problems to be found and diagnosed more easily.
4. Provide means to control or view the activity of a control system remotely through a web browser.
5. Provide a new perspective from which to view RCS and feed this back into our research into open architecture controllers and intelligent systems.

---

[1] PRODUCT ENDORSEMENT DISCLAIMER

References to specific brands, equipment, or trade names in this document are made to facilitate understanding and do not imply endorsement by the National Institute of Standards and Technology.

DEVELOPMENT PROCESS

Figure 1 shows a diagram of the development process and the position of the tools we built within it. The process begins with analysis of the control problem and a design of the RCS architecture. Although we do not provide any tools for this part of the process, it is essential. Once the developer has a basic idea of the hardware to control and the major tasks this system will be asked to perform, the developer can proceed to draw the hierarchy inside the RCS Design Tool. The hierarchy will consist of several cyclically executing control modules. Each module receives commands from its superior and sends commands to one or more subordinate modules. Subordinates provide status information to their superiors. The control modules will be examined in more detail later.

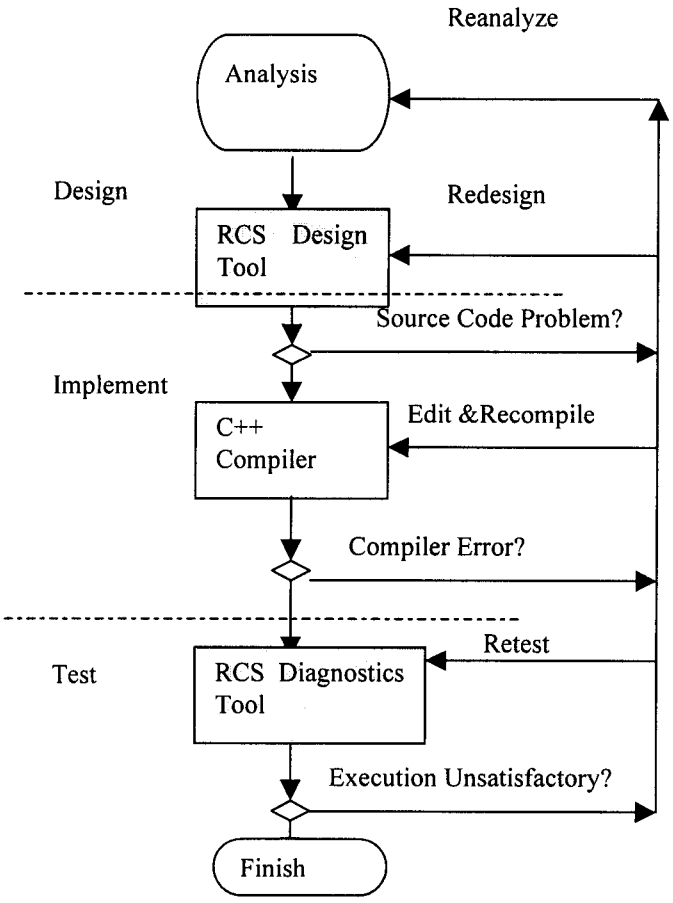

Figure 1. Development Process

The design tool then generates the C++ source code with stubs, `makefiles`, and configuration files. The source code may be edited either with a separate text editor or within the design tool. A `makefile` interpreter is run to call the compiler and generate the executable(s). These executables may be tested by running them with the RCS Diagnostics Tool. The diagnostics tool shows a hierarchy that is identical to the one from the design tool except that the current commands and status from the running controller are displayed. The tool also allows variables to be plotted, state tables to be examined, and errors to be logged. After each step of the process, the developer decides whether to proceed or go back to an earlier point in the process. Therefore, there is a need to be able modify to the existing system and allow as many options as possible to be kept open at each level.

RCS DESIGN TOOL

The purpose of the RCS Design tool is to provide a graphical view of the application hierarchy and to generate the C++ source code, configuration files and makefiles needed to build the application. Several views may be selected by a drop-down list in the upper left.

- `Hierarchy View`, Figure A1 - The left side has a set of controls used to add modules, select subordinates, and add commands and auxiliary channels. The right side has a graphical depiction of the hierarchy. The selected module appears highlighted. As the user enters modules in the "`Add Module`" text field on the left, the modules are added as subordinates of the selected module.
- `Files View`, Figure A2 – This is a convenient place to view or modify the generated source code and configuration files. A list of all the files needed to build and run the application is on the left of files view. A simple text editor showing the file selected from the list is located on the right.
- `Options View` -This view is populated with controls to change miscellaneous options such as the directories where files are written. (Not Shown)
- `Loops View` - The developer may use this view to assign modules to execution loops. All the modules in the same execution loop are linked together at compile time and must run on the same host at the same cycle time. However, there are advantages in synchronization to having the modules linked together. (Not Shown)

Buttons for creating the source code, running the makefile interpreter, running the application, and printing the graphical view of the hierarchy are located at the top of the design tool. A progress bar shows the progress and name of major tasks, such as creating source, which may take a few seconds.

GENERATED FILES

The design tool generates several different types of files. The detailed information needed to understand and work with each type of file is provided on the RCS Library Web Page [5]. An overview follows. The example files may also be found on the website.

NML message header files are C++ header files that define the classes of objects that can be sent to the module as commands or received from the module as status. We usually name the file according to the following convention <module name>+'n.'+<C++ header ext.>. (Some programmers prefer .h as a C++ header extension while others use .hh or .hpp to distinguish them from C header files. This is an option in the design tool.) The design tool generates a class for every command entered and one class for status. The programmer will probably add variables to these classes

Module header files define the C++ class that contains all the functions for the module. These functions are called by the NML_MODULE base class when the appropriate message is received, or every cycle, but are not used by other modules directly.

Module source files provide stubs for each command that the module accepts. Inside these stubs, the programmer is expected to add state tables related to each command and probably send commands to its subordinates using NML communication channels that have already been set up. The state tables for INIT and HALT are already set up by the design tool. The code from the state table can be made available to the diagnostics tool so the current line in the state table can be watched.

Main source files contain the main function used to start execution. The main function creates an object for each type of module. Each module's controller function is called inside a loop that pauses for a timer each cycle to provide consistent cycle times.

Other files generated include:

1. NML Configuration files that allow communications parameters and protocols to be selected.
2. Makefiles that control how the source code is compiled.
3. A configuration file used by the RCS Diagnostics Tool.
4. A shell script that allows multiple executables to be started with a single command.

SOURCE CODE MERGER

One problem code generators often have is a conflict between the automatic tool generating code and the programmer manually creating or modifying code in the same file with a separate text editor. A programmer may already have edited code inside a module and then decide to add a command or subordinate to that by running the code generator. Unless the code generator keeps the manual edits, the programmer might lose the work he has already done.

To solve this problem, the tool uses a component called the merger that allows both user edits and automatically generated code to be merged in the same file. The position of the merger within the RCS-Design Tool is shown in Figure 2. To the rest of the design tool, the merger appears to be an object with functionality that is similar to an output stream. As each line is written, it is compared to see if it matches a line that already exists in the file. If the line does not match any of the lines in the file, it is inserted in a position that keeps it in order with the lines that do match. The conditions for a match were altered to make the merger work more effectively. Because loading a file in some editors changes the way tabs and spaces or new lines are stored, the white space is stripped out before the comparison is made. In addition, a nesting level is computed for each line. The nesting level must match for the lines to be considered matches. The merger can be disabled either for short periods inside the code using special comments or for the whole file using an option inside the tool. When the merger is disabled, the output is sent directly to the file without checking the old source file.

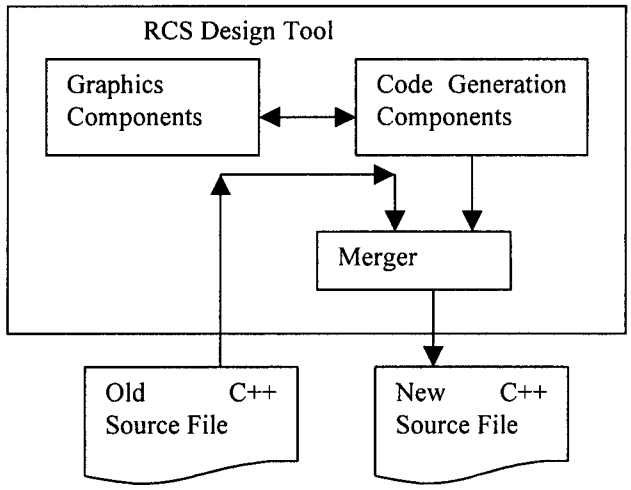

Figure 2. Internal RCS Design Tool Structure

RCS DIAGNOSTICS TOOL

Once the software has been compiled, it must be tested. Previously, we tested our systems with traditional source code debuggers. However, by developing a diagnostics tool specifically for RCS applications, we can provide many features that traditional source code debuggers do not.

1. Show the entire hierarchy in one view with the current commands and status.
2. Highlight the current line in the state table.
3. Allow data to be taken from the system without stopping it.
4. Allow multiple programmers to simultaneously view the state of the machine across a LAN or the Internet.
5. Control the system by sending commands to any module.

When the tool is started, it needs to load a configuration file. The user can type the file name or URL of the configuration file into the text box at the top. The location can also be pre-set with the HTML tag <PARAM

name="ConfigurationFile" value="app.cfg"> if the tool is to be loaded from a website dedicated to that application.

A diagnostics configuration file can be generated by the design tool or manually created with a text editor. This file provides information on the relationships between modules and the locations of the other files used in the application such as module source files, communications message headers, and communications configuration files that the diagnostics tool can use. It is not necessary to provide all of the files. For example, if the programmer preferred not to provide the module source code, the state table display would not be available. However, the developer could still send commands and examine status from the module.

The RCS Diagnostics tool has several views that can be selected with the drop-down list in the upper left.

- The `hierarchy view`, shown in Figure A3, provides a graphical representation of the hierarchy similar to the one in the design tool except that each module displays the current command and status type of the module. The color of the boxes in the hierarchy change depending on the status type. (DONE, EXECUTING, ERROR) Clicking on any of the boxes brings up a menu of the commands that can be sent to that module.
- The `details view`, shown in Figure A4, provides much more information about one particular module selected with a list control. The information includes the current value of every variable in the current command and status classes rather than just the type. Any variable can be selected and added to the list of variables that are logged for plotting.
- The `graph view`, shown in Figure A5, shows a running plot of each variable that was selected from the details view for plotting versus time.
- The `state table view`, shown in Figure A6, displays the source code of the currently selected module with the current line in the state table highlighted.

JAVA CHALLENGES

Java has some unique advantages for this type of application, but it also creates some special challenges as well. Since the Java code must be able to run on any platform, some platform specific functions are not available. For these tools, functions are needed for setting file permissions, interpreting makefiles and creating symbolic links. A recommended approach for handling this limitation is to check `System.getProperties ("os.name")` and then call `Runtime.getRuntime().exec( ... )`. This technique is not 100% Pure Java, but it allows operating system dependant code to be localized for porting between platforms.

Security barriers built into web browsers introduce some restrictions on running Java code within web browsers. One restriction is that applets generally may not open socket connections to hosts other than the web server from which they were loaded. This might for example prevent the diagnostics tool from being able to communicate with some or all of the modules. The following strategies can be used to solve this problem:

1. Run a web server on the same host as the control system and have browsers download the applet from it. Free or very inexpensive web servers are available for most platforms. This strategy allows the applet to connect to the modules on that host in almost any browser that supports Java.
2. Run a proxy program. A simple program called `tcpproxy` is included in the RCS library. It may run on the web server and accept requests from applets, forward them to the appropriate host where the modules run, and forwards the replies from the modules back to the applet. It can also be used on the host where the modules run to forward requests to the web server. Since the applet only needs to open a connection to the same host from which it was loaded, no security exception occurs.
3. Sign the applet. Microsoft Internet Explorer 3+, Netscape Navigator 4+ and HotJava all provide means of identifying signed applets and will allow those applets to connect to any host after the user warning. Unfortunately, they use three different types of files to store the signatures. This provides the minimum overhead on the server side. However, this method does not allow users with other browsers or with concerns about letting go of the security protections to communicate with the modules.

The challenges and advantages of Java seem closely linked. The cost for improved portability is the lack of some operating system specific functions. Building an application to run inside a web browser allows it to be distributed more easily and provides the user with better security; however, it requires some additional work to get the same flexibility.

CONCLUSIONS

RCS provides a very good match for graphical software design and diagnostics tools. RCS hierarchies are abstract enough to provide a high level graphical view of the application, yet concrete enough to be used for building and testing real applications. Using Java to build those tools allows them to be distributed to multiple platforms easily. Although Java also creates some challenges, those challenges can be overcome without too many burdens.

REFERENCES

[1] Albus, J. S., "Outline for a Theory of Intelligence," IEEE Transactions on Systems, Man, and Cybernetics, Vol. 21, No.3, May/June 1991.

[2] Albus, J. S., "RCS: A Reference Model Architecture for Intelligent Control," Computer, May 1992.

[3] Barbera, A.J., Albus, J.S., M.L. Fitzgerald, and L.S. Haynes, "RCS: The NBS Real-Time Control System," Robots 8 Conference and Exposition, Detroit, MI June 1984.

[4] Quintero, R., Barbera, A.J., "A Software Template Approach to Building Complex Large-Scale Intelligent Control Systems," 8th IEEE International Symposium on Intelligent Control, Chicago, IL, September 25-27, 1993.

[5] Shackleford, W. , "RCS Library Web Page", http://isd.cme.nist.gov/proj/rcs_lib.

Appendix

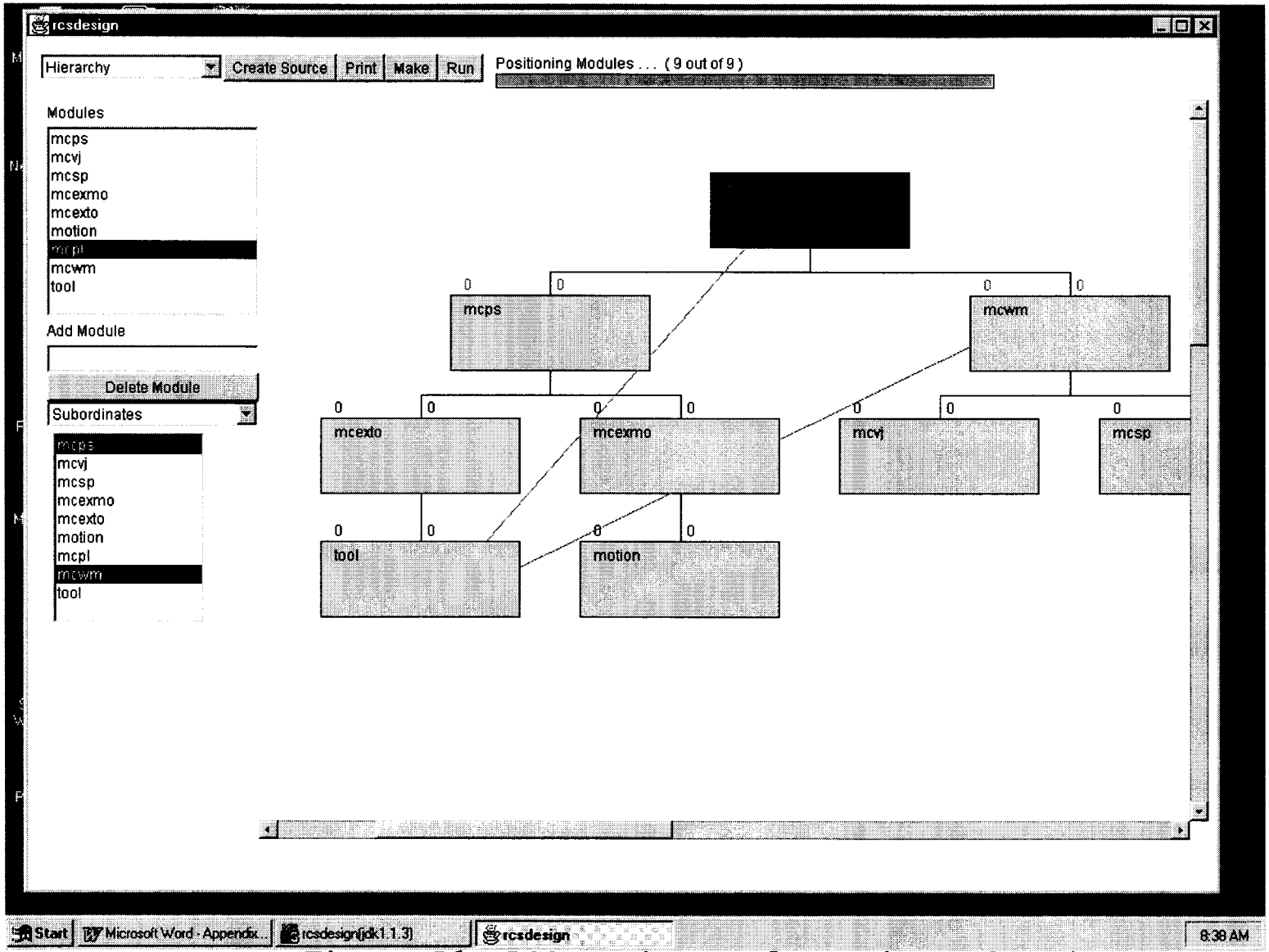

Figure A1. RCS Design Tool -- Hierarchy View

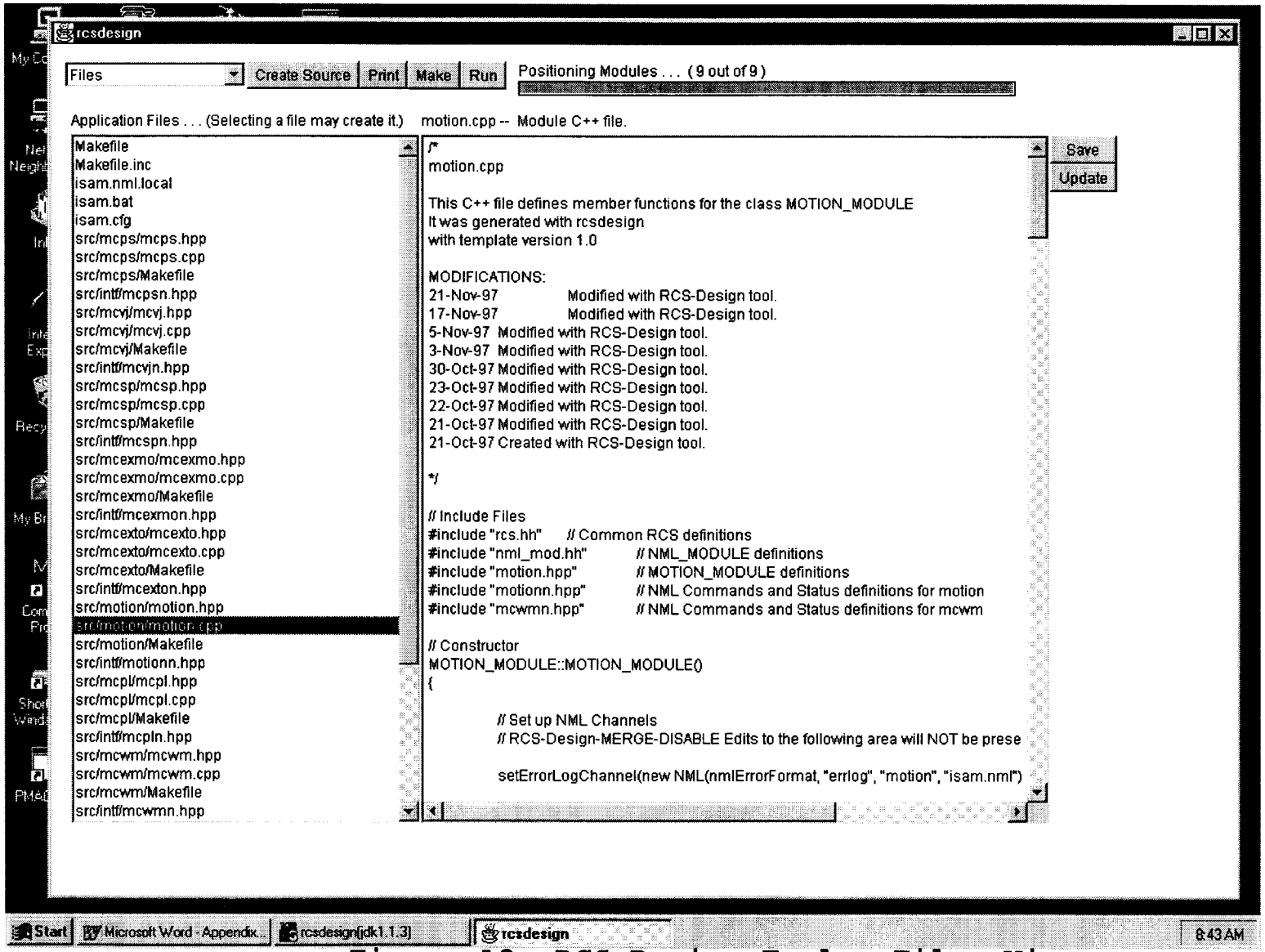

Figure A2. RCS-Design Tool - Files View

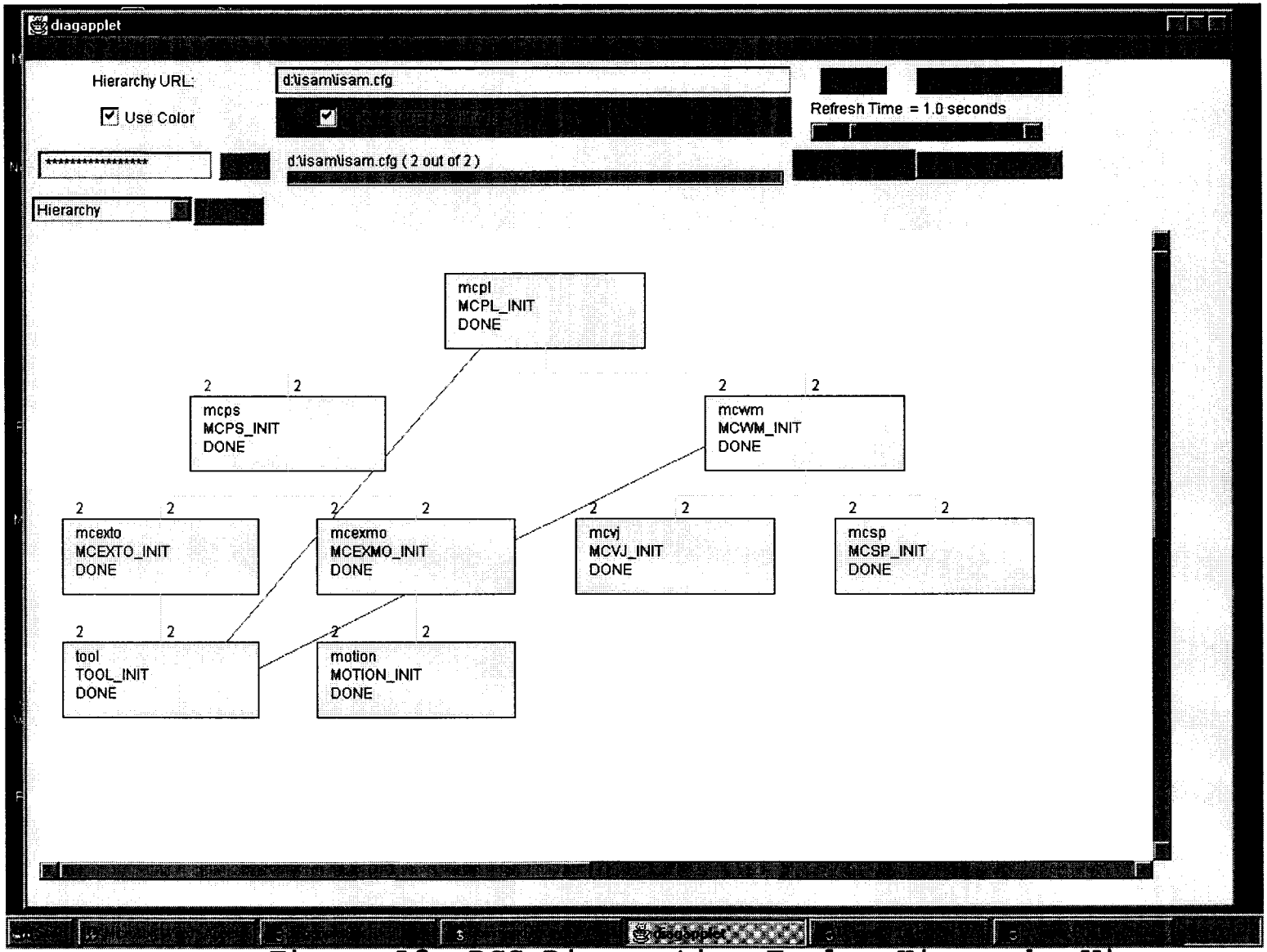

Figure A3. RCS Diagnostics Tool - Hierarchy View