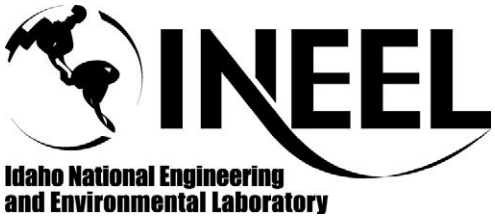

INEEL/CON-04-02221
PREPRINT

Java XMGR

Steven P. Miller
Dr. George L. Mesina

August 24-27, 2004

RELAP5 International Users Seminar

Java XMGR

Steven P. Miller and Dr. George L. Mesina

Idaho National Engineering and Environmental Laboratory (INEEL)
Idaho Falls, ID 83145-3880, USA
Phone: 208-526-1433
Email: millsp@inel.gov, mesinagl@inel.gov

Proceedings of the RELAP5 International Users Seminar
Sun Valley, ID, USA, August 24-27, 2004

KEYWORDS: Java, XMGR5, RELAP5

Abstract

The XMGR5 graphing package [1] for drawing RELAP5 [2] plots is being re-written in Java [3]. Java is a robust programming language that is available at no cost for most computer platforms from Sun Microsystems, Inc. XMGR5 is an extension of an XY plotting tool called ACE/gr extended to plot data from several US Nuclear Regulatory Commission (NRC) applications. It is also the most popular graphing package worldwide for making RELAP5 plots. In Section 1, a short review of XMGR5 is given, followed by a brief overview of Java. In Section 2, shortcomings of both tkXMGR [4] and XMGR5 are discussed and the value of converting to Java is given. Details of the conversion to Java are given in Section 3. The progress to date, some conclusions and future work are given in Section 4. Some screen shots of the Java version are shown.

1.0 Background

1.1 XMGR5

An excellent summary of what XMGR5 is can be found in the preface of the user manual [1]. "The XMGR5 software package is derived from an XY plotting tool for UNIX workstations called ACE/gr. The interactive version of ACE/gr is XMGR and includes a graphical interface to the X Windows system. Enhancements to the XMGR plotting tool have been developed which import, manipulate, and plot the output data from RELAP5 MELCOR [5], FRAPCON [6], and SINDA [7], and from U.S. Nuclear Regulatory Commission (NRC) databank files. The resulting product is called XMGR5." This was originally written by Paul J. Turner.

The first graphical version of XMGR was written using a Motif interface. Motif [8] is the "UNIX industry's standard user interface originally developed by the Open Software Foundation (OSF). Motif is based on the X-Window system and is a Presentation Manager look-alike." This version, originally released in 1991, is still a powerful plotting package. It has been modified to work with RELAP5 and now with RELAP5-3D©. Over

the years, numerous developments have been made at the INEEL to XMGR5 including adaptation to LINUX in 1996 and addition of the Cyrillic alphabet in 1998.

Unfortunately, XMGR5 does not run natively on MS-Windows platforms because of the use of the X-Window based Motif interface. For this reason in 1999, a second version of XMGR5 that does run natively on MS-Windows platforms was written. It replaces the Motif library calls with calls to the Tool Command Language/Tool Kit (Tcl/Tk) [9]. Tcl/Tk was chosen as the replacement windowing library for Motif for several reasons. First, it runs on all major operating systems, UNIX, MS-Windows, and Macintosh. Second, there is free version of Tcl/Tk that, at the time, was being maintained and developed. Third, the RELAP5 Graphical User Interface (RGUI) [10] had been under development for a year in Tcl/Tk and it was thought a good idea to have both user interfaces use the same graphics tools. This would reduce maintenance and development costs.

This version of XMGR5 that uses Tcl/Tk graphics tools is called tkXMGR [4]. Although tkXMGR does run on a variety of operating systems, Tcl/Tk has some disadvantages. A better choice, that was not available when tkXMGR5 was designed, is Java.

1.2 Java

Like Tcl/Tk, Java is freeware; it can be downloaded from the Sun Microsystems web site [11]. Also, like Tcl/Tk, use of Java is not encumbered with license issues, such as the GNU public software license, that can restrict or complicate use of XMGR5 or RGUI. Portability and licensing were perhaps the most important issues leading to the original choice of Tcl/Tk and of Java now.

Java has become the most popular programming language in the world. Its popularity has also lead to a number of books being written and courses on Java being developed. Many very good language features are included in Java because its popularity has lead to a great deal of development of the language. A number of Java class libraries are available to extend the capabilities of the language even further. So activities like dialog screens, file lookup and graphics conversion have already been created for ease of use. Compare the image generated by Tcl/Tk version of XMGR (Figure 1a) with the image generated by the Java version on the same data (Figure 1b). The Tcl/Tk version seems to use a screen capture because it also captures portions of the open dialogs, where the Java version uses graphics calls to generate the output.

Java also has many features that make it ideal for Graphical User Interfaces (GUI). One feature is the true object-oriented language nature of Java. That programming paradigm is particularly well suited to graphical user interfaces. Java has built-in security measures, mostly because it was designed for use in internet programming. Java is also architecture neutral; the Java byte-code can run on any platform that has a Java Virtual Machine (JVM). This makes Java programs quite portable between disparate computer platforms. Another important feature of Java is that it is multi-threaded, so it can take advantage of multiprocessor systems.

In fact, a number of GUIs in the nuclear industry are now being written with Java. Just for RELAP5, the prototype Java RELAP5-3D Graphical User Interface (RGUI) [12, 13], prototype Thermal Hydraulic User Model Builder (THUMB) [14] and SNAP [15] have been written in Java. It was found that the software aids, such as J-Builder, helped make the job of working with Java go much faster than originally anticipated.

2.0 Reasons for conversion

There are a number of reasons to develop a Java version of XMGR5. Both the Unix Motif version and the Tcl/Tk of XMGR5 have deficiencies. As mentioned in the Section 1, Java also has numerous advantages that will be useful to incorporate.

2.1 XMGR5 deficiencies

The primary deficiency of XMGR5 is that it makes use of operating system specific interface libraries and machine dependent coding. It will only run natively on an operating system that supports Motif. This restricts XMGR5 to UNIX. There are ways around this, such as by running a UNIX emulator on a Windows platform, but emulators have been found to be less reliable and incur a cost to the code user. Supplying an emulator also complicates the distribution of the XMGR5 product with license issues.

Beyond the Motif issues, XMGR5 is written in the C programming language. Special installation procedures and conditional coding are part of the XMGR5 package. This means that in order to run a platform specific version of XMGR5, it must be compiled on that platform, or one just like it. As the operating systems evolve and change, these special procedures may have to be upgraded or replaced altogether.

A fully integrated Java version of XMGR5 would have none of these problems. Java is a programming language where one can write a program once, and run it anywhere. The Java version, when fully developed, would be able to run on every platform that supports the Java Runtime Environment (JRE) version 1.4.2 or later. At this time this includes the following platforms: Windows (IA64, XP, 2000, ME, NT, 98 or 95), Linux (Intel or IA64), Solaris (SPARC or x86), and Mackintosh (OS 8, 9 or X) [11]. These are simple to install and comes already installed on most new computers.

These same reasons, portability, cost and licensing issues, were the reasons tkXMGR5 was developed. However, there are deficiencies in both Tcl/Tk and tkXMGR5. The major deficiency of tkXMGR5 is that it is incomplete; some features of XMGR5 are not enabled in tkXMGR5. The primary problem with Tcl/Tk is that the development of the freeware form of Tcl/Tk has stopped and there is little or no maintenance.

Because operating systems continue to change and new ones will be created, the lack of maintenance and development of the Tcl/Tk freeware will eventually cause it to operate incorrectly or fail completely on some future operating systems. Also, GUI development continues in the computing industry. New and useful widgets are being created that are unavailable in the freeware version of Tcl/Tk. These include multi-tabbed folders and

multi-document interfaces. Further, since development of the freeware version of Tcl/Tk has essentially stopped, certain problems will not be fixed. These include unwanted differences between screen displays in UNIX and PC, interactions with the operating system and focus problems.

There were a number of deficiencies in tkXMGR5 and some of the popular features of XMGR5 were not enabled. These include:

- Help is not enabled.
- Does not read non-RELAP5 data.
- Obvious "stair-casing" in plotted lines and curves
- Output of save-all cannot be read back in.
- Auto on feature (left side tool bar button) does not work.
- No R2DMX capability.

3.0 Conversion to Java

3.1 High-level considerations

In order to be sure that all the features the users want would be enabled in the Java [15] version of XMGR5, input from the principal users of XMGR5 at INEEL was obtained. These consultants also know the needs and wants of XMGR5 users worldwide and were able to represent them. By consulting with them, the following prioritized list of features to enable and the order in which they should be enabled was developed:

- Read in a RELAP5 file and Draw Graph
- Print Graph (Printer, Postscript, .jpeg, etc.)
- Rescale Plots (on either axis or both)
- Titles (axes, strings arrows, legends)
- Transformations – regression, evaluate expressions (unit conversion)
- Ability to modify Line width, Symbols, Colors, etc.
- Run Scripts
- Display Sets Status
- Arrange multiple graphs on a page (specify columns and rows)
- Ability to set default to auto scale when reading
- Unload a file

This group also indicated that they preferred much of the look and feel of the original version of XMGR5 to the current Tcl/Tk version.

The ultimate goal of the Java conversion project would be a completely Java version of XMGR. However, the original XMGR code was written in the C programming language, not some object oriented language, such as C++. The programming paradigm for Java is object oriented and a complete rewrite would start at the algorithmic level. Very little of the original code would be saved. The ultimate goal would therefore require a complete rewrite of XMGR5 and that is beyond the scope of this task

It suffices to follow the lead of the Tcl/Tk conversion project and simply convert the use of the Motif graphics library tools to Java tools. Much of the functionality in tkXMGR5 is still contained in the original C code. Most of the work involved in creating tkXMGR5 was in replacing Motif library use by accessing the Tcl/Tk libraries and in developing communications between Tcl/Tk and the C coding.

Choosing to retain the C language coding means that, in many instances, the code will still have to be compiled on each individual workstation. Therefore, throughout the project, any changes in the original code were made with this in mind. Standard ANSI functions were used wherever possible to produce the most generic code. This will reduce future maintenance and development costs. It should also significantly reduce the time it takes to compile on most computers. For example, in the current Windows version, only 2 files are required for XMGR to run and no on-site compiling is necessary.

3.2 Details of the work

Before the items listed in Section 3.1 cane can be worked on, there must be a base GUI from which to work. This will have a window frame, a skeleton menu bar, perhaps other skeleton bars, and a graphing area. None of these will do anything at first; they are simply placeholders. The base GUI will have a similar appearance to XMGR5. Once the base GUI is constructed with Java, the actual work of activating the features of XMGR5 can begin.

All the original code was written in C with calls to the Motif library. Because Motif is being replaced by Java, there are a few programming issues. First, the Java coding must communicate with C coding (to activate XMGR5 procedures and to send data) and C must to talk back to Java. Fortunately, Java has a translator called the Java Native Interface (JNI) to support both these functions. The second issue is the determination of which functions would be best left in C and which should be rewritten in Java. At the outset it was decided that all of the visual portion of the GUI would be developed on the Java side and all the computational formulas and functions would remain on the C side of things.

The work of creating Java XMGR5 then can be broken into two parts, the infrastructure work and the work of activating XMGR5 features. The activation work is reported in Section 4.0. The infrastructure work was broken into four distinct tasks or steps. These are:

- Build the base GUI
- Establish communication from Java to the C procedures of XMGR5
- Establish communication from the C procedures of XMGR5 back to Java
- Extract data from a RELAP5-3D restart-plot file

The first step was to construct the visual or on-screen portion of the base GUI (Figure 2). This is akin to building a circuit, creating leads that will later be hooked up to something larger. First, the work area with tool, menu and status bars was built. Second, a graphical area that would serve as a plotting screen later was constructed. At this point, it looked like XMGR5 with some modernization, but none of the buttons did anything

when clicked except to activate a screen that informed the user that the function was not available yet.

In order to make a button function or to have a mouse action in the plotting area cause something to happen, the Java visual coding must communicate information and a request to act to the appropriate XMGR procedures that are written in C. Then the C procedure must perform some operations and both communicate some data back to the Java coding and return control to the Java side. Tasks two and three then are to establish these communications and control flows.

Specifically, the second task was to use the JNI to communicate from the Java visuals to C coding in order to give function to the buttons and retrieve necessary information for display. It took approximately one week to get this to work. This took considerably less time then to develop that same line of communication from Tcl/Tk to C; there is no JNI equivalent so one has to create Tcl/Tk commands and "register" them to accomplish many of the things that JNI supplies.

The third task is enabling the communication of data from C to Java. There is a sizeable list of functions that must be created on the C side to communicate with the JVM. These functions must pass data they calculate to the Java side. Various means were tried that failed before a successful method was found. For example, one cannot create a new instance of the JVM from the C side because there can be only one instance of the JVM running from a given process at any time. Also, one cannot pass the JVM parameters into C and make them global so that any function that needs to use the parameters can use them because of a Java feature called the "Garbage Collector."

The Java Garbage Collector seeks data that is both stored in memory and is not being used; then it essentially deletes the data. This means that the parameters are created and destroyed every time they are called. A global variable is like a sign pointing to where the parameters are kept. After the parameters are picked up by the garbage collector they may or may not be recreated in the same place. So the sign pointing to where the parameters should be pointing to something else and this can cause problems. It would be equivalent to moving a blind person's furniture every couple of minutes; eventually there will be a crash.

Finally a successful communication method was found, namely, passing all the necessary parameters to each function involved in every process that involved the C side talking back to the Java side. This meant changing some 300+ functions to include the necessary Java parameters. This proved to be very time consuming, but once this arduous task was finished, communication from C to Java worked. This completed task 3.

The fourth task was to read in a RELAP5 restart-plot file. This is half of the first item listed in Section 3.1 and it involved two steps. The first was to read the restart-plot file and place the data into the memory of the C coding. The second was to access a list of XMGR5 channels; this is actually a list of data headers.

4.0 Progress, future work, and conclusions

With steps one through four accomplished, it became possible to hook up functionality to the buttons of the new interface to retrieve data from a RELAP5 restart-plot file, perform calculations with it, and transmit it to the Java side. It would then be possible to display the data. To accomplish this, it was necessary to write both the Java coding that button and dialog box activates to communicate with the appropriate C coding and the Java coding that would display the data on the screen. This has been done for the following XMGR5 capabilities (Figure 3):

- Read in a RELAP5 restart file and choose which channels to plot.
- Plot a graph from a RELAP5 restart file.
- Auto scale the graph.
- Zoom in and out on the graph.
- Move the graph up, down, left, and right.
- Change the view.
- Automatically resize the plotting panel.
- Automatically save settings (working directory, last file used, screen size and placement) on exit.
- Read status of sets, graphs, variables, and regions.
- Show fonts (e.g. Cyrillic).
- Set Auto-Ticks.
- Export or Print an image.

This summarizes the work done on XMGR5 during the project to date. The work that must be performed to complete the project includes completion of all items is listed in Section 3.1. Note that many of these items have a number of subtasks associated with them. Additional work that is not listed there includes testing it on numerous operating systems (which should require very little code modification), preparation for future transmission with RELAP5-3D product releases, and writing appropriate documentation. Also, there are additional features of XMGR5 that are not listed in Section 3.1 that would also be desirable to activate and some or all of these may become future work scope.

Acknowledgements

I would like to give special thanks to: Bob Bell, for providing me with the tools I needed to get this project moving, Jim Fisher; for all his technical expertise and patients, without which I would not have gotten started; Jim Galbraith, for answering my questions in a timely manner and in a way I could understand; and Guy Grinnell, for letting me in on the secrets of the JNI and saving me days worth of research.

References

[1] Jones, K. R. and Fisher, J. E. “XMGR5 Users Manual,” INEL/EXT-97-00346, Idaho National Engineering and Environmental Laboratory, March 1997.

[2] RELAP5-3D Code Development Team, "RELAP5-3D Code Manual," INEEL-EXT-98-00834 Revision 2.3, Idaho National Engineering and Environment Laboratory, June 2004.

[3] Horstmann, C. S. and Cornell, G., Core Java 2 Volume II-Advance Features, 4th Edition, ISBN 0-13-092738-4, Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[4] Mesina, G. L. and Galbraith, J. A., "New Developments and Value of the RELAP5-3D Graphical User Interface (RGUI 1.2)," Proceedings of the RELAP5 International Users Seminar, Jackson Hole, WY, USA, September 12-14, 2000.

[5] Gauntt, R. O., et al, "MELCOR Computer Code Manuals," NUREG/CR-6119, SAND2001-0979P, Sandia National Laboratory, April, 2001.

[6] Berna, G. A., Beyer, C. E., Davis, K. L., and Lanning, D. D. "FRAPCON-3: A computer Code for the Calculation of Steady-State, Thermal-Mechanical Behavior of Oxide Fuel Rods for High Burnup," NUREG/CR-6534, Vol. 2, PNNL-11513, December 1997.

[7] Gaski, J. D., "SINDA 1987/ANSI Computer Code," Network Analysis Associates, Oct. 1987.

[8] *Glossary*, Maui High Performance Computing Center, 1995.

[9] Welch, B., Practical Programming in Tcl and Tk, 2nd Edition, ISBN 0-13-616830-2, Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[10] Mesina, G. L. and Galbraith, J.A, "RGUI 1.0, New Graphical User Interface for RELAP5-3D," Proceedings of the 7th International Conference on Nuclear Engineering (ICONE-7), Tokyo, Japan, April 1999.

[11] Sun's Website http://java.sun.com

[12] Grinnell, G. E. and Mesina, G. L., "Java RGUI," Proceedings of the RELAP5 International Users Seminar, West Yellowstone, MT, USA, August 27-29, 2003.

[13] Mesina, G. L., "Visualization of RELAP5-3D Best Estimate Code," American Nuclear Society Winter Meeting, Best Estimate Seminar, November 2004, to appear.

[14] Cutforth, M. and Mesina, G. L., "THUMB: Thermal Hydraulic User Model Builder," Proceedings of the 2002 RELAP5-3D International Users Seminar, Park City, UT, USA, September 4-6, 2002.

[15] Gitnick, B. J., Gitnick, M. T., Larson, D., and Tomlinson E. T., "Extension of the SNAP Model Editor to Support the RELAP5-3D Code," Proceedings of the RELAP5 International Users Seminar, West Yellowstone, MT, USA, August 27-29, 2003.

Figures:

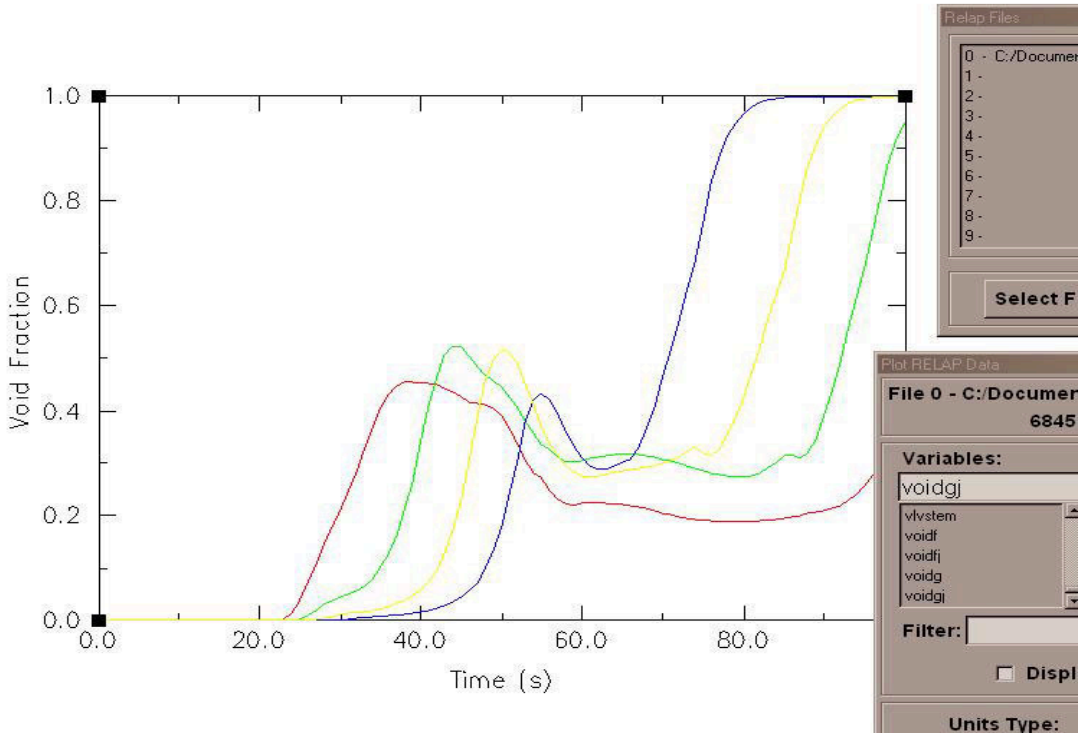

Figure 1a: tkXMGR5 Exported Plot

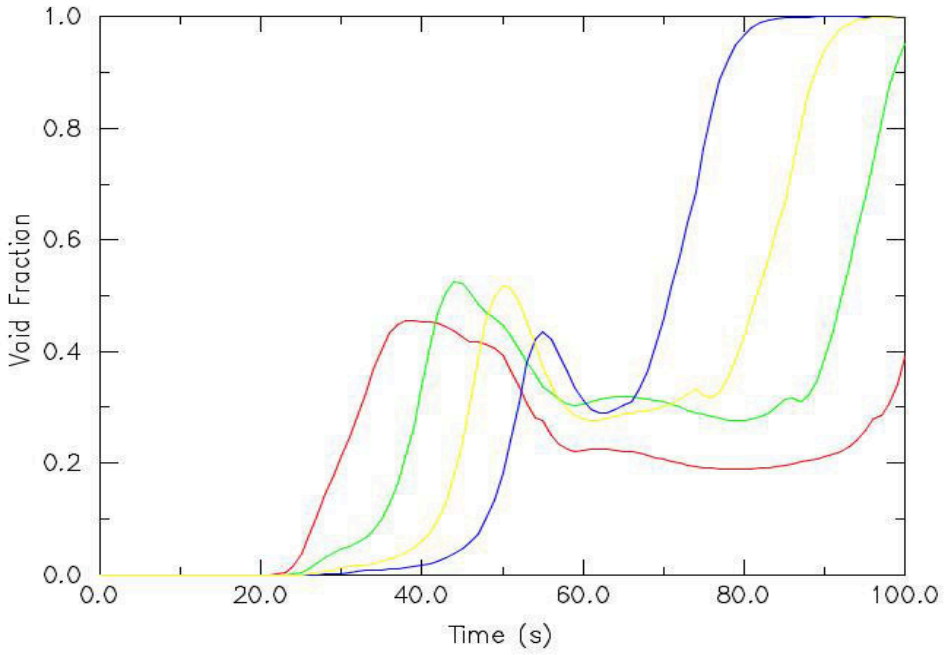

Figure 1b: Java XMGR Exported Plot on the Same Data as Figure 3a

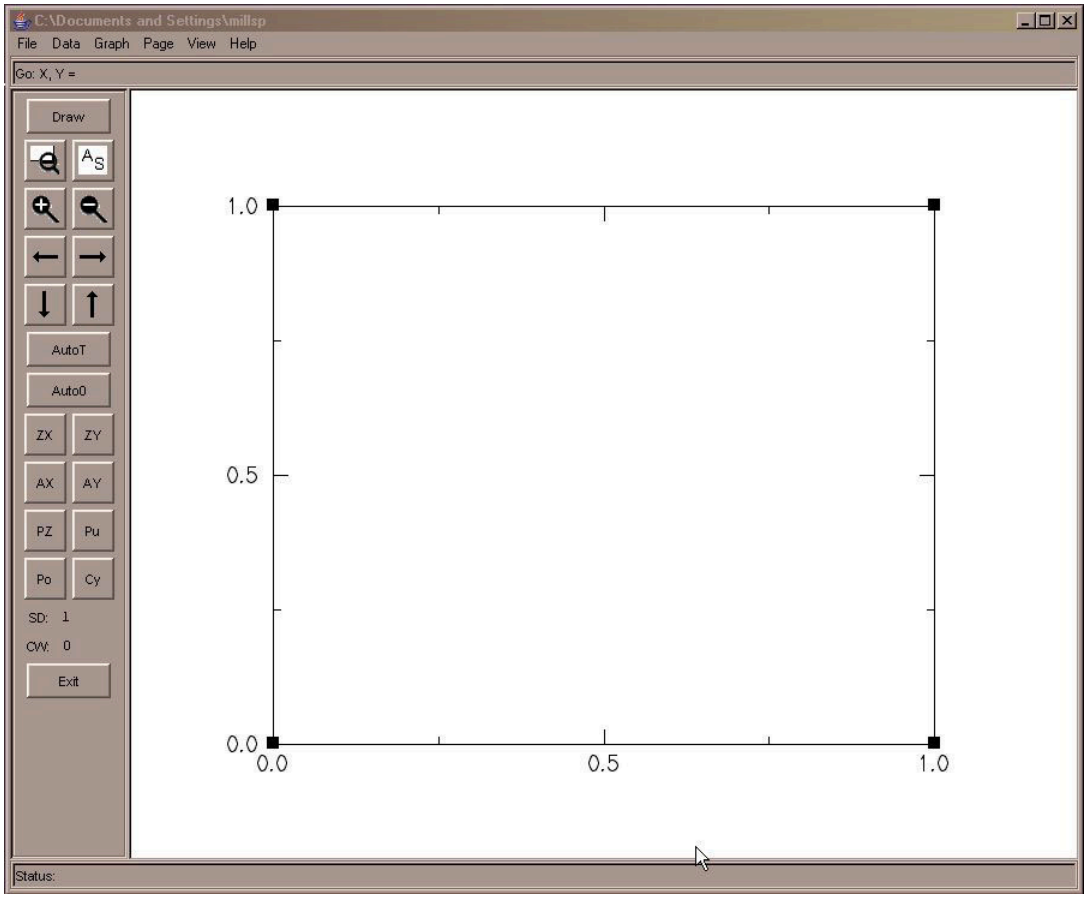

Figure 2: Java XMGR main Screen

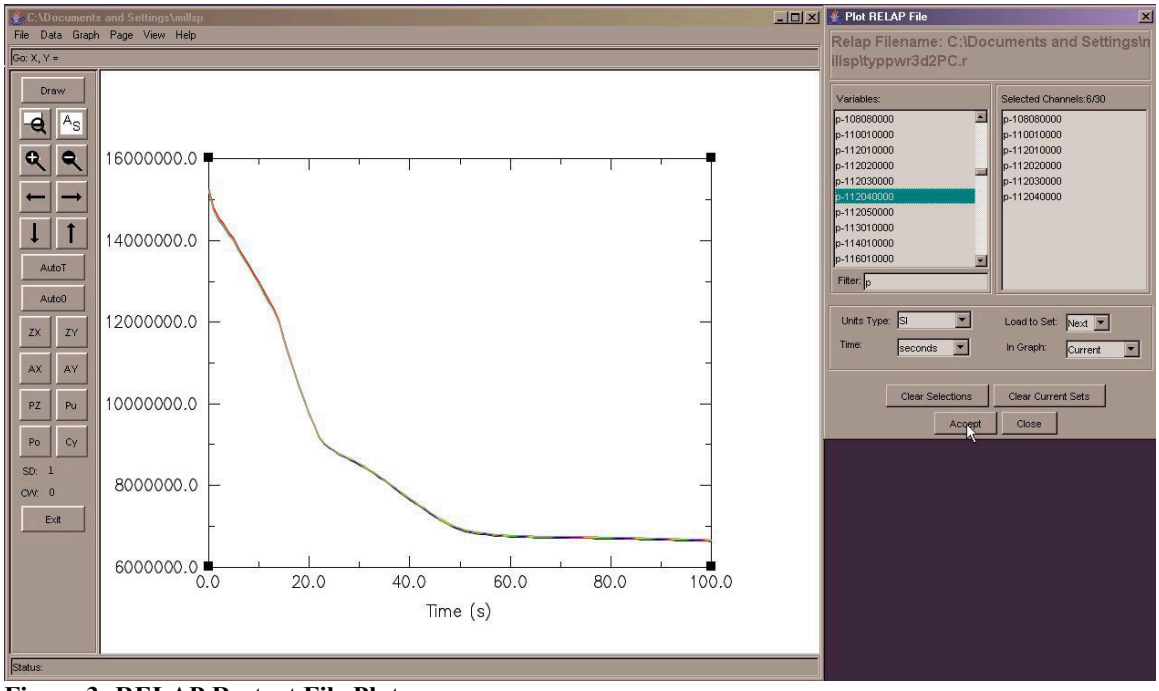

Figure 3: RELAP Restart File Plot