Writing a Client Application for Genesis II

Contributors: Chris Sosa (sosa@virginia.edu)

Last Modified: 06/05/2007

Introduction

The purpose of this White Paper is to discuss the use of the University of Virginia's Genesis II grid platform through a dynamically linked library (DLL). Since the core of the Genesis II platform is in the Java programming language, the purpose of this DLL is to simplify the work of programmers writing applications in native languages such as C, Basic and the like. In general, programmers wishing to write libraries outside of Java may do so with the use of the Java Native Interface (JNI). The JNI, provided by Sun [3], allows native code to interact with Java. However, we feel that its use is an unnecessary burden for most application developers who wish to use the Genesis II platform.

For those programmers who find our DLL insufficient, there are many tutorials on using JNI for making calls into our Genesis II core [1][2][3][4]. These tutorials formed the basis of the DLL's implementation and will be very helpful for a programmer who is unfamiliar with JNI. The rest of this document will concern itself with the use of the Genesis II library.

The C to Genesis Library

In order to capture the majority of use cases of the Genesis II grid platform, our library provides a subset of a POSIX-like interface. We also amend the interface to include additional functionality to login and logout of the Genesis II platform. These functions must be included in order to provide security guarantees of resources on our

grid platform. Unfortunately, the login() command is still interactive and therefore only appropriate for console-like applications. We will be working on an auto-login() feature for the C library.

As discussed previously, the core of this library uses JNI. Each function provided by the interface makes a call into a small wrapper on the Java end that makes the appropriate Genesis II call on behalf of the client. The library, however, hides exceptions raised in the Java code and returns either a failure or success code along with other return values. The JNI calls are made with a negligible amount overhead once the Java Virtual Machine is loaded as all the calls are made in user space as they would be if a client's code was written in Java.

The following page is a diagram of how the DLL interacts with the Grid, Java and the client application.

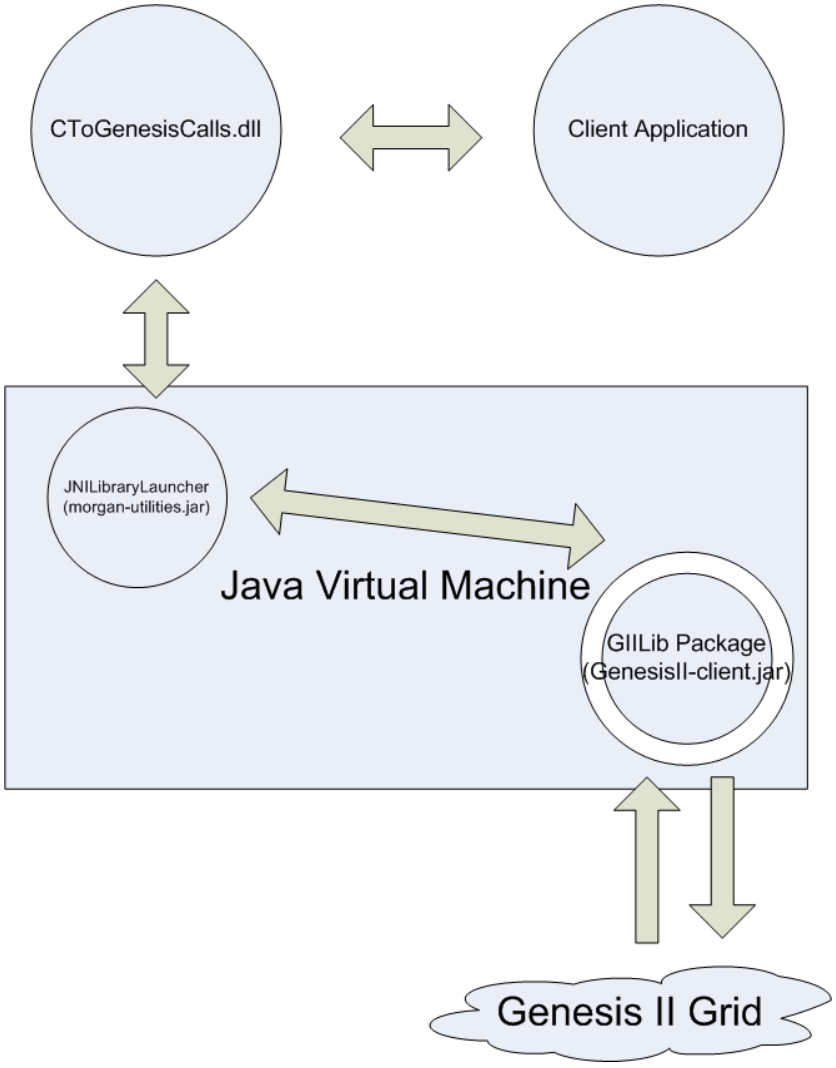

Figure 1: DLL Interaction

The Interface

Our interface provides the following functionality: reading bytes from a Genesis II resource, changing the working directory, getting a directory listing, getting the working directory, copying files to and from Genesis II, moving files within Genesis II, and removing resources from Genesis II. In order to use this DLL, a developer must run the initializeJVM method first. This method takes in the Genesis II install directory as an input parameter. If not run, it is run at the first Genesis II call with a default directory,

however, for many users this default directory will be incorrect. Further versions of Genesis II may de-necessitate the use of this method.

The interface is defined in MakeGenesisIICalls.h. These are the functions that are exported by the DLL. MakeGenesisIICalls.h must be included in any code that makes use of these methods.

Unless otherwise noted, all functions returns an integer value >= 0 if successful and a -1 if there was an error

- **#define GII_FILE_HANDLE int**

  This is the File Handle that is used in Genesis II calls

- **int initializeJavaVM(char * genesis_directory)**

  This method will initialize the Java Virtual Machine with the given Genesis II directory e.g. "C:/Program\ Files/Genesis\ II". This must be called before any other calls to the library are made.

- **int genesisII_login(char * keystore_path, char * password)**

  This method performs a text-based login using the keystore and password given. By default, the Genesis II installation provides a keystore in ~/security/keys.pfx. This method should be called before most commands are run as they will fail otherwise.

- **int genesisII_logout()**

This method will log the current user out of Genesis II. It will log the user out under any circumstance without error (regardless of successful login).

- **int genesisII_directory_listing(char *** listing)**

This method takes in a pointer to an un-initialized array of strings and returns an array of strings that represent the directory listing of the current directory.

- **int genesisII_change_directory(char* new_directory)**

This method changes the current directory to the directory given.

- **char* genesisII_get_working_directory()**

This method returns a string representing the current working directory. If there is an error, a NULL is returned in its place.

- **int genesisII_make_directory(char * new_directory)**

This method will create the directory given.

int genesisII_remove(char * path, int recursive, int force)

This method will remove the target path. Forcing (giving a 1 value to force) may leave dangling pointers e.g. forcing the deletion of a directory. It is recommended that deletion of a directory be called with force only when it is also done recursively.

- **int genesisII_copy(char *src, char* dst, int src_local, int dst_local)**

This method will copy a file from the src to dst and allows copying from/to local when the two booleans are set accordingly.

- **int genesisII_move(char *src, char * dst)**

This method moves a file from src to dst where both paths reside within GenesisII. If a failure occurs, either nothing was performed or a copy was made. In no instance will a file be removed without a copy.

- **GII_FILE_HANDLE genesisII_open(char * target, int create, int read, int write)**

This method returns the Genesis II file handle for the target specified. It opens the file with the three Booleans specified: create (whether or not to create it – note that this will only create if it does not already exist), read (to be able to read), and write (to be able to write). If the user does not have permissions for any property of the resource that he/she is requesting, the entire operation will abort.

- **int genesisII_read(GII_FILE_HANDLE file, char * data, int offset, int length)**

Tries to read up to length bytes starting at the offset for the specified file located at the target. The data parameter must be initialized to hold up to length bytes. Returns the number of bytes read or -1 if an error occurred.

- **int genesisII_write(GII_FILE_HANDLE file, char* data, int offset, int length)**

This method writes the data array to the file at the given offset. The length field should be the size of the data array.

- **int genesisII_close(GII_FILE_HANDLE file)**

This method closes a file. Data is only committed when a file is properly closed. Once closed, the file handle has no value.

Installation Instructions

In order to use this DLL, you must include the header file MakeGenesisIICalls.h provided by the library in any code that is written. This header file contains all the methods that are exported by the DLL. You must also link the application with the .lib /.dll file of the library. In addition, you must append your path (the environment variable) with the directories that contain JNI DLL (jvm.dll) and the CToGenesisCalls.dll. The JNI dll is located in your JRE_directory/bin/client. Note, if you have the JDK, the jre directory is located at JDK_directory/jre.

We provide a step-by-step tutorial for doing this using Visual Studio 2005 and Windows XP:

1. The DLL and LIB and header files are located in your Genesis_II_install_directory/jni-lib.
2. Go to your Environment Variables settings (located in Control Panel->System) and add your JRE_directory/bin/client and the directory from 1.
3. In your application code (in VS 2005), go to your project properties and under C/C++->General, add the additional directory of the location of jni library.

4. Additionally, in project properties, under Linker->General insert the additional library directory of the location of jni library. Move on to Linker->Input and add CToGenesisCalls.lib to your additional dependencies.
5. Finally, make sure that you include <MakeGenesisIICalls.h> in any source file where you need to make Genesis II calls.

Issues

There needs to be a way of hiding many of the exceptions output from the Java end of the program. In terms of efficiency, the use of the Launcher-like wrapper on the Java end to handle calls from the library is causing the application to substantially slow down. Work should be done in obviating the use of such a launcher and loading the libraries statically at the beginning of the application. This work would both improve the efficiency of the library in addition to the efficiency of our grid prompt.

Future Work

More work should needs to be done to obviate the use of any form of launcher. The launcher (when outside of a development environment) increases the amount of time it takes for most calls by a factor of two. We also need to increase the amount of functionality provided by the library. Currently, many functions have been left out of the library. Further work in adding functionality this library will make it a more valuable tool for the intended audience.

Experiences and Observations Using JNI (notes)

- In order to create a library with the JNI, you must include the header files provided in the JDK/include and JDK/include/win32 (or the like if using Linux).
- The jvm.lib file of the JDK must be linked as well as the environment path variable appended with the location of the jvm.dll.
- Using Visual Studio 2005, you must follow a very similar procedure as you did for the Genesis DLL.
- Do not use gcc unless you have to in windows. CL (VS) is a much better options as the java libraries for windows are compatible with the way CL looks for exported functions from A DLL. If you get errors in linking where the linker cannot find specified exported functions, type the string command (bash) in jvm.lib to see what methods are exported and compare to the errors. If they don't match, then you need to change your version of your compiler. If they do, then you did not add the .lib correctly.
- If you get an error that it can't find the .dll (at runtime) then the DLL is not in your path.
- You must restart VS (and command prompt etc) in order to see changes to your environment variables. Environment variables are seen as they were when an application was started and not updated.
- While coding the Java wrapper, never hide exceptions on the Java side or else these errors may manifest themselves in different way and therefore present themselves as different errors.

- Infinite loops and the like on the Java side will kill the JVM and result in GenesisII: "can't find" errors. Since the same JVM is used for all calls (for efficiency sake), this will propagate to all other calls.
- You have to be logged in to do most operations. During testing, login()'s should be called before every set of operations (*Genesis II specific*).
- JVM destruction is iffy at best. I would recommend not using under most cases. Issues with threads being created / destroyed will keep the JVM open. We will need to implement a function on the Java side that will kill all Genesis client threads before this will function properly (this method waits for all threads to complete before destroying the JVM). If Genesis has hanging threads, this call will hang indefinitely.
- You cannot include the \bin folder in your classpath for Genesis II when initializing the JVM. Otherwise, the launcher will fail to load the appropriate classes and you will receive many errors of the type: "cannot find specified class" in Java.
- Forcing the removal of a directory removes some data but leaves hanging data.
- It is difficult to include all libraries for Genesis II and run the client or container of Genesis II from the command line. This is why we have a Launcher class. Use grid.bat and runContainer.bat as examples and study the code in Launcher.java to see how Launcher loads most of the libraries and invokes the main classes. For the library, we developed an alternate Launcher that functions slightly different than the former and is intended for specific use with the C Library.

References

[1] Stricker, Scott. *Java Programming with JNI*.

http://www.ibm.com/developerworks/edu/j-dw-javajni-i.html

[2] Stearns, Beth. *Integrating Native Code and Java Programs*

. http://www.science.uva.nl/ict/ossdocs/java/tutorial/native1.1/implementing/index.html

[3] Sun. *Chapter 5: JNI Technology*.

http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jni.html

[4] Havey, Michael. *Calling Java from C*. http://java.sys-con.com/read/45840.htm