Genesis II Development Tutorial

Duane Merrill 2/6/2007

Overview

This document describes the tools and configuration suggested for Genesis II development, as well as a walk-though of the creation of a simple Genesis II grid service.

At present time, Windows and Linux are our preferred development platforms (although the required tools below may run on many others). The reason for this is that we rely on native JNI C/C++ code for some Genesis II functionality that is not supplied by the JDK, and we currently only have ports of this JNI code for Windows and Linux. Feel free to port the JNI code to other platforms, if you wish!

Requirements

There are a handful of required and recommended tools for configuring a Genesis II development environment:

- Java 5 SDK: http://java.sun.com

  To my knowledge we have not tested GII on Java 6, although we should have a discussion on the merits of doing so. You will probably want to make sure that your PATH environment variable includes the JDK/bin directory. Even if your Linux distro has Java 5 in /usr/bin, you will need to install your own copy of the SDK (see the Configuration section below for more information). And it should go without saying that you don't want to use whatever Java ships with Windows.

- Eclipse IDE: http://www.eclipse.org

  Not required, but highly recommended as a development environment. Among many other things, Eclipse provides built-in CVS integration (with GUI features for diff'ing workspaces/files/etc.), dynamic code injection while debugging (no need to restart your Jetty container), call-hierarchy searching, auto-formatting, etc.

- Apache Ant: http://ant.apache.org

  We use Apache Ant for driving our build processes; Ant is a (portable) XML driven build tool without some of the headaches of make/gmake. Although Eclipse is good for running your code directly from the .class files it auto-magically compiles (for quick incremental testing), we use Ant to take care of several code-generation pre-processing steps, native JNI compilation (if necessary), jar-creation post-processing steps, etc. (Technically Eclipse ships with Ant support, allowing you to run Ant targets from its Outline view.) You'll definitely need the standalone version of Ant if you are planning on running GII containers or tools from a command-line shell (our runContainer.[sh|bat] and grid.[sh|bat] load our bytecode from .jar files, which are only created/updated during a

full Ant build) or without Eclipse. Your Linux distro may have an installation of Ant in /usr/bin that may work fine; you will want to obtain a copy if you are developing in Windows (you will probably want to make sure that your PATH environment contains the ant/bin directory).

- CVS: http://www.tortoisecvs.org, http://www.cygwin.com

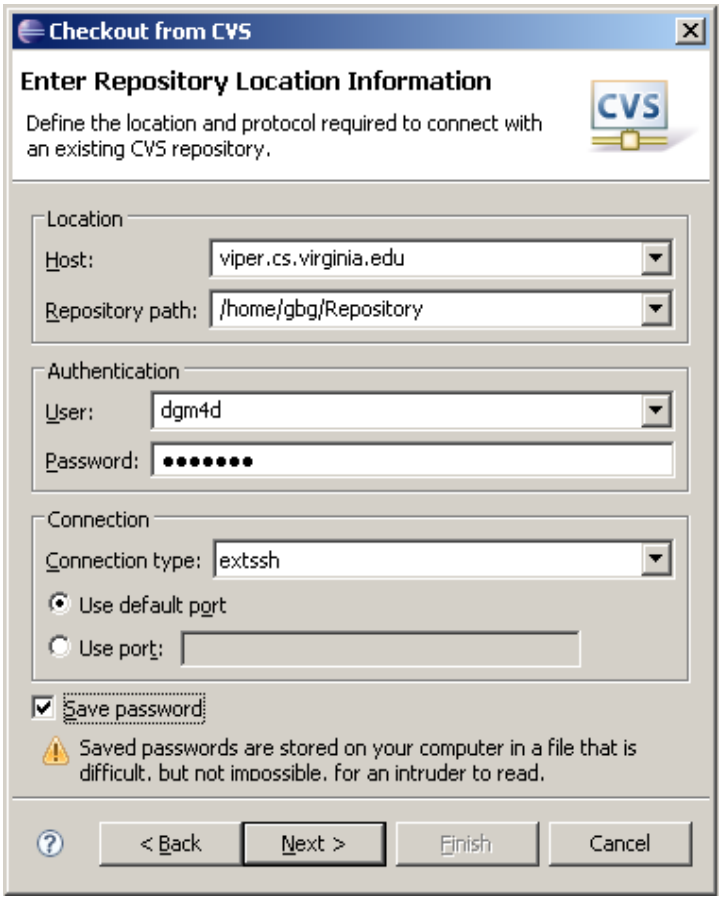
  We presently use CVS for software version management. Your Linux distro should already include an installation of the CVS client tools in /usr/bin. If you are planning on developing in Windows without Eclipse, you can obtain reasonable CVS tools from TortoiseCVS or Cygwin.

Configuration

The first step is obtaining a copy of the Genesis II source. This may be accomplished by downloaded a Genesis II distribution from the VCGR website (http://vcgr.cs.virginia.edu/genesisII, follow the download and installation instructions) or by doing a CVS checkout from the Genesis II module ("`cvs checkout GenesisII`") with the following settings:

```
CVSROOT=:ext:<username>@viper.cs.virginia.edu:/home/gbg/Repository
CVS_RSH=ssh
```

If you are developing in Eclipse, you will probably want to create a new CVS project (using extssh):



In the prompts that follow, Eclipse will ask you to enter or browse for the GenesisII module (either method is fine). You will probably also want to select the "Check out as a project in the workspace" option and to check out the HEAD tag.

After the source distribution has been obtained, you must now configure your JRE to allow the BouncyCastle JCE security provider to run at full strength (no restrictions on key length or algorithm selection). This is can be done by copying the Unlimited Strength Java Cryptography Extension (JCE) Policy Files (the two .jar files found in `<GII-BASE>/ext/jce`) to the JDK's jre\lib\security directory.

Building and Running Genesis II:

The two Ant targets that you will most likely be using are the `build, clean`, and `wipestate` targets. The `build` target loosely performs the following activities:

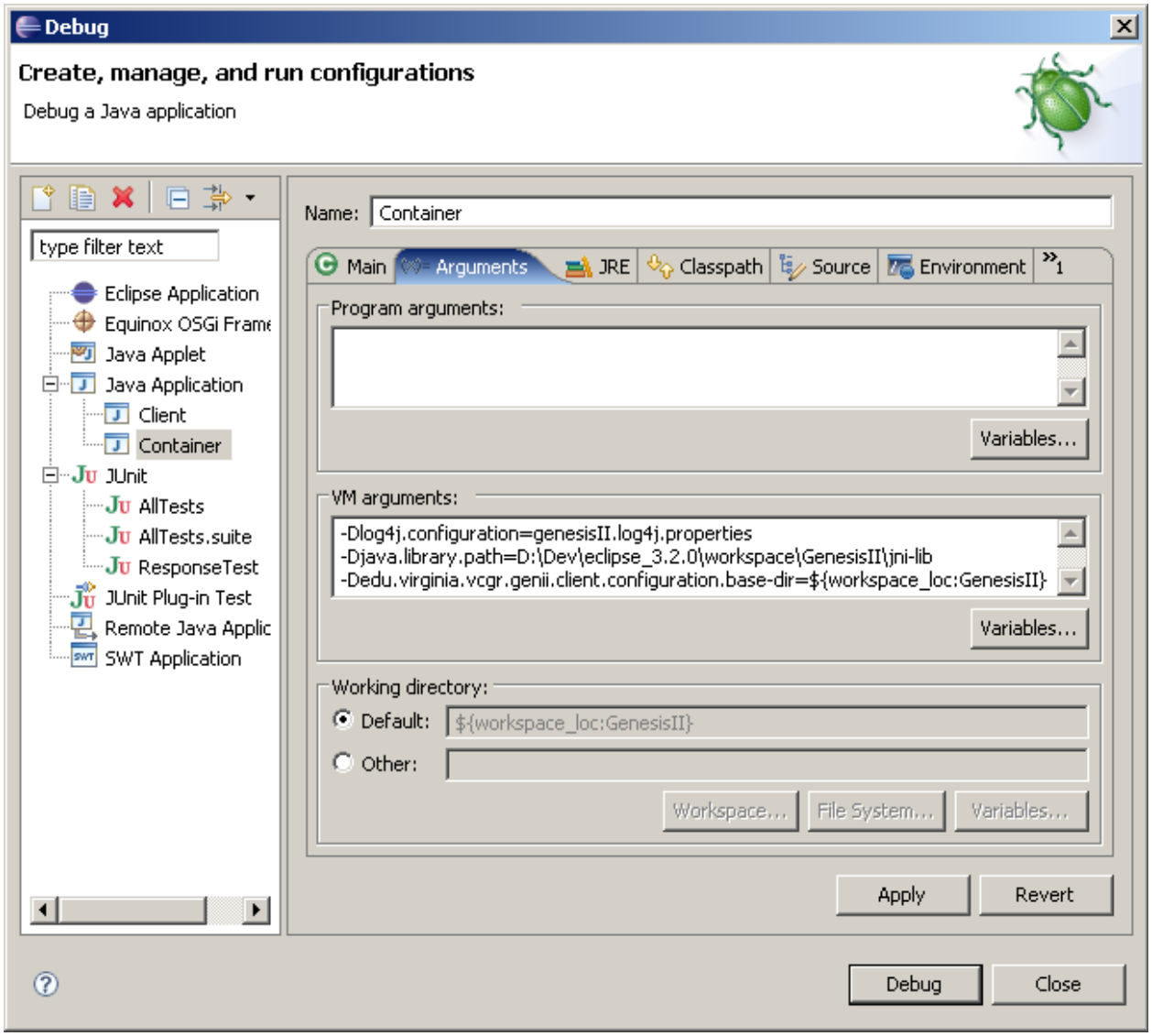
- Creates directories for generated sources
- Normalizes our extended form of WSDL (GWSDL) into proper service WSDL
- Runs Axis WSDL2Java stub generation on our service WSDL to create the Java stub classes used by client tools and the data-structure classes for representing operation parameters within both client and server code.
- Performs some housekeeping on the .wsdd service deployment descriptors so that the Axis web application can find the proper classes to invoke methods upon
- Compiles both the generated and pre-existing sources
- Archives the bytecode classfiles into their respective utility, client, and container .jar files.
- Creates shell scripts for starting the container / command-line tools in the base GenesisII directory

If you are developing (and running your GenesisII container/tools) within Eclipse, you do not have to run a full build prior to every update. As long as you do not modify any WSDL (which would require re-generation), the Eclipse IDE will automatically incrementally compile any updates you make to source (and even reflect those changes in any live debugging sessions).

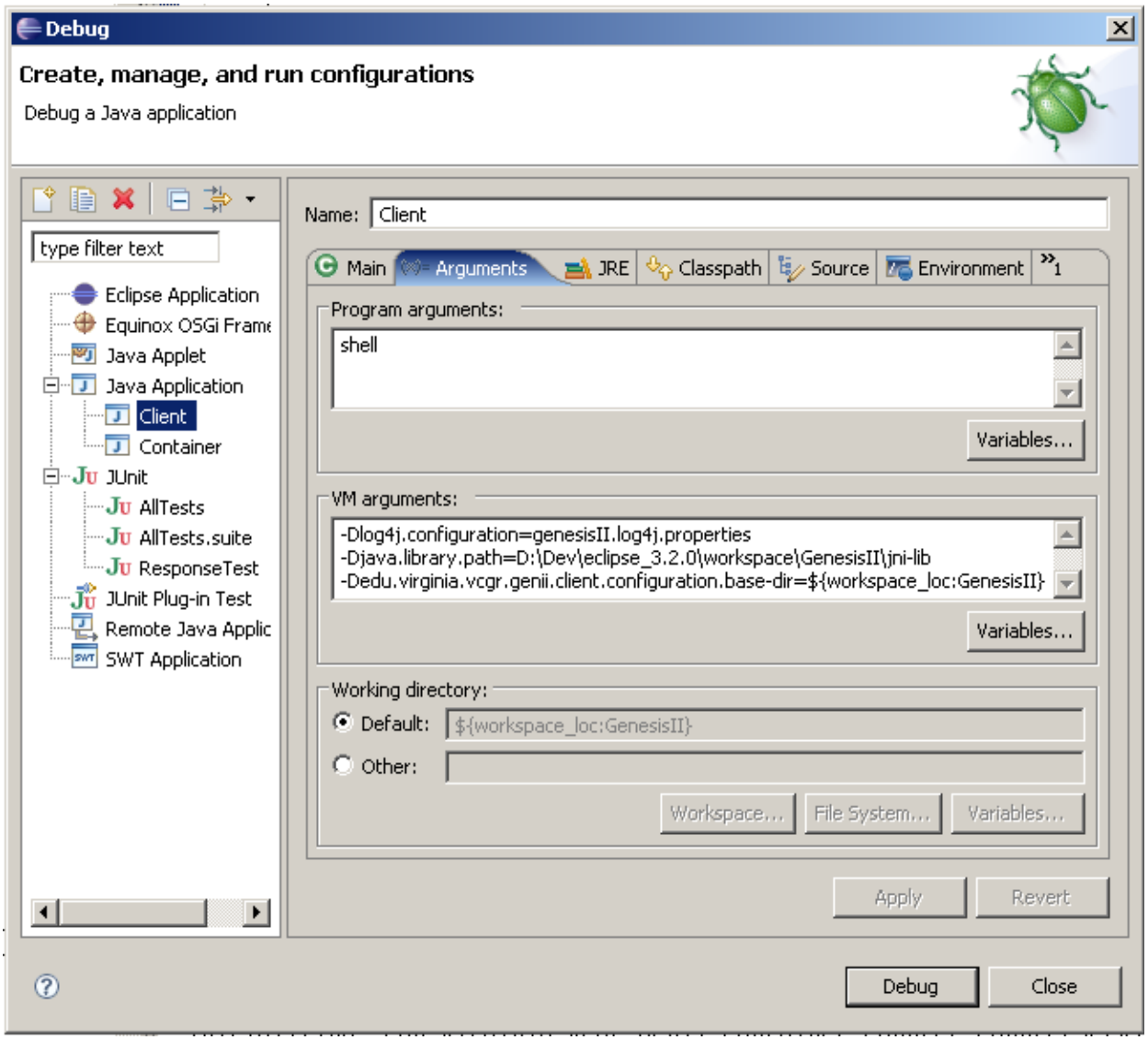
When running, the GenesisII container (by default) creates and manages its database state in a .genesisII directory ("$HOME/.genesisII" for Linux, "C:\Documents and Settings\<username>\.genesisII" for Windows). Additionally, the command-line client tool(s) also keep their session state (transient login credentials, current working directory, etc.) in this directory. This `.genesisII` directory is destroyed when the Ant `wipestate` target is called (which is also a dependency of the `clean` target). You may also find yourself manually deleting this directory in order to restart your container in a "clean state" without having to clean+build the entire source.

Once the source has been built, you can start the container using the `runContainer.[sh|bat]` script in the base GenesisII directory. This will start a container listening (by default) for HTTPS connections on port 18080. (The port can be adjusted within the `<GII-BASE>/configuration/server-config.xml` configuration file.) You can use the command-line (CLI) client tool by running the `grid.[sh|bat]` script in the base GenesisII directory. The CLI tool has a bunch of subtools (including bootstrap), which may be viewed by running `grid help`.

For running the container within Eclipse, you will need to configure a debug/run profile. You can do this by right-clicking on the `edu.virginia.vcgr.genii.container.Container` class (in the Package or Navigator view) and selecting the "Debug As…" option. In the Main tab, make sure that your configuration has the `edu.virginia.vcgr.genii.container.Container` class set as the Main Class. Add the following to the VM arguments pane in the Arguments tab:



Now you should be able to run the Genesis II container simply by selecting it from the drop-down list of the debug launcher toolbar button. To configure a CLI tool launch configuration, perform the previous steps using the following analogous configuration:



This will create a launch configuration that starts the CLI tool in its simple "shell" mode, allowing you to use its full complement of subtools. (Selecting the shell argument saves you from having to create/edit limitless configurations for each variant of each subtool.)

Bootstrapping a Grid:

Although many OGSA resources (and their containers) may comprise a grid, it is likely that you will want to establish a hierarchical directory namespace to arrange them in. The bootstrap process does just that: it establishes a root RNS directory on a particular Genesis II container, creates some reasonable subdirectories (`/factories`, `/containers`, `/bes-containers`, `/queues`, `/collections`, etc.), and links some of the per-container resources into the namespace as well (e.g., `/bes-containers/BootstrapBES`, `/containers/BootstrapContainer`, etc.).

To bootstrap a container in this fashion, we simply use the `script` subtool's scripting functionality to run the `<GII-BASE>/bootstrap.xml` script file. (Which you can tailor to alter the bootstrap process as you wish.) An example of bootstrapping is shown below (run from the `grid` tool in simple "shell" mode):

```
vcgr:$>script bootstrap.xml
Please select a certificate to load:
   [0]:  CN=skynet1.cs.virginia.edu, OU=VCGR, O=UVA, L=Charlottesville,
ST=Virginia, C=US
   [1]:  CN=Duane George Merrill III 15, EMAILADDRESS=dgm4d@virginia.edu, OU=UVA
Standard PKI User, O=University of Virginia, C=US
   [x]:  Cancel

Selection?  0
Creating Root of RNS space.
- Cannot confirm trusted identity for
https://moogle.cs.virginia.edu:18080/axis/services/RNSPortType: EPR for
https://moogle.cs.virginia.edu:18080/axis/services/RNSPortType does not contain a
certificate chain.
Storing configuration to "context.xml".
- Cannot confirm trusted identity for
https://moogle.cs.virginia.edu:18080/axis/services/VCGRContainerPortType: EPR for
https://moogle.cs.virginia.edu:18080/axis/services/VCGRContainerPortType does not
contain a certificate chain.
Setting permissions for bootstrap directories and serviecs
Forming general directory structure.
Adding groups
Adding users
Adding machines
Setting security on containers/services
Setting security on container BootstrapContainer
Setting security on ExportedFilePortType in container BootstrapContainer
Setting security on BESPortType in container BootstrapContainer
Setting security on BESActivityPortType in container BootstrapContainer
Setting security on CounterPortType in container BootstrapContainer
Setting security on RNSPortType in container BootstrapContainer
Setting security on ExportedRootPortType in container BootstrapContainer
Setting security on BasicSchedulerPortType in container BootstrapContainer
Setting security on RandomByteIOPortType in container BootstrapContainer
Setting security on ExportedDirPortType in container BootstrapContainer
Setting security on VCGRContainerPortType in container BootstrapContainer
Setting security on StreamableByteIOPortType in container BootstrapContainer
Setting security on GeniiSubscriptionPortType in container BootstrapContainer
Creating BES resource for BootstrapContainer.
Linking bes BootstrapContainer into scheduler
```

The first thing the script does is to run "login" to prompt you to select the necessary credentials to interact with your brand new container. The static, per-container resources (e.g., the RNS service that allows you to create RNS resources) are initially established with access-control lists configured to only allow the identity corresponding to the container's certificate. (The default container identity is "skynet1.cs.virginia.edu", whose certificate and keypair are contained within the `<GII-BASE>/security/skynet.cer` certificate file and `<GII-BASE>/security/keys.pfx` keystore file, respectively. You can change the default certificate by un-commenting and editing the `genii.security.authz.bootstrapOwnerCertPath` property in the `<GII-BASE>/configuration/server-config.xml` configuration file.

Other things to note in the bootstrap script are the `create-rns-root` and `ln` commands to create the root context resource and link in the container's Container resource: they include the hostname and port number of your container. (You may want to modify them if you change the port your container starts up on.) Additionally, the `create-rns-root` command outputs a `context.xml` file (to the local current working directory) that contains the EPR of the root context /. (By sharing this EPR document file, others can connect to your grid or mount your namespace into their grid(s).)