Modeling Heterogeneous Multiprocessors with the Cmpware CMP-DK

Cmpware, Inc.

Introduction

The *Cmpware Configurable Multiprocessor Development Kit (CMP-DK)* is based around fast simulation models for multiple processors. The goal of this toolkit is to provide support across a wide variety of multiprocessing platforms. While many multiprocessors may opt for a single type of processor, many may include a mix of processing elements of various types. These multiprocessors containing more than one type of processing node are typically referred to as *heterogeneous* multiprocessors.

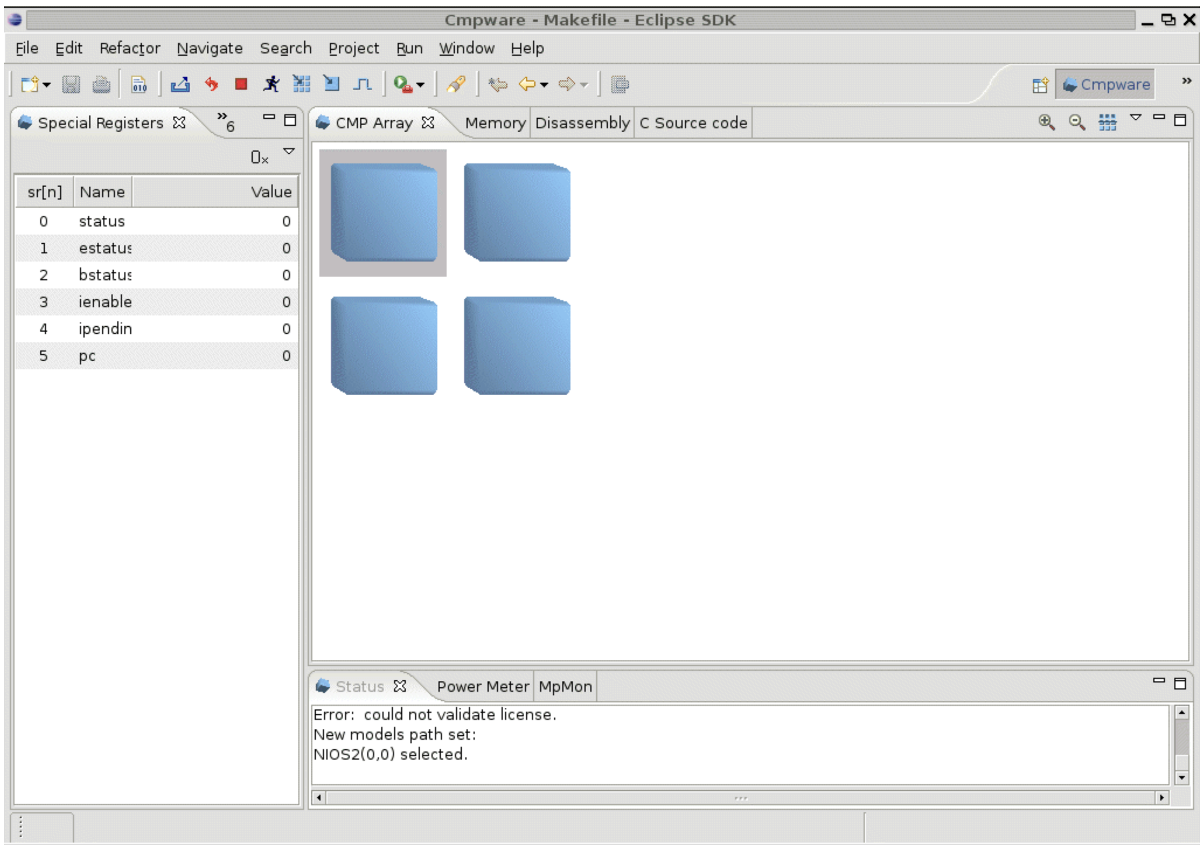
Heterogeneous multiprocessors increase the complexity of a multiprocessor system by requiring that more than one type of processing node be designed. Also significantly, each node type will require a set of tools including a simulation model. Such systems tend to require highly specialized software support and have typically required fully custom software solutions. In spite of their increased hardware and software complexity, heterogeneous multiprocessors can provide dramatically higher levels of performance often while simultaneously lowering overall power consumption. Depending on the requirements of the system, the added complexity of a heterogeneous system may be worth the additional effort.

While heterogeneous multiprocessors provide a software tools challenge, this is an area where the *Cmpware CMP-DK* excels. Because the *Cmpware* models are built with a standard interface used to interact with other models, heterogeneous multiprocessors present no particular modeling problem. This document describes the process used to build a heterogeneous multiprocessor consisting of two different types of processing nodes. In this example, existing models for a *MIPS32* processor and an *Sparc8* will be used.

Of course, other processor configurations, including custom processors and hard wired logic can be modeled using these same techniques. This processor arrangement is chosen primarily for simplicity. The same techniques can be used for arbitrarily large and diverse heterogeneous multiprocessors.

Defining the Multiprocessor

This entire example is performed on a *Dell Precision 340* PC running *Debian Linux* version 3.1, *Eclipse* 3.1 and the *Cmpware CMP-DK* version 1.3.6. The results should be the same for other similar configurations.


Cmpware - Makefile - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Cmpware
Special Registers
CMP Array
Memory
Disassembly
C Source code
Status
Power Meter
MpMon

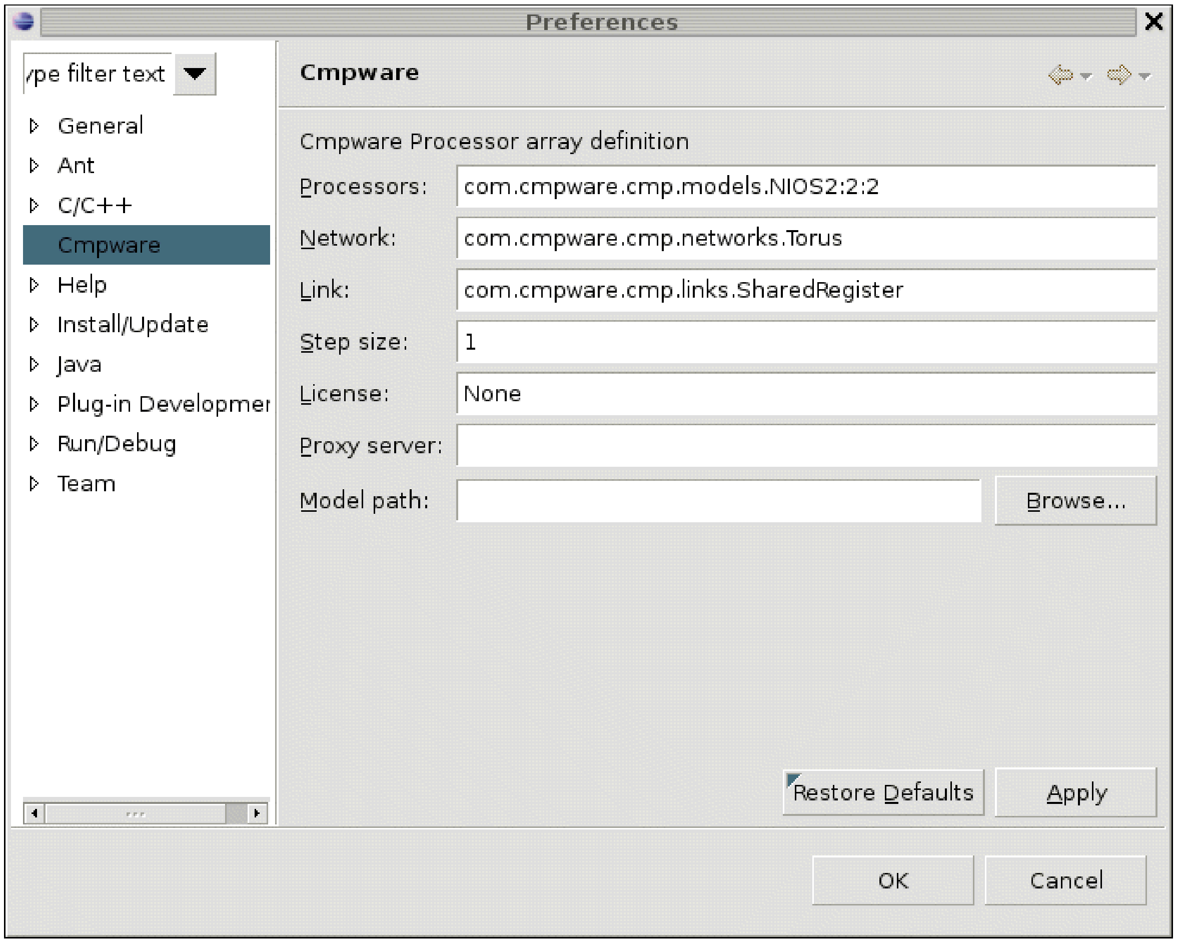
sr[n]	Name	Value
0	status	0
1	estatus	0
2	bstatus	0
3	ienable	0
4	ipendin	0
5	pc	0

```
Error:  could not validate license.
New models path set:
NIOS2(0,0) selected.
```

Figure1: The *Cmpware CMP-DK* default screen.

When the *Cmpware CMP-DK* is loaded, the default configuration, as shown in Figure 1, is a 2 x 2 array of *Altera NIOS2* processors. To change this default, the Cmpware *Preferences Page* must be brought up. The Preferences Page contains various entries used to configure the *Cmpware CMP-DK*. Most prominently, this is where the multiprocessor is defined.

The *Cmpware* Preferences Page is brought up by selecting the menu items **Windows --> Preferences** and selecting the **Cmpware** list item. This brings up the dialog box as shown in Figure 2.



Cmpware Processor array definition

Processors: com.cmpware.cmp.models.NIOS2:2:2

Network: com.cmpware.cmp.networks.Torus

Link: com.cmpware.cmp.links.SharedRegister

Step size: 1

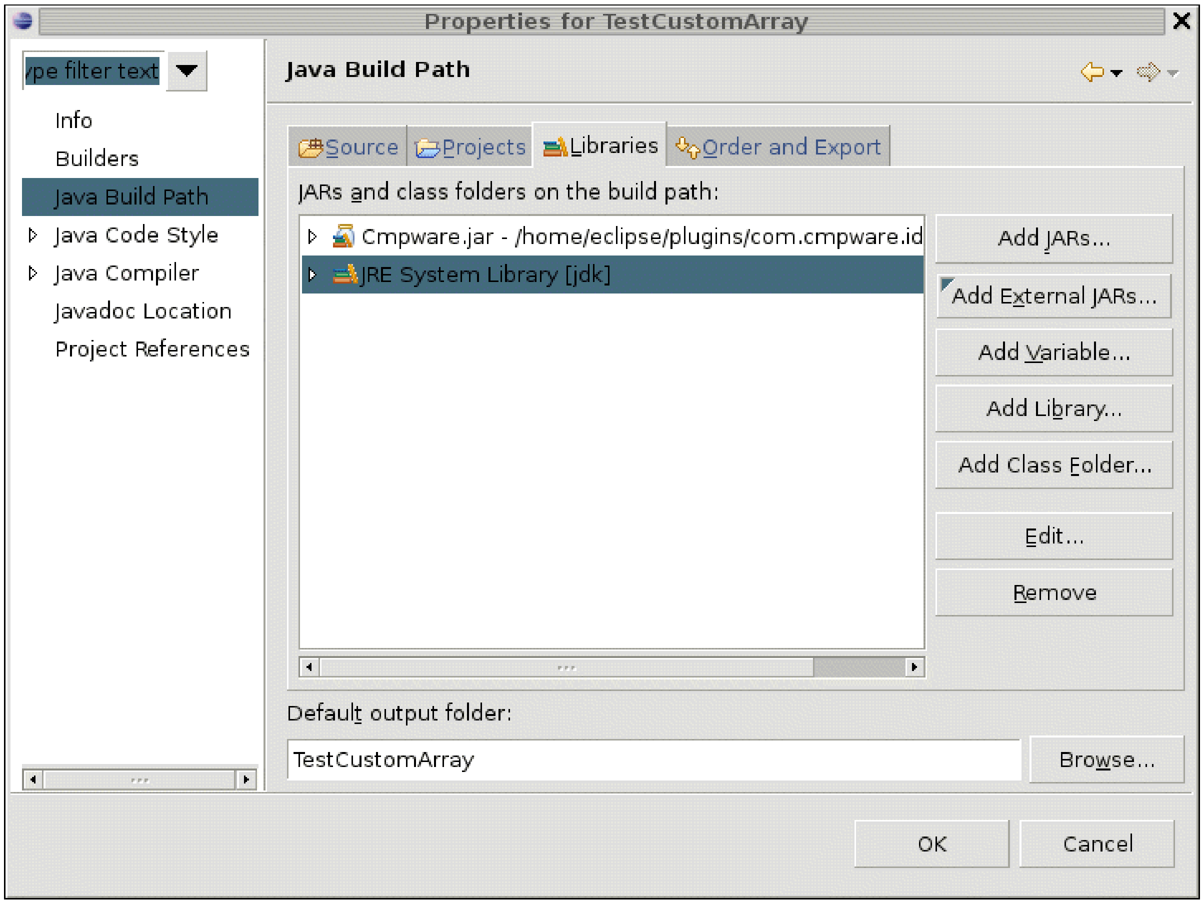
License: None

Proxy server:

Model path:

Figure 2: The default *Cmpware* preferences.

The defaults on this page indicate that the processor array is made up of a homogeneous 2 x 2 array of *Altera NIOS2* processors as modeled by the class **`com.cmpware.cmp.models.NIOS2`**. The suffix "**`:2:2`**" indicates that the array has two rows and two columnsof processing nodes, respectively. Similarly, the *Network* field indicates that the default network interconnection is a Torus as defined by the **`com.cmpware.cmp.networks.Torus`** network model. This Torus model uses the Shared Register link as defined in the **`com.cmpware.cmp.networks.SharedRegister`** link model. These three fields are used to define the three classes which in turn define the multiprocessor model. While the default *Processor* description defines a homogeneous array heterogeneous arrays are also possible. These are described by using a class of the type **`CustomArray`** in place of the single **`Processor`** model. The **`CustomArray`** class is very simple, and defines a single method which returns a 2D array of strings. These strings give the fully qualified name for the **`Processor`** classes which make up the array. The array indicies correspond directly to the coordinates of the processor in the multiprocessor array. That is, the string at element `[0][0]` names the **`Processor`** class corresponding to the processor at *(0,0)* in the multiprocessor array.


Properties for TestCustomArray

Java Build Path

JARs and class folders on the build path:

- Cmpware.jar - /home/eclipse/plugins/com.cmpware.id
- JRE System Library [jdk]

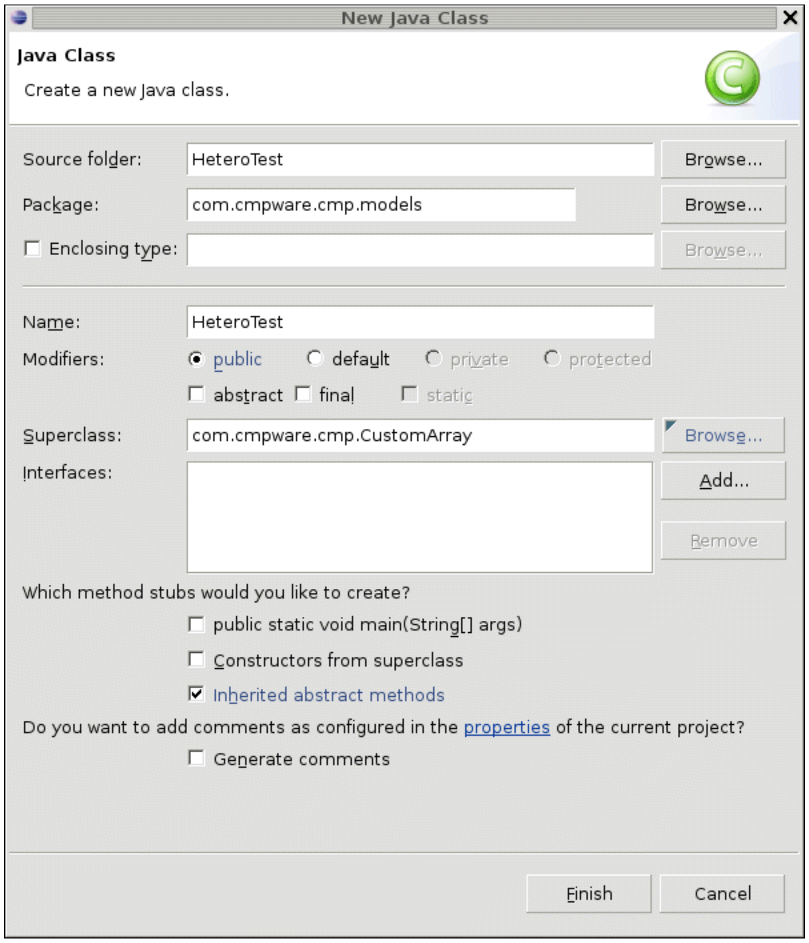
Default output folder:

TestCustomArray

Figure 3: The Java Build Path.

To create this class, a standard Java source code file must be typed in and compiled, then the resulting *class* file used by the *Preferences* page. This can be done with any Java compiler, but it can most easily be done within the same *Eclipse* IDE used by the *Cmpware CMP-DK*.

First, make a new Java project by selecting the menu items **File --> New --> Project...** This will bring up a dialog box with a list of choices. Select **Java Project** and select the **[Next -->]** button. Type in a Project Name of **HeteroArray**, and select the **[Next -->]** button. Select any other defaults offered, and a new Java project should be ready.

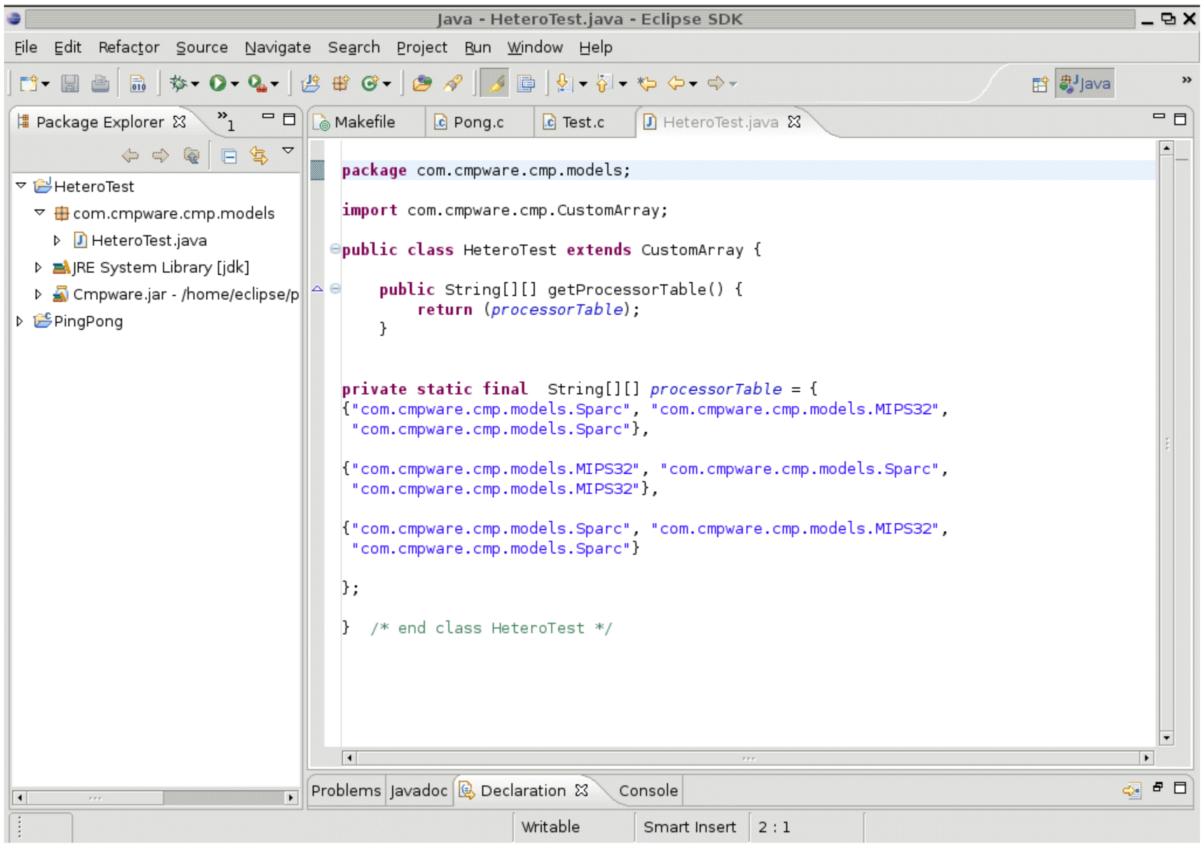


Source folder: HeteroTest
Package: com.cmpware.cmp.models
☐ Enclosing type:
Name: HeteroTest
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: com.cmpware.cmp.CustomArray
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments as configured in the properties of the current project?
☐ Generate comments

Figure 4: The new Java Class creation dialog.

Next we must make the existing Cmpware classes visible in this new project. This is done by selecting the **Project --> Properties** menu items, and selecting the **Java Build Path** item in the list brought up by the menu commands. If it is not already selected, select the **Libraries** tab and select the **[Add External JARs ...]** button. This brings up a file dialog box. Use this to select the *Cmpware.jar* file from the *Eclipse plugin* directory as shown in Figure 3. In this case, the file is in the: */home/eclipse/plugins/com.cmpware.ide_1.3.6* directory, but this will vary from system to system.

At this point, the new Java project is initialized and all of the *Cmpware* classes required to build models are properly referenced. Now a new class file can be entered and compiled. Make a new class file by selecting the menu items **File --> New ...** and select the **Class** menu item in the dialog box brought up by the menu. This brings up the dialog box as shown in Figure 4. The class name should be filled in with the name of the class, in this case, **`HeteroTest`**. The package should be the same as the other models, or **`com.cmpware.cmp.models`**. Finally, the superclass **`com.cmpware.cmp.CustomArray`** should be specified. The other values can remain at their defaults.

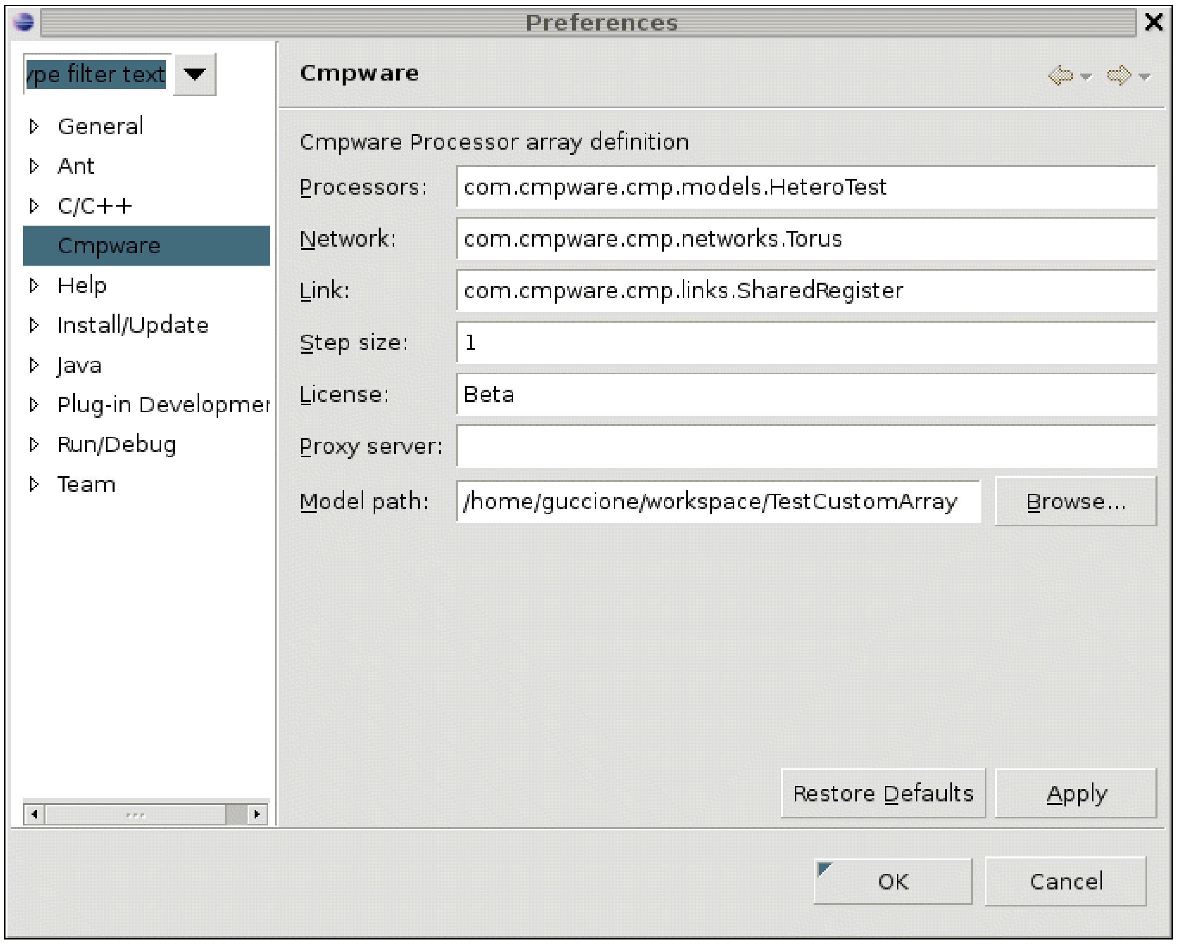


```
package com.cmpware.cmp.models;

import com.cmpware.cmp.CustomArray;

public class HeteroTest extends CustomArray {

    public String[][] getProcessorTable() {
        return (processorTable);
    }


private static final  String[][] processorTable = {
{"com.cmpware.cmp.models.Sparc", "com.cmpware.cmp.models.MIPS32",
 "com.cmpware.cmp.models.Sparc"},

{"com.cmpware.cmp.models.MIPS32", "com.cmpware.cmp.models.Sparc",
 "com.cmpware.cmp.models.MIPS32"},

{"com.cmpware.cmp.models.Sparc", "com.cmpware.cmp.models.MIPS32",
 "com.cmpware.cmp.models.Sparc"}

};

}  /* end class HeteroTest */
```

Figure 5: The new Java Custom Array model.

This produces a new Java source file skeleton suitable for editing. Even the **`getProcessorTable()`** method, as required by the superclass, is provided. All that is required now is to fill in the array values and compile the Java source code file into a compiled Java class file. Figure 5 shows the completed source code file, *HeteroTest.java*. This file is also reproduced in *Appendix A*. Once the source code has been completed, the menu items **Project --> Build All** should produce the class file.

This file defines a custom 3 x 3 array of processors. In this case there are just two different types of processors, a *Sparc* and a *MIPS32*. These are put in a checkerboard pattern, which is intended to make it easy to verify that the configuration produces the desired result.



Cmpware

Cmpware Processor array definition

Processors: com.cmpware.cmp.models.HeteroTest

Network: com.cmpware.cmp.networks.Torus

Link: com.cmpware.cmp.links.SharedRegister

Step size: 1

License: Beta

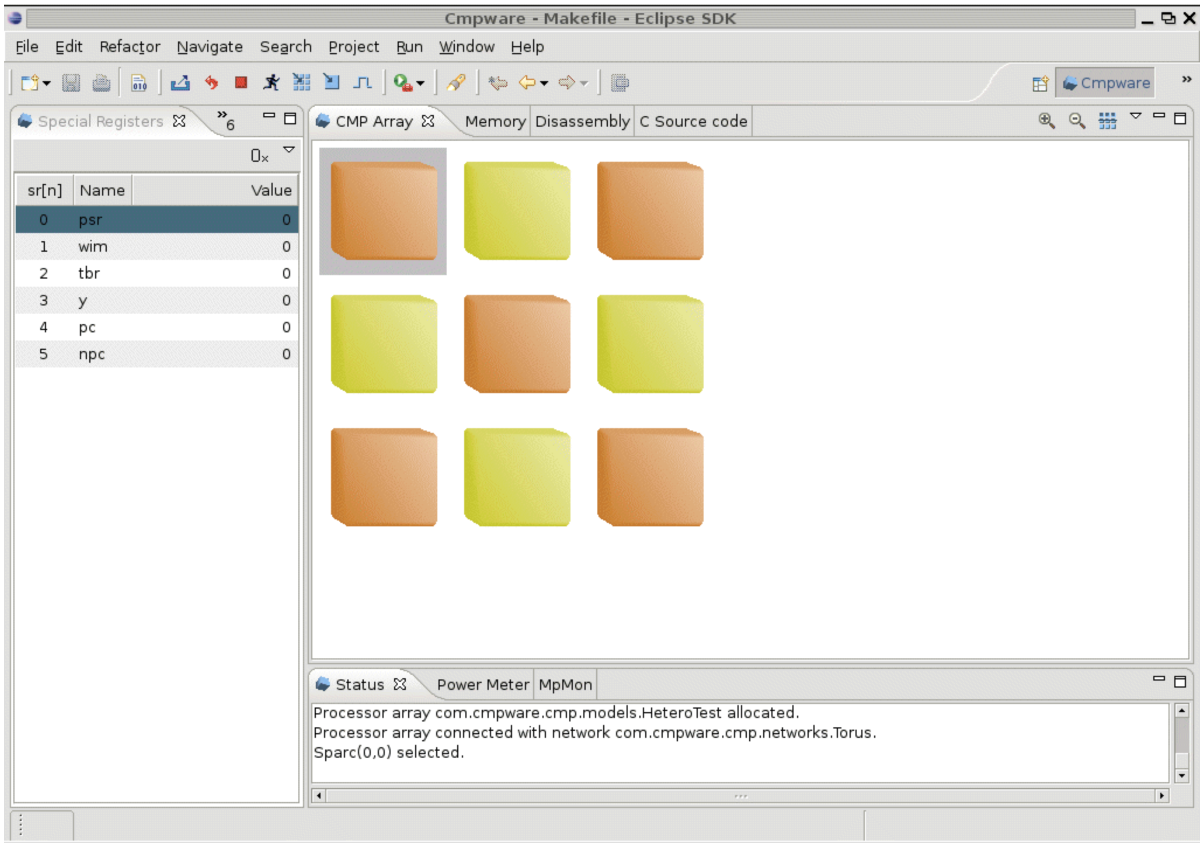
Proxy server:

Model path: /home/guccione/workspace/TestCustomArray

Figure 6: The Cmpware Preference for the HeteroTest.

Simulating the Multiprocessor Model

Once the *HeteroTest* class file has been compiled, it can be used to build a multiprocessor model in the *Cmpware CMP-DK*. First, the Java development Eclipse 'perspective' must exited and the *Cmpware* perspective opened. This is done with the menu commands **Window --> Open perspective --> Other ...**


Cmpware - Makefile - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Cmpware
Special Registers
CMP Array
Memory
Disassembly
C Source code
Status
Power Meter
MpMon

sr[n]	Name	Value
0	psr	0
1	wim	0
2	tbr	0
3	y	0
4	pc	0
5	npc	0

```
Processor array com.cmpware.cmp.models.HeteroTest allocated.
Processor array connected with network com.cmpware.cmp.networks.Torus.
Sparc(0,0) selected.
```

Figure 7: The HeteroTest Sparc / MIPS array.

This brings up a menu of the available installed perspectives in the current Eclipse installation. One should be the *Cmpware* perspective. Select this item and then select the **[Ok]** button. This should being up the standard Cmpware set of windows as in Figure 1. If the *Cmpware* perspective has previously been in use, it may not come up with the default windows, but with the previous window configuration from the last time the *Cmpware CMP-DK* toolkit was used.

From the *Cmpware* perspective, the *Cmpware Preference Page* can be used to instantiate this new multiprocessor configuration described in *HeteroTest*. To do this, first, select the Cmpware Preferences Page via the **Windows --> Preferences ...** menu item. This will bring up the dialog box in Figure 2. The fields in this dialog box will have default values, or the last values set if the default values have been changed.

To configure the model as the 3 x 3 mixed *Sparc* and *MIPS* array defined in *HeteroTest*, two of the fields must be changed. First, the Processor definition should be changed to the full class name of the *HeteroTest* Java class. In this case, the class name **`com.cmpware.cmp.models.HeteroTest`** should be entered in the Processor field. The default Network and Link fields should work, producing a Torus network connected by Shared Register links.

The final change is the bottom field, the *Model Path*. This is used to tell the system that there is an alternate location of simulation models. In this case, the the path is under a user home directory. This directory should be the one containing the '**`com`**' directory leading to the compiled Java class files.

Selecting the **[Apply]** or the **[Ok]** button will cause a new multiprocessor model to be loaded and initialized. Even for relatively large systems, this should take on the order of a second. As Figure 7 indicates, a 3 x 3 array of *Sparc* and *MIPS* processors in a check board pattern is indeed allocated and initialized. Clicking on the processors and viewing the various windows associated with the processors will verify that the processors are indeed *Sparc* and *MIPS* models.

At this point, it should be mentioned that this demonstrates one of the strong points of the *Cmpware CMP-DK*. When a processor is selected, all of the displays update and present information about the selected processor. Note that when, for instance, a *Sparc* processor is selected and the *Special Registers* window viewed, the special registers will be specific to the *Sparc* processor. Similarly, the *Disassembly* window will display *Sparc* disassembly. This permits a variety of processors to be examined with a standard set of views. The *Cmpware CMP-DK* takes care of presenting the data in the proper form.

Conclusions

This document describes the configuration of a heterogeneous multiprocessor model in the *Cmpware CMP-DK*. All that is required is that a small Java file describing the array be entered and compiled. This file is then used to configure the processor array. An arbitrarily large number and type of processing elements is supported by the approach. Once the heterogeneous array is instantiated, standard interactions via the *Cmpware CMP-DK* interface is provided. Note that while the devices within the multiprocessor array may vary, the *Cmpware CMP-DK* supports them uniformly. Each processor can be selected and will drive the various displays in the system. Similarly, instructions and data may be loaded into the processors and execution controlled just as with homogeneous processors.

Troubleshooting

The *Cmpware Configurable Multiprocessor Development Kit (CMP-DK)* is based on the *Eclipse* IDE. While *Eclipse* is a very popular, some versions may occasionally exhibit unstable behavior. These recommendations are general advice and apply to all *Eclipse* plugins, not just the *Cmpware CMP-DK*.

The first thing to try if internal errors or unusual behavior occur is to simply close the perspective with the **Window --> Close Perspective** menu command and re-open it. If this does not solve the problem, closing and re-starting the *Eclipse* IDE will usually clear out any buffers that may be causing these problems. Finally, re-starting *Eclipse* with the "-clean" flag will purge any state data that may be cached and lingering in the filesystem.

Such problems should be rare and these and simple commands should bring everything back up in progressively 'cleaner' states. If none of these techniques work, a last resort is to uninstall the offending plugin and re-install it. After uninstallation, it may even be necessary to manually delete any remaining files in the particular plugin directory. This should not be necessary, but could help in the situation where executable files may be been corrupted. Finally, contact *Cmpware, Inc.* if problems persist.

Appendix A - The HeteroTest.java

```
package com.cmpware.cmp.models;

import  com.cmpware.cmp.CustomArray;


/**
**  This implements a demonstration heterogeneous
**  processor array.
**
**  <p>
**  Copyright (c) 2005 Cmpware, Inc.  All Rights Reserved.
**  <p>
**
**  @author SAG
*/

public class HeteroTest extends CustomArray {


public String[][] getProcessorTable() {
   return (processorArray);
   }  /* end getProcessorTable() */


/** A processor array description */
private static final String processorArray[][] =
{{"com.cmpware.cmp.models.Sparc", "com.cmpware.cmp.models.MIPS32",
  "com.cmpware.cmp.models.Sparc"},
 {"com.cmpware.cmp.models.MIPS32", "com.cmpware.cmp.models.Sparc",
  "com.cmpware.cmp.models.MIPS32"},
 {"com.cmpware.cmp.models.Sparc", "com.cmpware.cmp.models.MIPS32",
  "com.cmpware.cmp.models.Sparc"},
};

}  /* end class HeteroTest */
```