Detailed SystemJ Guide
Version 2.0

By now you've installed the SystemJ Development kit for Eclipse and likely heard about its strengths in designing and programming concurrent and distributed software systems, so what now?

If you just want to jump straight in to development look at the 'Quick-Start' guide for how to use the toolset in Eclipse, or else if you want a more detailed demonstration of SystemJ itself, keep reading. Please note that these guides and tutorials are available in several locations; the website (http://www.systemjtechnology.com/index.php?option=com_content&view=article&id=62&Itemid=73) and in the Eclipse documentation at Help>Help Contents>SystemJ Developer Guide>Development Guides. Additionally there is an FAQ available at http://systemjtechnology.com/index.php?option=com_content&view=article&id=48&Itemid=41. Should you have any further questions on what is needed or should be explained better please send an email to info@systemjtechnology.com, or file a bug report at http://www.systemjtechnology.com/trac/systemjcompiler/newticket and we will get back to you as soon as possible.

As per the installation guide, by this point you are now assumed to have a licensed version of SystemJ installed on your machine. Also, you are likely reading this guide to learn about how to program in SystemJ or you are just looking to brush up on your syntax. While this guide assumes that you have somewhat of a Java background it will proceed in an easy to understand manner; learn by doing essentially, thus providing a bridge between the technical documentation and the application of the language. Using the tools, we will put together the non-networked example that is available as a default template.

To begin with, there is a conventional structure to which SystemJ projects adhere as shown in Figure 1. Firstly, the .sysj file should be kept at the root of project whereas the Java packages can be placed in the src folder. A SystemJ project is organised in this manner primarily for clarity due to the typical project configuration; one or two .sysj files and several Java packages Following this is the naming convention which helps the tools to relate different file types to one another. For instance, the .xml file which defines the network structure used by SC.sysj should be called SC.xml; the tools have been designed to automatically search for files which satisfy this convention when the respective .sysj file is launched. The same convention applies to the .sjio debugging file.

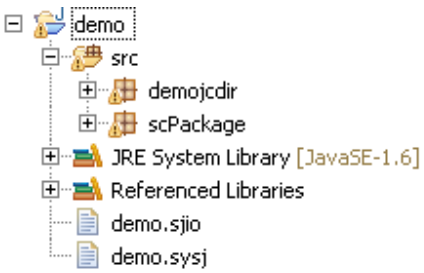

Figure 1: Project Structure

Now that you understand the expected project layout and available file types let us create one using the SystemJ project wizard. Click File > New > Project.. > SystemJ > SystemJ Project. A three page dialog as shown in Figure 2 should be displayed. The first page lets you define the name, location and setup of the project. Give the project a meaningful name and then press Next. On this page you should see displayed the attached .jars that are pivotal to the compilation and execution of the project so do not be tempted to remove them! But, feel free to add more if your Java packages require them. Following this, the final page presents the possible templates you can use. Please untick all of the checkboxes so that a blank project is generated, and click Yes when prompted to switch to the SystemJ Perspective.

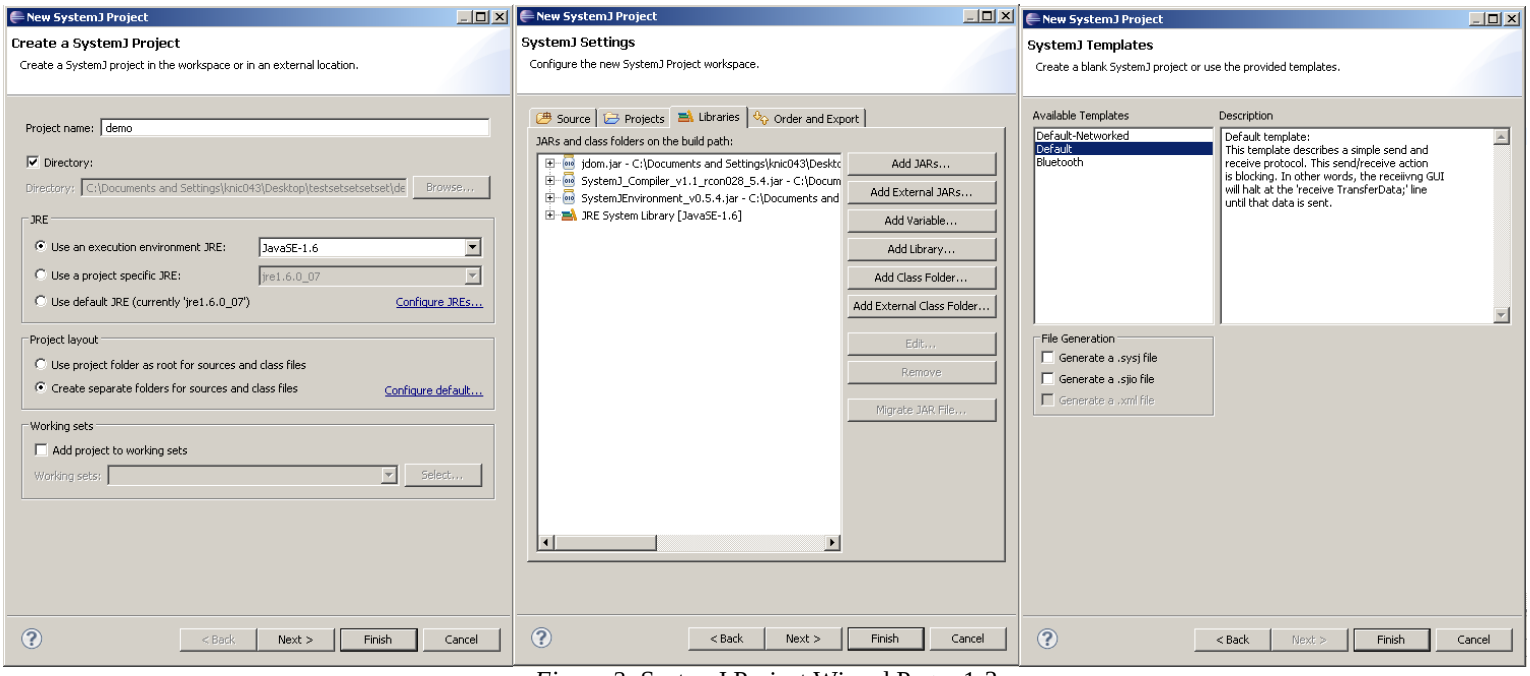

Figure 2: SystemJ Project Wizard Pages 1-3

Press right-click > New > SystemJ File on your new project to open up the SystemJ file wizard as shown in Figure 3. If the name of your project has not been auto completed, either type it into the text field or click browse and navigate for it. Once the appropriate location has been found click Next. As with the project wizard there are several templates available but we wish to have a blank .sysj file so select 'Generate blank .sysj file.' option and then click Finish. The .xml and .sjio wizards follow the same approach but they will be examined at a later time. Once generated the new file will take focus.

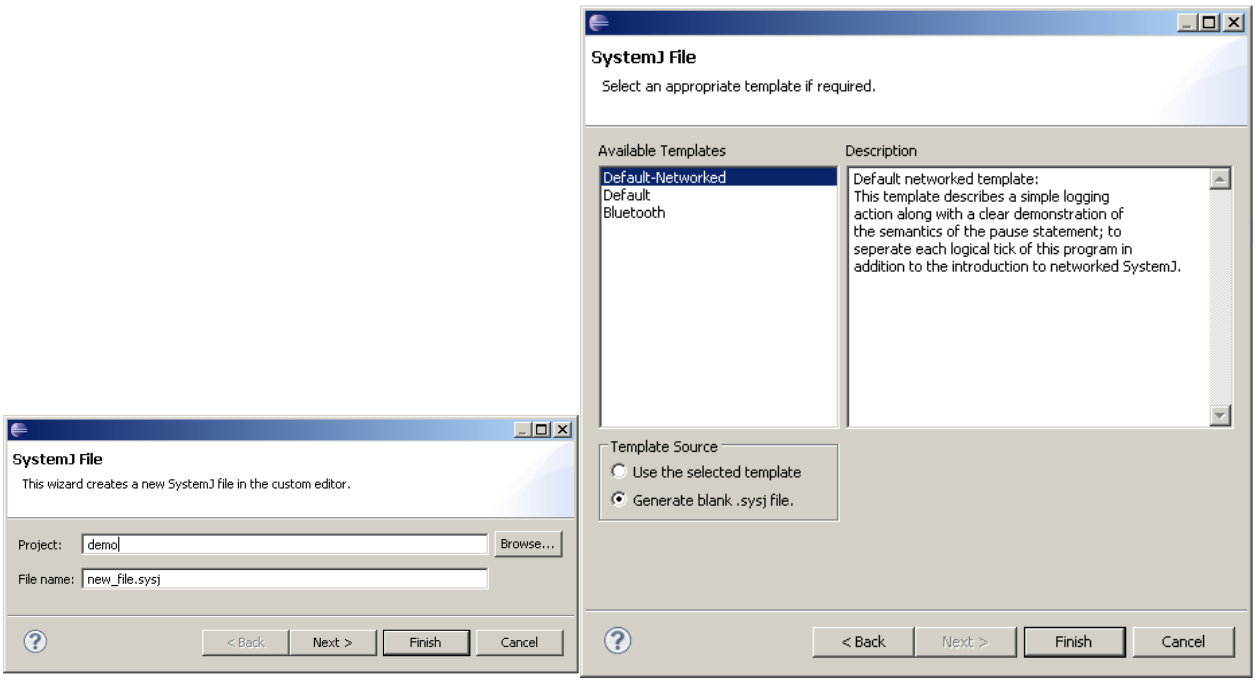

Figure 3: SystemJ Filet Wizard Pages 1-2

Now we will put together a simple example where two reactions in two different clock domains communicate using send and receive statements to demonstrate several of the keywords of the language. The final result should consist of two reactions and a system definition; we will start with the system definition. Please note, if you require more detail feel free to consult the technical documentation or contact us through any of the aforementioned means.

```
system{
        interface{
                /*
                 * Channel Declaration for rendezvous communication
                 */
                output String channel TransferData;
                input String channel TransferData;
        }
        {
                /*
                 * Calling the sending and receiving nodes in asynchronous domain
                 */
                sending(TransferData) //clock-domain
                ><
                receiving(TransferData) //clock-domain
        }
}
```

Figure 4: System Definition

As shown in Figure 4 there are two parts to defining this interface. Firstly, you must declare each of the channels in the format <direction> <type> channel <name>. Following this the asynchronous clock domains need to be defined with respect to the data being sent. The '><' symbol means that the relationship is asynchronous and be aware that the order of the variables in the clock domain definitions above must match that of within the reactions themselves. As this is

only a simple example there is only a single reaction per clock domain and as such the terms will be used interchangeably.

```
reaction sendingGUI(: output String channel TransferData){
	String toSend = null;
	while(true){
		toSend = "Hello World";
		send TransferData(toSend);
		pause;
	}
}
```

Figure 5: Sending reaction

Figure 5 shows how a String “Hello World” is continually being sent along the typed channel TransferData. As you can see the syntax of the while loop matches that of its Java equivalent so bar the requirement to use the keywords ‘reaction’, ‘output’ and ‘channel’ the difference between the Java and SystemJ syntax here is the use of the ‘pause’ command. This denotes a tick boundary and helps to clearly define the consumption of a single time instant (*tick*) of the reactions within a clock domain.

```
reaction receivingGUI(: input String channel TransferData){
	String transDataStr = "";
	while(true){
		receive TransferData;
		transDataStr = (String)#TransferData;
		if(transDataStr != null) {
			System.out.println(transDataStr);
		}
		//Just pausing for a breather before starting again
		pause;
	}
}
```

Figure 6: Receiving reaction

There are three pieces of important information hidden in Figure 6. The most important is the fact that the ‘receive’ command is blocking and as such you must be careful that should it not receive a response appropriate error handling or abort wrappings are in place. Also, the data that comes through the channel needs to be recast to the correct type; a String in this case. Lastly, it is considered bad practice to set and use a variable in the same tick; in fact the compiler will even tell you to fix it. By now your workbench should look similar to that shown in Figure 7.

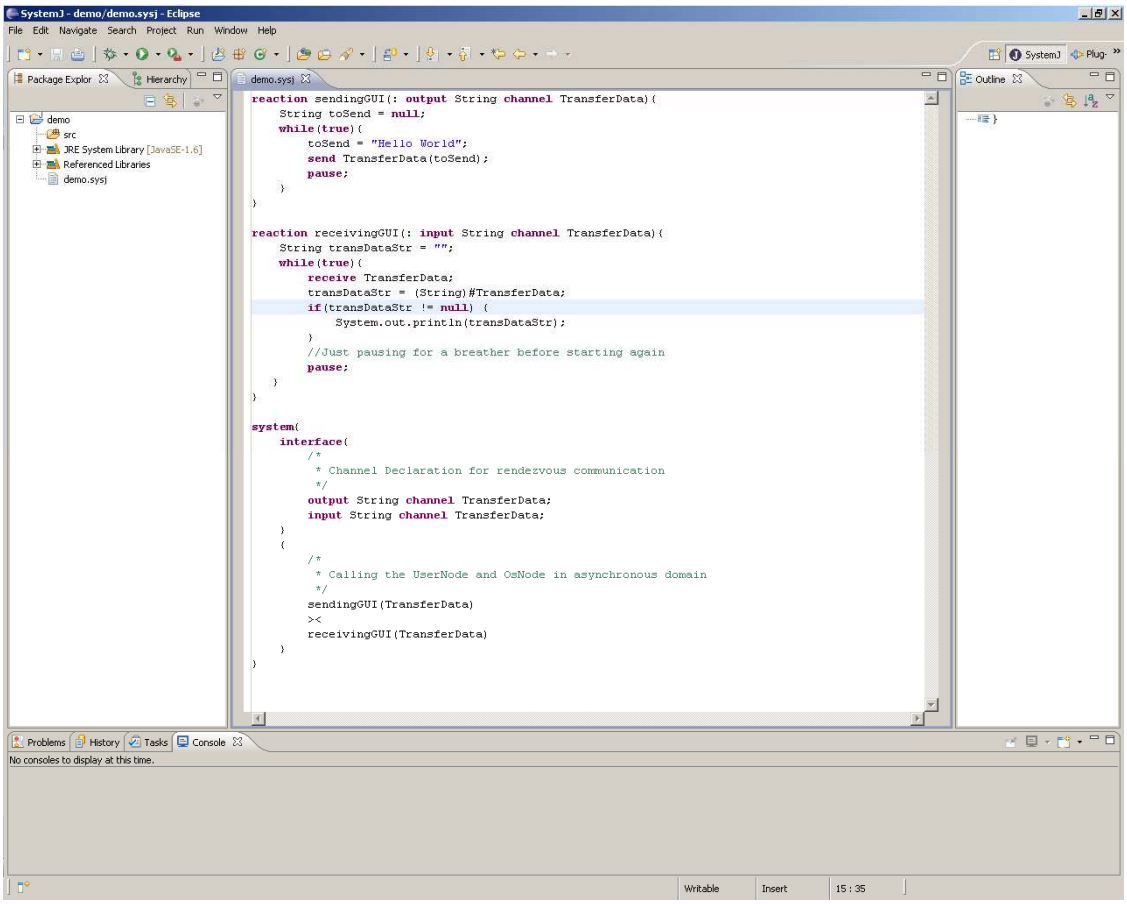

Figure 7: Workbench

By this stage you should have a fair idea of what the .sysj is. To clarify, it can be considered a definition of the control flow in a project while the Java packages define the data aspect. It is because of this clear distinction that SystemJ is able to powerfully, simply and safely describe distributed and concurrent software systems. It does this because it has abstracted away the need of dealing with the delicate details of operating systems and low-level mechanisms for synchronization and communication of concurrent behaviours.

You are probably itching to try out your SystemJ Hello World example by now, right? Well, the good news is that you are literally two clicks away from running or debugging it. Within Eclipse the process of a launch configuration is shortcut>delegate>launch, and through the tools you, the user, can take action at either of the first two levels. The shortcuts have been made context specific and are available at the project level, the respective compiled directory level and the .sysj file itself under 'Run As > SystemJ Application' and 'Debug As > SystemJ Application' whenever any of the three are right-clicked. Moreover, what a shortcut does is essentially configuring a delegate (Figures 8, 9) with default parameters and then launching them. Should you want more control over the launch configuration you can easily alter the parameters at the 'Run > Run Configurations..' (Figure 8) or 'Debug>Debug Configurations..' (Figure 9) options respectively.

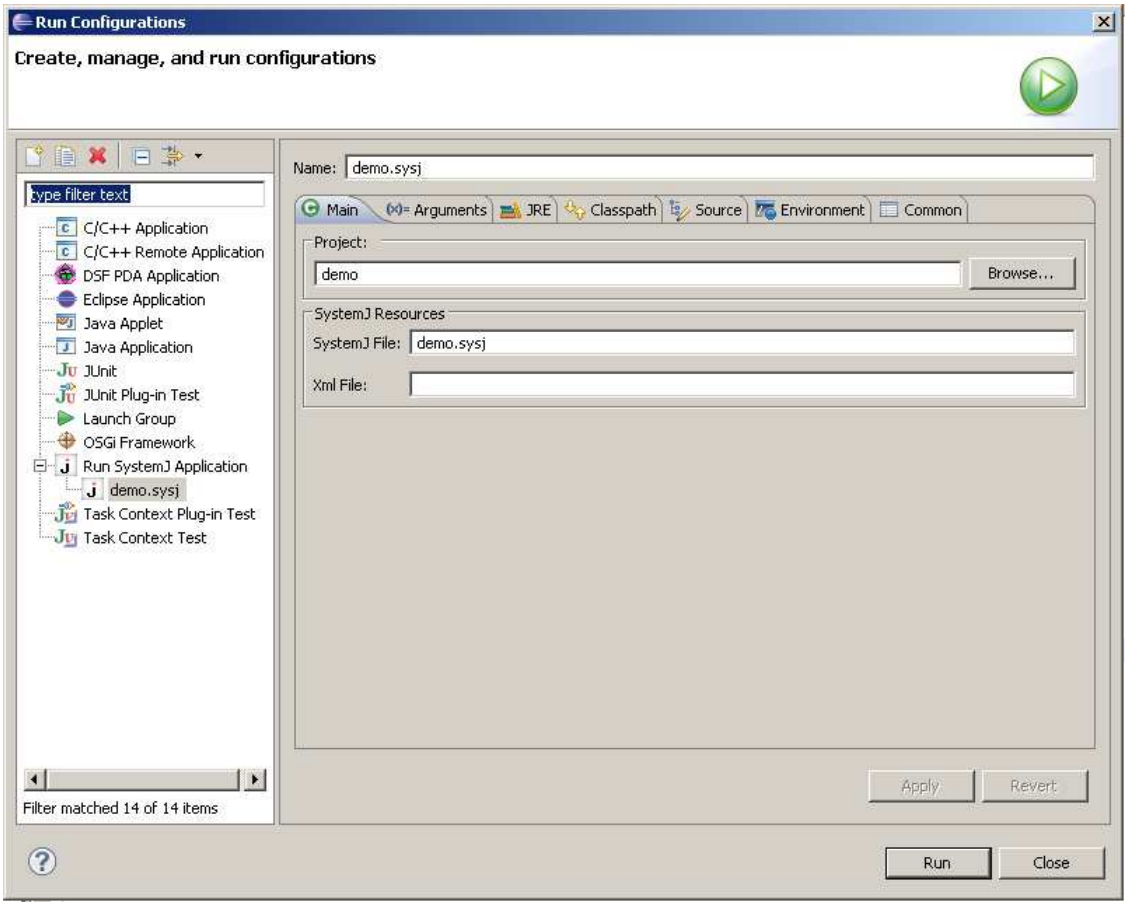

Figure 8: Run Delegate

Take note though, that the debug launch type does not require an .xml due to the manner in which it is compiled. Because of this, as seen in Figures 8 and 9, the two delegates differ slightly in what settings need to be configured for a launch.

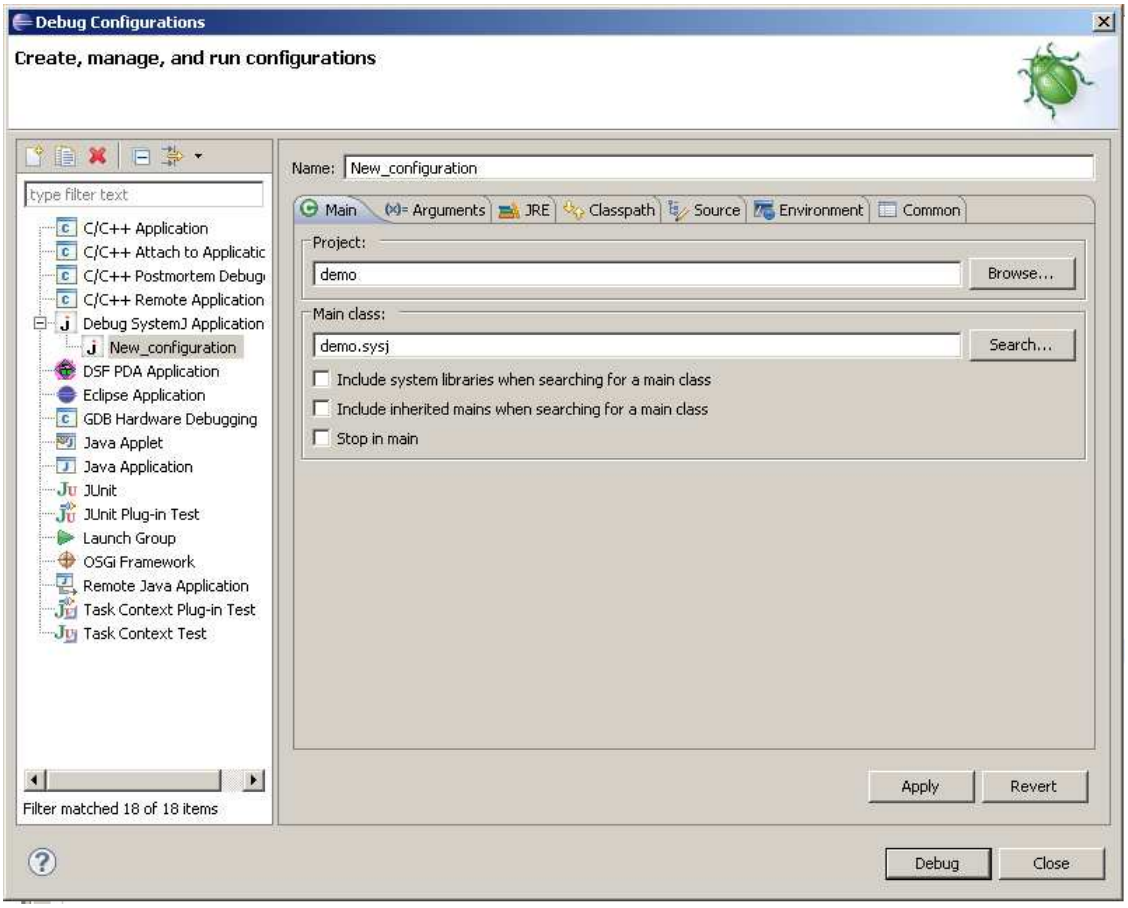

Figure 9: Debug Delegate

SystemJ can also be used for programming networked systems. The project setup for network programming differs slightly compared to the aforementioned project setup. To explain network setup we revisit the setup from the start. Figure 10 shows the three initial dialog boxes that are used to setup the main project components. The first two dialog boxes set the name of the project and the libraries as previously seen in Figure 2.Essentially the difference lies in the third dialog box as, compared to the one in Figure 2, the designer needs to tick the 'Generate a .xml file' option along with the .sysj file option. SystemJ uses an external XML (Extension Markup Language) file to bind the channels and interface signals with the underlying physical layer.

Figure 11 shows an example SystemJ program with two clock-domains acting as a server and client communicating with each other via channels bound to the underlying physical network layer. The top left screen shows the SystemJ program code, the top right shows the corresponding XML file, the bottom left highlights the TransferId output channel binding for clock-domain UserNode and the bottom right highlights the TransferId input channel binding for clock-domain OsNode.

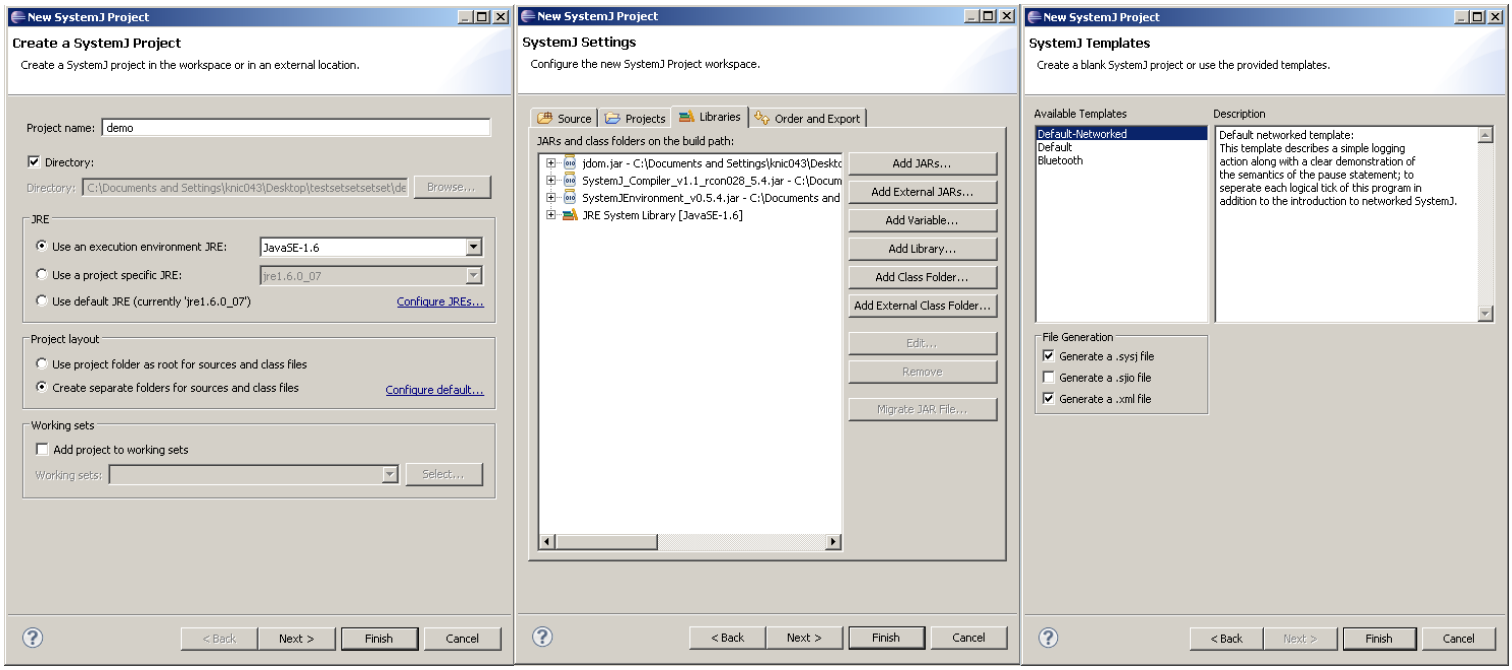

Figure 10: Setting up a SystemJ project for programming networked systems

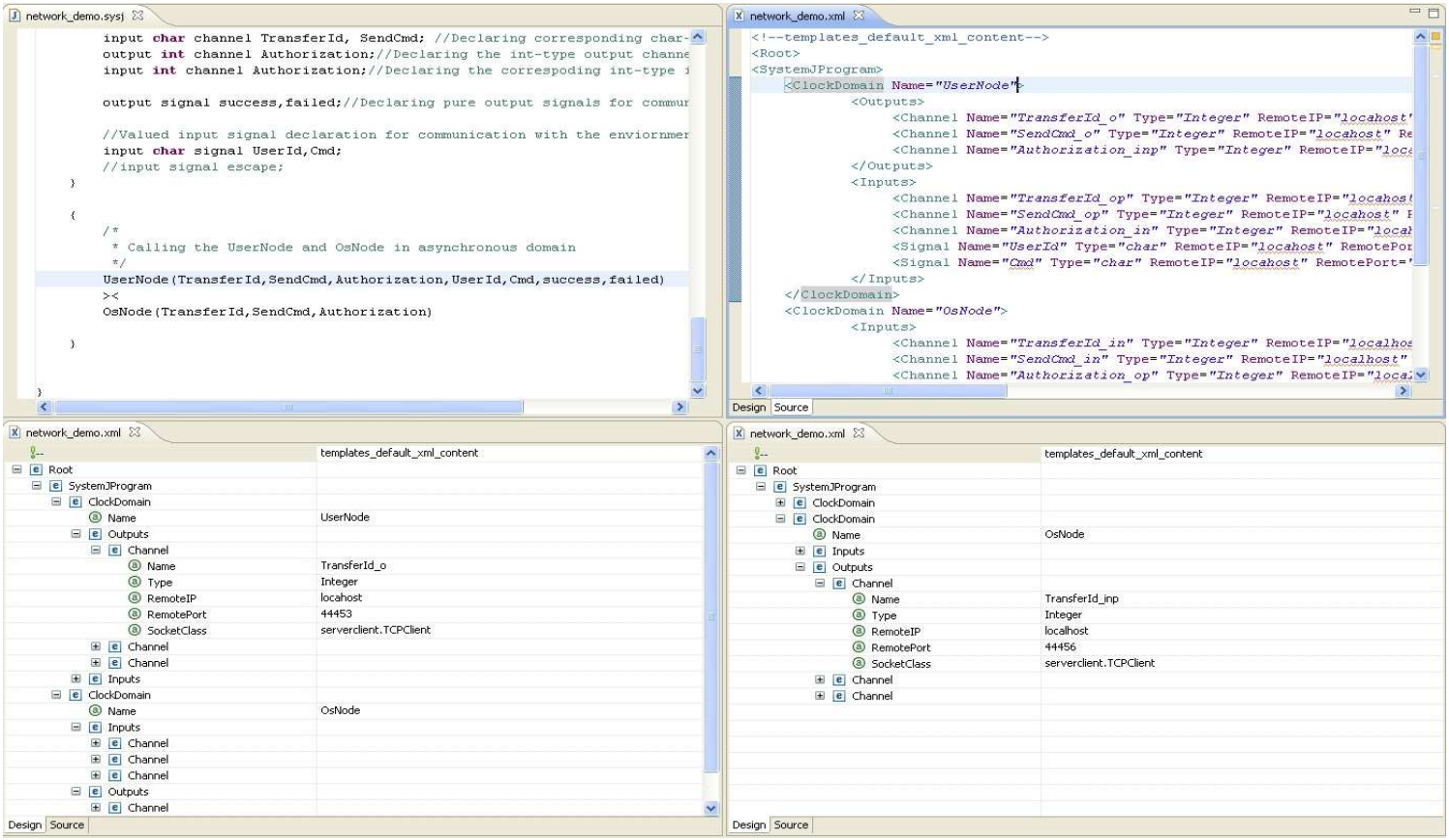

Figure 11: An Eclipse session split into four windows.