

# The Tempo User Interface



VeroModo, Inc.

August 24, 2011

## **Abstract**

The Tempo User Interface provides a convenient environment for developing and working with Tempo specifications. It provides a Tempo-aware editor, automatic static checking, and access to tools such as the Tempo simulator, the Uppaal model checker, the PVS theorem prover, and the LaTeX translator. This user interface is much easier to use than a standard text editor and the Tempo command-line interface.

This document was prepared with version 0.2.4 Beta of the Tempo User Interface. If you are using a newer version of the UI some of the graphics in this document may be outdated.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Recent Changes</b>	<b>1</b>
<b>3</b>	<b>Tutorial</b>	<b>1</b>
3.1	Installing the Tempo Toolset . . . . .	1
3.2	Starting the Tempo User Interface (UI) . . . . .	1
3.3	Overview of the UI window . . . . .	2
3.4	Getting ready to write Tempo specifications . . . . .	3
3.5	Writing and viewing Tempo specifications . . . . .	5
3.6	Checking specifications . . . . .	7
3.7	Checking multiple files together . . . . .	9
3.8	Using Tempo Tools . . . . .	9
<b>4</b>	<b>Tempo Tools Tutorial</b>	<b>11</b>
4.1	Simulator . . . . .	11
4.2	Other Plugins . . . . .	13
<b>5</b>	<b>Implementation details</b>	<b>13</b>
<b>6</b>	<b>Troubleshooting</b>	<b>13</b>

# 1 Introduction

The Tempo User Interface (*the UI*) belongs to the Tempo Toolset. It provides a window-based environment for writing Tempo specifications and for applying other tools to those specifications. This manual provides a tutorial introduction to the UI. It is not intended as an introduction to, or reference guide for, the Tempo Language [6] or the other tools in the Tempo Toolset: the Tempo Simulator [5] and the interfaces to the Uppaal Model Checker [1, 4] and the PVS Theorem Prover [3, 2]. Each of these tools has its own user guide which can be found in the documentation directory of the Tempo Toolset.

## 2 Recent Changes

This document has been updated to version 0.2.4 Beta of the Tempo user interface. All images have been updated and the content has been revised to reflect the latest version of the toolset.

## 3 Tutorial

This tutorial provides a step-by-step introduction to installing and using the Tempo User Interface.

### 3.1 Installing the Tempo Toolset

Visit <http://www.veromodo.com>, where the Tempo Toolset is available for download in the form of zipped bundles. Download the bundle which best suits your system, and unzip in the desired directory.

Tempo Toolkit is built on top of Java, and we required either Java EE 1.5 or Java JDK 1.5 or later must be installed and configured prior to running this toolkit.

### 3.2 Starting the Tempo User Interface (UI)

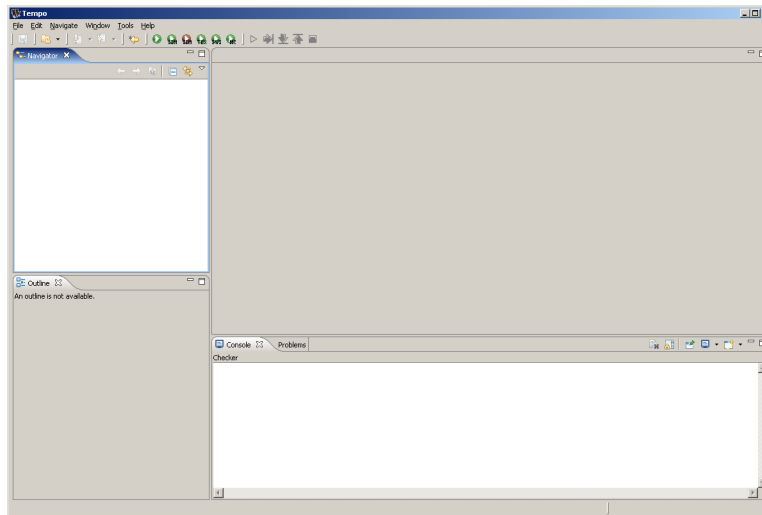
Start the Tempo UI by following the appropriate platform-specific sequence of steps.

**Windows** Browse to the directory where the toolkit was unzipped and find `gui/tempo.exe` file. This file can be executed either by double clicking or directly from the command prompt.

**Macintosh OS X** Using the **Finder**, click on `tempo.app` in the Tempo installation directory. Alternatively, from the command line in a terminal emulator, `cd` to the Tempo installation directory and type `open -a tempo`.

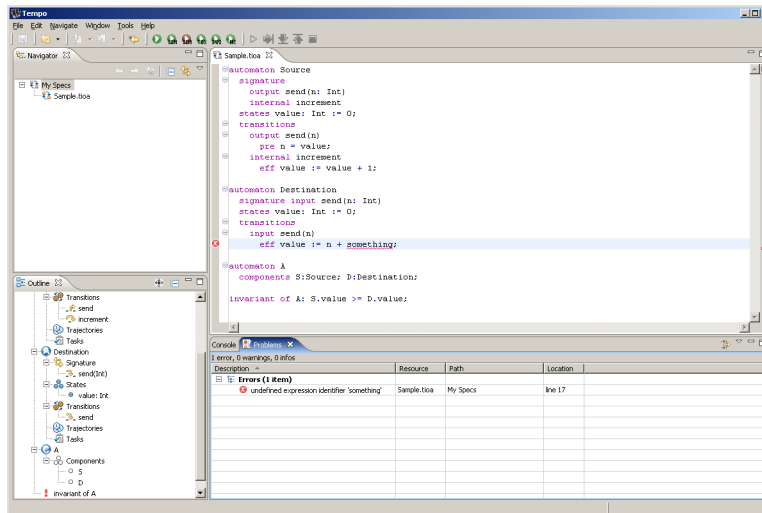
**Linux** Use the command line to `cd` to the Tempo installation directory. Then type `./tempo`.

The first time the UI is started, a blank UI window will appear. The window's appearance may vary slightly from platform to platform, but will be similar to this.



### 3.3 Overview of the UI window

By interacting with the UI, users control both the contents and layout of the UI window. A typical UI window looks like.



At the top of the UI window is a *menu bar*, which contains six pull-down menus (on a Macintosh, the six Temporo menus appear in the menu bar at the top of the desktop, not in the UI window). Three of these menus—**File**, **Edit**, and **Help**—provide standard functionality for text editors (e.g., creating and saving files, cutting and pasting text, and obtaining help). Another—**Navigate**—enables users to jump to a specified location in a file. The **Window** menu enables users to control the appearance

of the UI window and allows access to the Tempo Toolset preference pages. The **Tools** menu provides access to the Tempo Toolset.

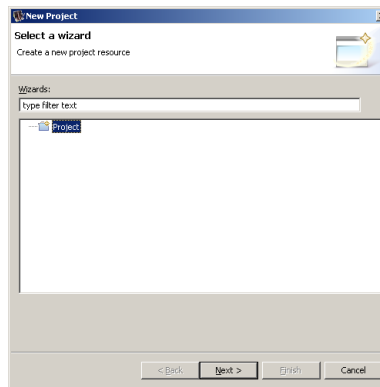
Below the menu bar is a *toolbar*, which enables users to perform certain tasks quickly by clicking an icon. At the bottom of the UI window is a *status bar*, which provides useful information about the current state of the UI (e.g., the location of the cursor, the progress of the current compilation).

Between the menu bar and the status bar are tabbed panes that contain *editors* and several kinds of *views*:

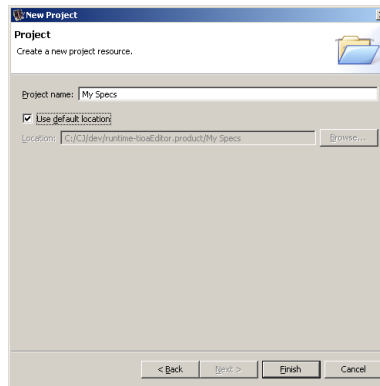
- The **Navigator** provides a hierarchical view of the *projects*—collections of directories and files—accessible to the UI.
- If a Tempo specification (i.e., a file with a name ending in `.tioa`) appears in the currently active editor, the **Outline** provides an overview of its contents: vocabularies, automaton definitions, their primary components, and assertions.
- The **Problems** view calls attention to errors detected by the Tempo parser and static semantic checker.
- The **Console** displays the textual output from the tools in the Tempo Toolset.

### 3.4 Getting ready to write Tempo specifications

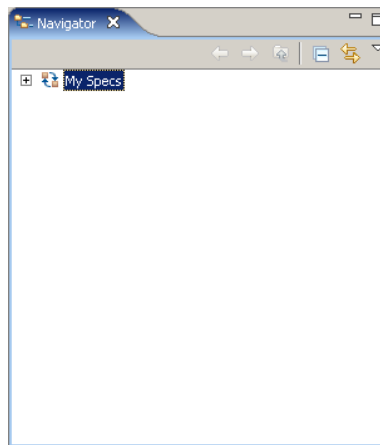
Tempo specifications are stored in *projects*, which are directories with some attached (and generally hidden) metadata. To create a project, right click in the **Navigator** and select **New > Project...** to bring up the first page of the **New Project** wizard.



Double click on the **Project** icon to bring up the second page of the wizard, and enter a name (e.g., **My Specs**) for the new project.



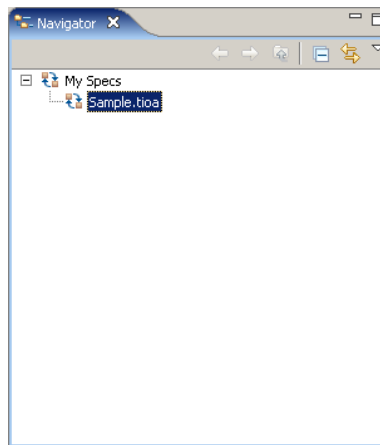
Then click **Finish**. A new project directory will appear in the **Navigator**.



Now right click on the project directory in the **Navigator** and select **New > File** to bring up the **New File** wizard. Fill in a file name (e.g., **Sample.tioa**), making sure that its name ends with **.tioa**.



Then click **Finish**. In response, the UI creates a new empty file, opens an editor for that file, and displays the file name in the **Navigator**.



### 3.5 Writing and viewing Tempo specifications

Now type a Tempo specification into the editor and observe how the UI highlights Tempo keywords.

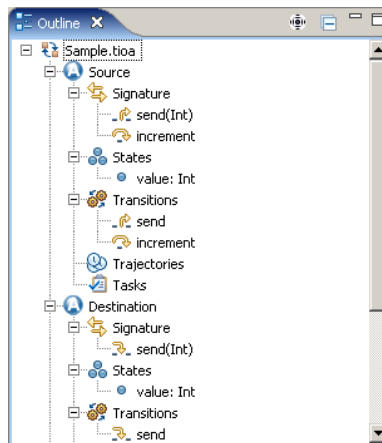


```
automation Source
signature
output send(n: Int)
internal increment
states value: Int := 0;
transitions
output send(n)
pre n = value;
internal increment
eff value := value + 1;

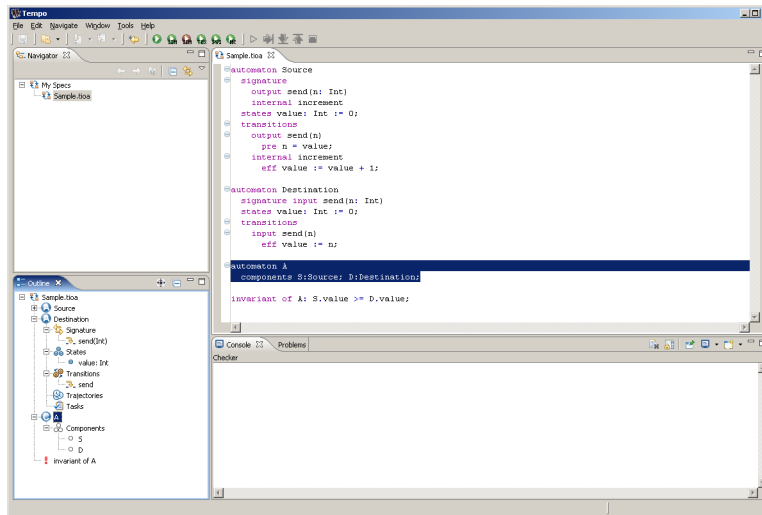
automation Destination:
signature input send(n: Int)
states value: Int := 0;
transitions
input send(n)
eff value := n;

automation A
components S:Source; D:Destination;
invariant of A: S.value >= D.value;
```

The asterisk that appears next to the file name indicates that the editor has unsaved changes. To save these changes, either select **File > Save** or use its shortcut (**Command+S** on a Macintosh, **Ctrl+S** elsewhere). If the specification contains no syntactic errors, the **Outline** view will provide a hierarchical view of its contents.

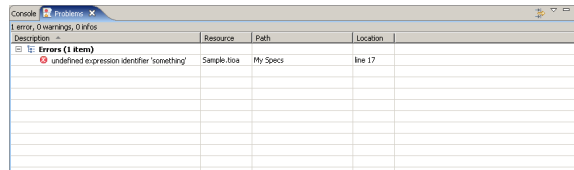


Both the outline and the text being edited can be collapsed or expanded independently by clicking on the plus or minus signs in the boxes in the outline or in the circles in the left margin of the editor. Selecting an item in the outline causes the editor to highlight the corresponding region of text.

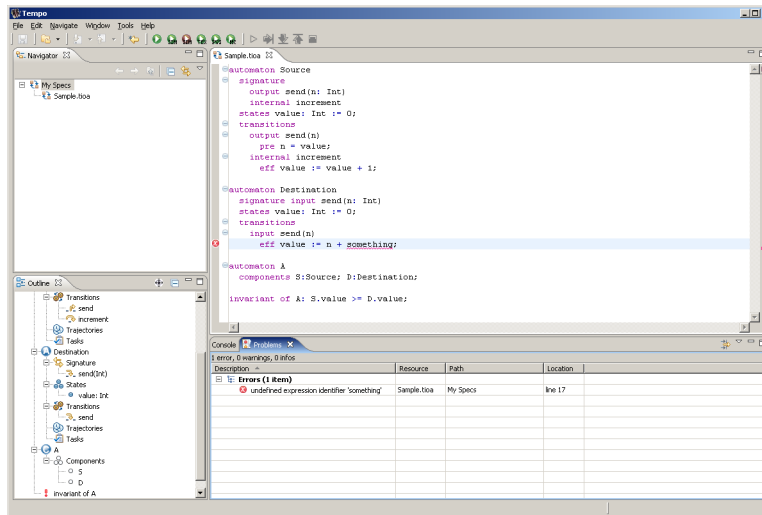


### 3.6 Checking specifications

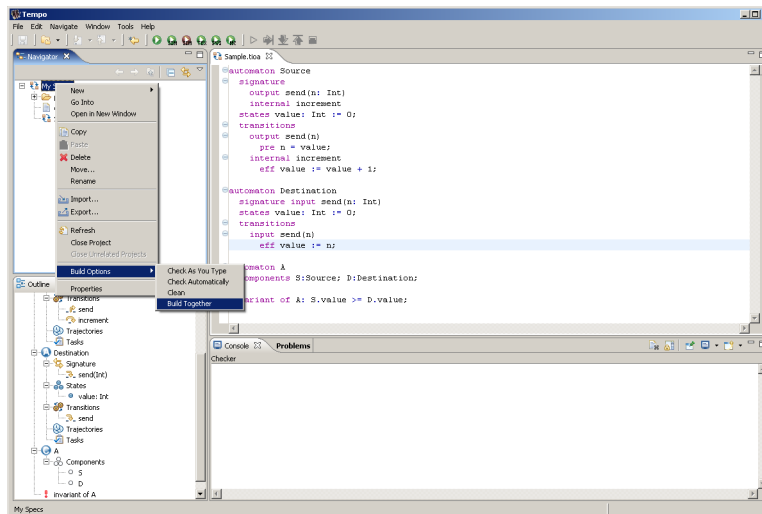
Static semantic checking is performed automatically on all Tempo specifications inside a project. The editor indicates compilation errors by adding underlines and marks in the left margin. Furthermore, the **Problems** view lists all errors found in all open projects.



Hover over a marker in the left margin of an editor to see a description of the problem. Double click on an error in the **Problems** view to highlight the error in an editor and to move the cursor there.



By default the error markers attached to a file are updated each time the file is saved. This behavior is called **Check Automatically**. The project context menu can be used to change the build behavior.

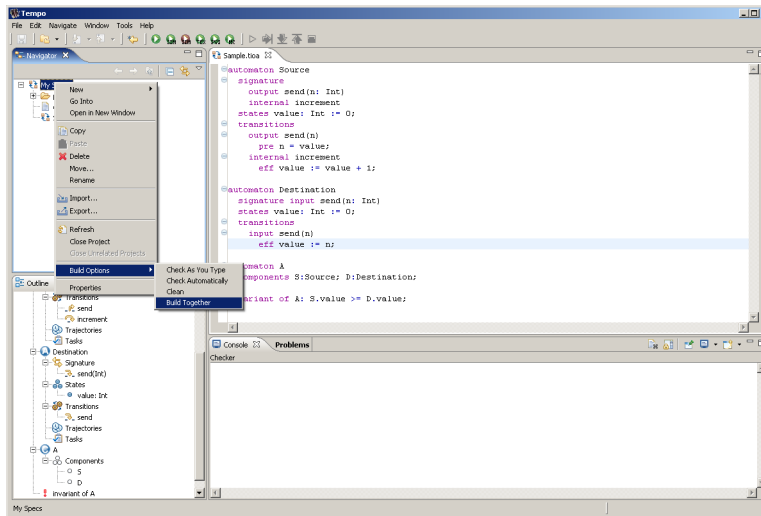


- The **Check Automatically** mode will check the specification each time it is saved.
- The **Check As You Type** mode will check the specification as it is being typed into the editor.
- The **Clean** option marks a file as out of date. The file will be re-checked the next time automated checking occurs.
- If neither **Check Automatically** or **Check As You Type** are enabled the specification must be built manually.

- The **Build Together** option will be explained in the *Checking multiple files together* section of this tutorial.

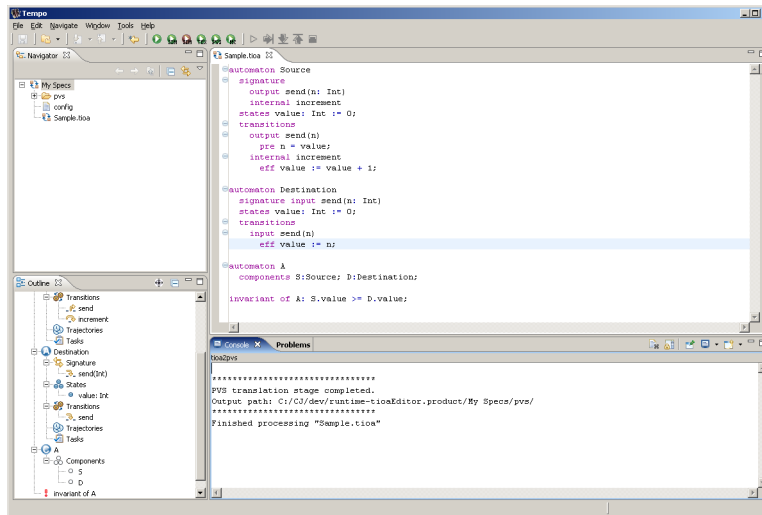
### 3.7 Checking multiple files together

By default the UI will analyze each `.tioa` file separately. In some cases it is convenient to break one specification across several files. A project can easily be configured to compile all its files together by right clicking the project and selecting **Build Options > Build Together**.

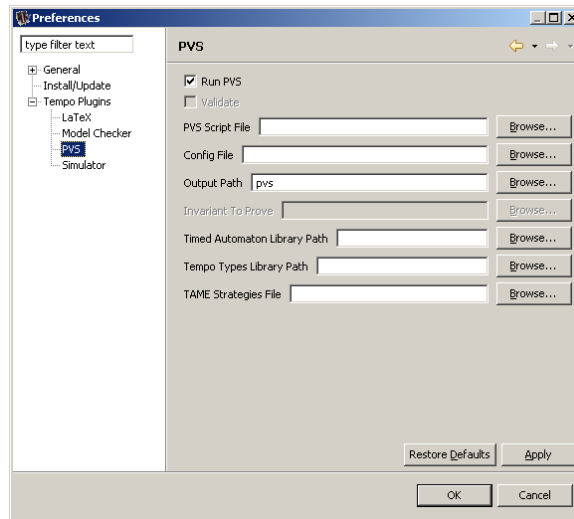


### 3.8 Using Tempo Tools

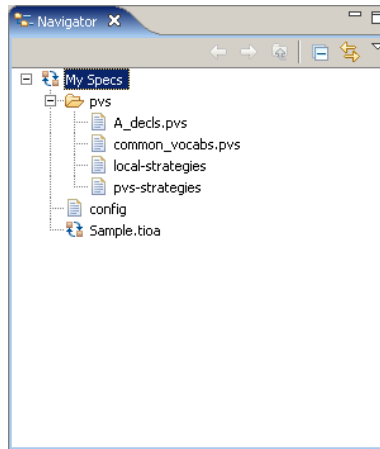
The **Tools** menu allows access to other tools in the Tempo Toolset that provide assistance for more extensive checking. After selecting one of the tools from the menu the active example will be executed with that tool. The resulting output will be printed in the **Console** view.



Each tool can be configured from the **Tempo Preferences** window located in **Window > Preferences...**



Some tools might provide additional output in the form of external files. If the tool has generated additional output, select **Refresh** from the project's context menu in the **Navigator**. For example, the PVS Translator tool generates an additional directory containing several output files.

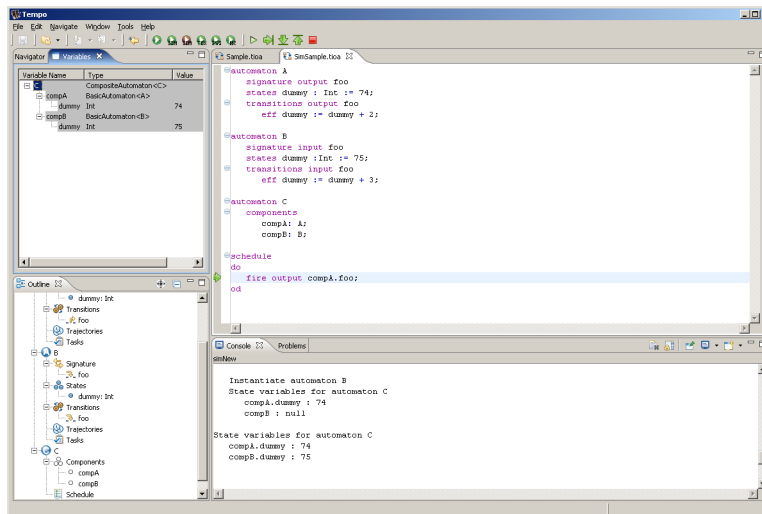


## 4 Tempo Tools Tutorial

This tutorial provides a brief introduction to special features of the Tempo Tools which are only available in the Tempo User Interface.

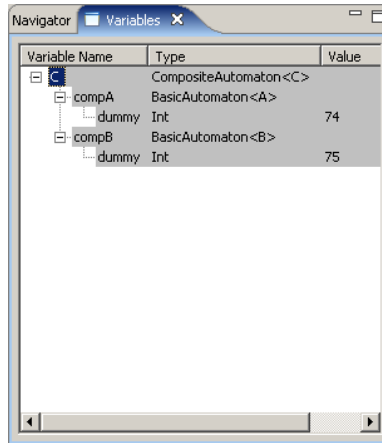
### 4.1 Simulator

In addition to the standard command line functionality, the Tempo Simulator also supports a GUI driven debugger.



The debugger is initiated by clicking on the simulation tool icon. Once the debugger is running, the arrow in the left margin of the editor window will indicate which part of the model is currently

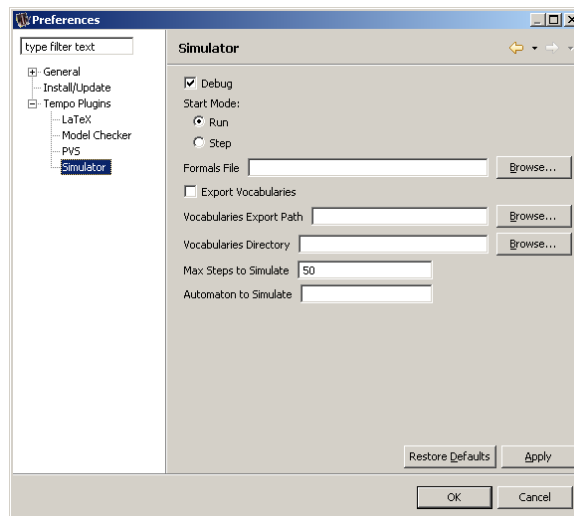
being executed. A break point can be added or removed by double clicking in the left margin. When a break point is reached, the simulation will pause and the details of the current state of the simulation can be viewed in the **variables view**.



While in this paused state the debugger toolbar is used to resume, step, step into, step out, or end the simulation.



The Simulator's Debugger has several configuration options which are not available at the command line.



- The **Debug** check box turns on and off the debugging functionality.

- The **Start Mode** option toggles how often the simulation will pause. While in **run mode** the simulator will run until it hits a break point and will pause at that break point. While in **step mode** the simulator will pause at the first instruction.

## 4.2 Other Plugins

Currently the LaTeX, Uppaal, and PVS translators do not have any extended functionality in the Tempo User Interface. These tools can be configured via the Tempo preferences pages. Details regarding usage of these configuration options can be found in each tool's respective user manual.

## 5 Implementation details

The UI stores the projects that it creates in the **workspace** subdirectory of the its installation directory. The **workspace** subdirectory also contains a file `.metadata/.log`.

## 6 Troubleshooting

Please report any problems to the Tempo development team. This can be done by accessing the Tempo Ticket System — look for it in the Links at <http://www.veromodo.com>, or directly via email [support@veromodo.com](mailto:support@veromodo.com). Following are answers to the common problems:

**After unzipping Tempo does not start.** On Mac and Linux systems, this may be to the permission bits not set correctly.

1. **Mac OS:** open shell prompt and change directory to `<installationdirectory>/tempo.osx.xxx/gui/tempo.app/Contents/MacOS`, then check if the execution bit is set using the following command `ls -l tempo`. If the output you get is similar to

```
-rw-r--r--@ 1 tempouser staff 25296 Aug 15 08:40 tempo
```

then the execute bit is not set. To fix it, type in the following command: `chmod +w tempo`. Now running `ls -l tempo` should return something like this:

```
-rwxr-xr-x@ 1 tempouser staff 25296 Aug 15 08:40 tempo
```

2. **Linux distributions:** open shell prompt and change directory to `<installationdirectory>/tempo.linux\_xxbit/gui`, then check if the execution bit is set using the following command `ls -l tempo`. If the output you get is similar to

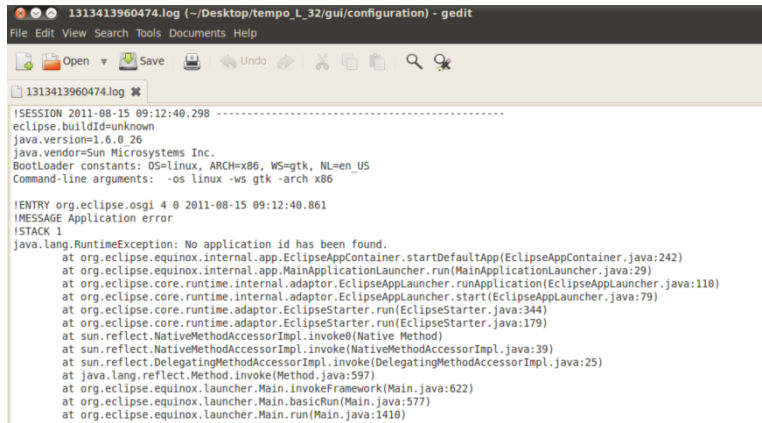
```
-rw-r--r--@ 1 tempouser staff 25296 Aug 15 08:40 tempo
```

then the execute bit is not set. To fix it, type in the following command: `chmod 755 tempo`. Now running `ls -l tempo` should return something like this:



```
-rwxr-xr-x@ 1 tempouser staff 25296 Aug 15 08:40 tempo
```

**Tempo fails to start and I get a JVM error message.** On different systems behavior may differ, but in general after attempting to run the toolkit, a JVM error message appears, or a general error message stating that the error was written to a log file. The error will look something like this:



```
1313413960474.log (~/.Desktop/tempo_1_32/gui/configuration) - gedit
File Edit View Search Tools Documents Help
1313413960474.log
!SESSION 2011-08-15 09:12:40.298 -----
eclipse.buildId=unknown
java.version=1.6.0_26
java.vendor=Sun Microsystems Inc.
BootLoader constants: OS=linux, ARCH=x86, WS=gtk, NL=en_US
Command-line arguments: -os linux -ws gtk -arch x86

!ENTRY org.eclipse.osgi 4 0 2011-08-15 09:12:40.861
!MESSAGE Application error
!STACK 1
java.lang.RuntimeException: No application id has been found.
    at org.eclipse.equinox.internal.app.EclipseAppContainer.startDefaultApp(EclipseAppContainer.java:242)
    at org.eclipse.equinox.internal.app.MainApplicationLauncher.run(MainApplicationLauncher.java:29)
    at org.eclipse.core.runtime.internal.adaptor.EclipseAppLauncher.runApplication(EclipseAppLauncher.java:110)
    at org.eclipse.core.runtime.internal.adaptor.EclipseAppLauncher.start(EclipseAppLauncher.java:79)
    at org.eclipse.core.runtime.adaptor.EclipseStarter.run(EclipseStarter.java:344)
    at org.eclipse.core.runtime.adaptor.EclipseStarter.run(EclipseStarter.java:179)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:25)
    at java.lang.reflect.Method.invoke(Method.java:597)
    at org.eclipse.equinox.launcher.Main.invokeFramework(Main.java:622)
    at org.eclipse.equinox.launcher.Main.basicRun(Main.java:577)
    at org.eclipse.equinox.launcher.Main.run(Main.java:1410)
```

Probably the cause of this message is architectural incompatibility of your JVM with the one used to build the distribution bundle. To remedy this problem you must install the appropriate JVM or configure run options of the tempo file so that it is opened using the right version.

## References

- [1] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer*, 1–2:134–152, 1997.
- [2] Hongping Lim. TIOA to PVS/TAME translator: User manual. 2006.
- [3] Sam Owre, Sreeranga P. Rajan, John M. Rushby, Natarajan Shankar, and Mandayam Srivas. Pvs: Combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414. Springer-Verlag, 1996.
- [4] Veromodo, Inc. *TIOA Model Checker User Guide and Reference Manual*, 2006.
- [5] Veromodo, Inc. *The TIOA Simulator How-To*, 2006.
- [6] Veromodo, Inc. *TIOA User Guide and Reference Manual*, 2006.