JPVTC Help

by

Feras Al-Hawari
## Table of Contents

1. Introduction .............................................................................................. 1  
2. Starting JPVTC ............................................................................................ 1  
3. Opening a new behavioral graph .............................................................. 1  
4. Opening an existing behavioral graph ...................................................... 2  
5. Working with multiple windows .............................................................. 2  
6. Element types ........................................................................................... 2  
7. Editing graphs ........................................................................................... 3  
8. Annotating elements with benchmark data .............................................. 4  
9. Working with loops .................................................................................. 4  
10. Viewing element properties .................................................................... 5  
11. The Tools menu ..................................................................................... 5  
12. Saving a behavioral graph ...................................................................... 7  
13. Exiting JPVTC ....................................................................................... 7
1 Introduction

The JavaPorts Visual Task Composer (JPVTC) is a graphical tool, which can be used, in conjunction with the JavaPorts Visual Application Composer (JPVAC), to construct hierarchical, two-level, structural-behavioral application models. Such models can be used to facilitate QoS negotiations, performance evaluations and rapid prototyping of distributed network computing applications. A JavaPorts (JP) application consists of a set of distributed interacting tasks and its structure can be described using an Application Task Graph (ATG) abstraction. The ATG can be considered as the top (structural) level in the hierarchical application representation. JPVTC is used to construct graphically behavioral models for the tasks of distributed applications. A behavioral model is associated with every application task. Therefore behavioral graphs of tasks can be considered as the lower (behavioral) level in the two-level hierarchical application representation.

2 Starting JPVTC

Make sure you are in the JavaPorts working dir (e.g. ~/javaports-3.0). To start the tool from the command line issue the following:

source jpvtc.sh

After hitting return, the JPVTC tool (see Figure 1) should be launched.

3 Opening a new behavioral graph

Click the New button (see Figure 1) or from the File menu, choose New. A new graph window will open on the JPVTC desktop. You can then begin to construct your behavioral graph in that window.
4 Opening an existing behavioral graph

Click the Open button (see Figure 1) or from the File menu, choose Open. A dialog box will appear with all the .xml files (behavioral graphs are saved in XML format) in the current working directory listed. Choose one of the files in the list and click Open in the dialog to view/edit the behavioral graph.

5 Working with multiple windows

The tool allows you to have multiple graphs open at the same time in separate windows. Each window can be minimized, maximized, or resized within the JPVTC desktop. To move between windows, select the desired window.

6 Element types

In order to capture the behavior of a JP task, the code constructs that contribute to the task's total execution time significantly should be modeled. A JP task might contain iteration constructs (for loops), conditionals (if statements), in addition to sequential code blocks. Moreover, a JP task may contain synchronous/asynchronous message passing operations and may spawn new threads. The JPVTC tool supports several elements for modeling the basic code constructs. Most of the elements can be annotated with attributes and benchmark performance data, as needed to estimate the overall application's performance. The currently supported elements and their attributes are listed in Table 1. Each supported element has its own symbol within the tool. A snapshot of the supported elements symbols is provided in Figure 2. Thus a behavioral task graph consists of nodes (elements) representing basic code constructs and of edges representing dependencies between them.

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Attribute(s)</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>codeSegment</td>
<td>Average execution time</td>
<td>Sequential code block; execution time (in time units) on a reference machine.</td>
</tr>
<tr>
<td>fork</td>
<td>-</td>
<td>Spawn a new thread.</td>
</tr>
<tr>
<td>beginIf</td>
<td>Probability of entering a block</td>
<td>Start of a conditional block.</td>
</tr>
<tr>
<td>endIf</td>
<td>-</td>
<td>End of a conditional block.</td>
</tr>
<tr>
<td>beginLoop</td>
<td>Number of iterations</td>
<td>Start of a loop block.</td>
</tr>
<tr>
<td>endLoop</td>
<td>-</td>
<td>End of a loop block.</td>
</tr>
<tr>
<td>AsyncWrite</td>
<td>Data size (Kbytes), port, message key</td>
<td>Non-blocking write</td>
</tr>
<tr>
<td>SyncWrite</td>
<td>Data size (Kbytes), port, message key</td>
<td>Blocking write</td>
</tr>
<tr>
<td>AsyncRead</td>
<td>Port, message key</td>
<td>Non-blocking read</td>
</tr>
<tr>
<td>SyncRead</td>
<td>Port, message key</td>
<td>Blocking read</td>
</tr>
<tr>
<td>beginAsyncReadLoop</td>
<td>Port, message key</td>
<td>Start of an AsyncRead loop block.</td>
</tr>
<tr>
<td>endAsyncReadLoop</td>
<td>-</td>
<td>End of an AsyncRead loop block.</td>
</tr>
</tbody>
</table>

Table 1: The JPVTC element types and their attributes.
7 Editing graphs

There are five editing modes in JPVTC, these are:

1. **Add Element mode**: To add an element:
   - Click the *Add Element* button (see Figure 1) or from the *Edit* menu, check *Add Element* (to enter the *Add Element* mode).
   - From the *Element* menu, choose an element to be added.
   - Select a window, and then click in any location in the selected window to add the element (note that, while in this mode, every click in the selected window results in adding a new element).

2. **Connect Elements mode**: To connect two elements:
   - Click the *Connect Elements* button (see Figure 1) or from the *Edit* menu, check *Connect Elements* (to enter the *Connect Elements* mode).
   - Hover the mouse over an element (or its connection port), press and drag the mouse over another element (or its connection port), release the mouse to establish the connection between the two elements.

3. **Move Element mode**: To move an element:
   - Click the *Move Element* button (see Figure 1) or from the *Edit* menu, check *Move Element* (to enter the *Move Element* mode).
   - Hover the mouse over an element, press the mouse and move the element to another location, then release the mouse to place the element in the new location.

4. **Copy Element mode**: To copy an element:
   - Click the *Copy Element* button (see Figure 1) or from the *Edit* menu, check *Copy Element* (to enter the *Copy Element* mode).
   - Hover the mouse over an element, press the mouse and drag the new element instance to the desired location, then release the mouse to add the element copy in that location.
5. **Delete Element mode:** To delete an element:
   - Click the *Delete Element* button (see Figure 1) or from the *Edit* menu, check *Delete Element* (to enter the *Delete Element* mode).
   - Click on an element to delete it or click on a connection port to delete the associated connection.

8 **Annotating elements with benchmark data**

Most elements must be annotated with benchmark performance data in order to allow the tool to estimate an application’s performance. We assume that all the benchmark data are collected on a reference machine (e.g. fastest machine in a cluster). To annotate an element with its corresponding benchmark data, click right on the desired element. The corresponding element attributes dialog will appear (see Figure 3). Enter the attribute(s) value(s) and OK the corresponding dialog. If an element does not have attributes a confirmer will appear to inform the user.

![Figure 3: Element attributes dialogs for: (a) codeSegment, (b) beginIf, and (c) beginLoop.](image)

9 **Working with loops**

JPVTC allows modeling loop blocks that contain message-passing operations with ports and/or keys dependent on the loop index (i.e. variable ports and keys). An example on this type of loops is shown in Figure 4(a). In this example, the port and key of the *AsyncWrite* element vary depending on the *bloop1* and *bloop2* indices respectively. The user can model this behavior by specifying the *IDs* of the loops that control the port and key of the *AsyncWrite* operation in the *AsyncWrite* element attributes form that is launched by clicking right on the element (see Figure 4(a)). This kind of loop is quite convenient in modeling the behavior of a Master-Workers distributed application in which the load is partitioned among several Workers, and the Master sends/receives messages to/from several Workers using different ports. Supporting this type of loops in JPVTC reduces the complexity and size of the constructed graph, since a loop encapsulating a single communication operation is equivalent to many message passing operations with different ports and keys.

*AsyncRead* loops can also be modeled using the *beginAsyncReadLoop* and *endAsyncReadLoop* elements as shown in Figure 4(b). Such a loop is exited when the *AsyncRead* finds a received message in the port list element with the specified key. This kind of loop is useful when the distributed application can do something else while it is also polling for an expected message to arrive.
for(p = 0; p < W; p++) { // bloop1
    for(k = 0; k < L; k++) { // bloop2
        port_[p].AsyncWrite(data, k);
    } // eloop2
} // eloop1

while((message = (Message)port_[0].AsyncRead(key))){ // do something
}

(a) Modeling an AsyncWrite with a port and key depending on the loop indices, and (b) modeling an AsyncRead loop.

10 Viewing element properties

To view elements properties (i.e. Ids, types, attributes, or associated comments), select the desired property (to be displayed) from the View menu. Then, the selected property will be displayed beside each element in all the open windows (see Figure 5).

11 The Tools menu

The following items are supported in this menu:

- **Check Graph**: Allows checking the syntax (e.g. if every beginIf element has a matching endIf element) and connectivity of a selected graph.

- **Organize Graph**: Allows centering and organizing a selected graph.
- **Assign Graph:** Allows assigning a selected graph to an application task. When the Assign Graph item is selected, a dialog will appear with a list of all the JavaPorts applications in the JavaPorts working directory (see Figure 6(a)). Select an application name from the list and click the Select button in the dialog. If the application is valid, a dialog with a list of the application tasks will appear (see Figure 6(b)). Select a task and OK the dialog. As a result, the selected behavioral graph will be assigned to the selected application graph. Moreover, the graph will be named after its associated task and will inherit its attributes (e.g. port numbers, application name, AMTP tree). Furthermore, when the Save button is clicked, the graph will be saved in the JavaPorts/App dir of its associated application.

![Select a JavaPorts Application](a)

![Assign Graph to an App Template](b)

**Figure 6:** (a) Select a JavaPorts application dialog, (b) assign graph dialog.

- **Estimate Performance:** Allows estimating the performance of an application. When the Estimate Performance item is selected, a dialog will appear with a list of all the JavaPorts applications in the JavaPorts working directory (see Figure 7(a)). Select an application name from the list and click the Estimate Application Performance button in the dialog. If the application is valid (i.e. a JavaPorts application with behavioral graphs assigned to each of its tasks), its AMTP tree and the behavioral graphs of its tasks will be loaded in JPVTC. In addition, three consecutive dialogs will appear to allow you to enter the information of the reference machine, logical machines, and the logical links between the machines (see Figures 7(b), 7(c), and 7(d)). The machines and links information, as well as the application models are used to estimate the application’s performance. After setting the resources information, the performance estimator will be executed and then its results will be displayed (see Figure 8). An error dialog will appear, instead of the report, when the algorithm detects application deadlock. Moreover, the behavioral graphs of the application tasks will be opened in several windows to allow inspecting the application behavior.
12 Saving a behavioral graph

To save a graph, select a graph window, and then click the Save button (see Figure 1) or choose Save from the File menu. A dialog will appear that allows you to specify a graph name. Enter a graph name and click the Save button in the dialog. If you choose the name of an existing graph, you will be prompted to overwrite the existing file.

13 Exiting JPVTC

To exit the JPVTC tool, choose Exit from the File menu. You will be prompted to save any currently open graphs before exiting.