Refactoring User Interfaces

D4.3

Due date of deliverable: 30 September 2013

Start date of project: October 1st, 2011

Type: Deliverable
WP number: WP4
Task number: T4.2

Responsible institution: University of St Andrews
Editor and editor’s address: Christopher Brown, University of St Andrews
Executive Summary

In this deliverable we describe the user interfaces associated with the refactoring tools as developed under WP4. In particular, we show the user interface for the C++ refactoring tool, as described in D4.2, and the Erlang refactoring tool user interface, as described in Deliverable D4.1. In addition to the refactoring tools user interfaces, we also describe a preliminary user interface prototype for the pattern discovery system.

Positioning of Deliverable 4.3

The positioning of this deliverable (D4.3) with respect to other deliverables is shown in Figure 1. In particular the work presented in D4.3 is directly based upon the work presented in D4.1 (Pattern Transformation System), D4.2 (Enhanced Pattern Transformation System). In addition, the user interfaces have been heavily influenced from feedback by the industrial partners resulting from D6.4 and D6.5.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Executive Summary</td>
<td>1</td>
</tr>
<tr>
<td>Positioning of Deliverable 4.3</td>
<td>1</td>
</tr>
<tr>
<td>1.1 What is new in this deliverable?</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Highlighting Code in Eclipse</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Menu of Refactoring</td>
<td>4</td>
</tr>
<tr>
<td>2.3 User Input</td>
<td>5</td>
</tr>
<tr>
<td>2.4 The Preview Screen</td>
<td>7</td>
</tr>
<tr>
<td>2.5 Displaying the Refactored Code</td>
<td>7</td>
</tr>
<tr>
<td>2.6 Undo</td>
<td>8</td>
</tr>
<tr>
<td>3.1 Highlighting Code in Emacs</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Menu of Refactoring</td>
<td>9</td>
</tr>
<tr>
<td>3.3 User Input</td>
<td>10</td>
</tr>
<tr>
<td>3.4 The Preview Screen</td>
<td>10</td>
</tr>
<tr>
<td>3.5 Displaying the Refactored Code</td>
<td>13</td>
</tr>
<tr>
<td>3.6 Undo</td>
<td>13</td>
</tr>
</tbody>
</table>
1. Introduction

This deliverable describes the user interfaces associated with the refactoring tool support developed for WP4. In particular, in this deliverable we illustrate two novel refactoring tools for both Erlang and C++. The mechanisms and refactoring techniques for both of these refactoring tools are not described here, but in D4.1 for the Erlang refactorings in Chapter 2 and in D4.2 for the C++ refactorings in Chapter 3. In addition to this, in Chapter 4, we describe a prototype pattern discovery interface, that aims to find sources of parallelism in Erlang programs and, by using high-level cost models, displays to the user the best predicted skeleton configuration to achieve near optimal results, together with which appropriate refactoring sequence to apply.

When designing the refactoring tools for C++ and Erlang we have made conscious decisions to keep them unified in as many ways as possible. For each refactoring tool, the programmer uses them in a similar way, typically by highlighting a piece of code in the editor window and then selecting the appropriate refactoring to apply from a drop down menu. Any additional parameters for the refactoring are then provided by the user, a preview option is available for both C++ and Erlang refactoring tools together with an undo feature.

1.1 What is new in this deliverable?

- We show the user interfaces developed for the Erlang refactoring tool, Wrangler, under the Emacs development environment, giving detailed screen shots and explanations;

- We show the user interfaces developed for the C++ refactoring tool, built under Eclipse using the CDT plugin, also with detailed screen shots and explanations.

- We show a preliminary prototype interface for the pattern discovery tool, as part of the Paraphrase-Enlarged extension.
2. Refactoring C++

In this section we describe the C++ refactoring tool user-interface [?]. The refactoring tool itself was previously described in D4.2 The Enhanced Pattern Transformation System. The C++ refactoring tool is built on top of the CDT plugin for Eclipse, this allows the refactorings to be integrated into one of the most active and popular IDE for C++ programmers, giving us several advantages:

- refactoring support is available for programmers already familiar with using Eclipse, so little new programmer intuition is needed;
- the refactorings can take advantage of the many features provided in the CDT plugin for Eclipse, such as previewing, menu support and user input;
- inverting the refactorings can be integrated into the standard undo feature with Eclipse; and,
- pre-condition checking and errors can be presented to the programmer using the Eclipse environment, so that any conflicts of variable names, etc. can be easily highlighted to the programmer.

Figure 2.1 shows a screenshot of the refactoring tool built into Eclipse.

2.1 Highlighting Code in Eclipse

In order to invoke a refactoring, the code is first highlighted in the Eclipse IDE before the programmer selects which refactoring to perform.

2.2 Menu of Refactoring

The programmer selects which refactoring to perform by selecting the desired transformation from a drop-down menu, as illustrated in Figure 2.2. In the fig-
Figure 2.1: The Eclipse refactoring tool.

ure, the Refactor menu is shown in Eclipse, with a number of possible refactorings to apply. The refactorings from D4.2 to introduce new parallel skeleton structures are shown, such as Introduce Farm Declaration, Introduce Farm, Introduce Pipeline Declaration and Introduce Pipeline. Figure 2.2 also shows a number of other structural based refactorings that can also be applied to the highlighted code, such as renaming, extract variable, etc. These refactorings are available for C++ as part of the standard CDT refactoring plugin in Eclipse. In order to invoke one of these refactorings, the user must first highlight a portion of code that he/she wants to convert to a parallel skeleton, as discussed in Section 2.1.

2.3 User Input

Often a refactoring requires additional information from the programmer, such as the number of workers for a farm skeleton, or the new name for the farm or pipeline skeleton. In this case, the refactoring asks the user for additional information after the refactoring is selected from the menu. Figure 2.3 shows this input page, where the user is required to fill in the parameters for the refactoring. For all refactorings, default parameters are given, such as a default pipeline/farm skeleton name. If the programmer inputs a parameter that causes a refactoring precondition to fail, such as specifying a name that already exists in scope in the program, the refactoring gives an error message to the programmer, informing them to enter another name. The input page has a cancel button, that cancels the refactoring, a preview button to invoke a preview of the refactoring transformation (described in Section 2.4) and a OK button, to perform the refactoring directly without a preview.
Figure 2.2: The Menu of Refactorings to Apply in Eclipse

Figure 2.3: Screen capture showing a C++ refactoring requiring further programmer parameters, such as the name of the new pipeline to be introduced and the name of the component
Figure 2.4: The refactoring preview screen in Eclipse, showing the before and after effects of applying the desired refactoring.

2.4 The Preview Screen

Figure 2.4 shows the preview screen in Eclipse. The code on the left shows the current program without the refactoring applied and the code on the right shows after the refactoring has been applied. The preview screen also highlights the code in the right that will be inserted or changed by the refactoring process, with an arrow to its original position in the left. The top of the preview screen shows all of the files that will be modified by the refactoring. The programmer can see the changes made to each file by selecting the appropriate file in top of the preview screen. The preview screen allows the programmer to return back to the input page (described in Section 2.3), to cancel the refactoring, or to apply the refactoring change.

2.5 Displaying the Refactored Code

Figure 2.5 illustrates the output of a refactoring process. Here the code is modified directly, with the code insertions/modifications affected by the refactoring is presented directly to the programmer.
2.6 Undo

All of the refactorings in Eclipse have a fully supported undo feature, integrated into Eclipse’s undo button in the edit menu, allowing a complete refactoring step to be undone.
3. Refactoring Erlang

This section describes the user interface for the Erlang refactoring tool, embedded within the Wrangler [1] refactoring tool. Previous discussion on the Erlang refactorings for parallelism are described in D4.1 The Pattern Transformation System. Wrangler is embedded into the Emacs text editor by default, targeting a popular text editor used by the Erlang community. Like using Eclipse for C++, using Emacs for Erlang programmers has the advantage that the refactorings are available to them in a familiar environment, lending to intuition and usability. The user-interface described here has the same features as the C++ refactoring tool for Eclipse, such as the preview feature, undo, highlighting code to be refactored and a menu of possible refactorings to apply. And errors in applying the refactorings themselves are relayed to the user through Emacs. Figure 3.1 shows a screenshot of the Erlang refactoring tool, Wrangler, embedded into Emacs, a popular text editor for Erlang programs. Emacs supports full Erlang text-highlighting, together with a menu for possible refactorings to apply.

3.1 Highlighting Code in Emacs

In order to invoke a refactoring in Wrangler, first the programmer highlights a block of code such as an existing skeleton or a sequential expression that they wish to transform into another parallel skeleton or to introduce a parallel version. Code that is highlighted is shown with a yellow background, which can be configured to a user-defined colour, if required.

3.2 Menu of Refactoring

Figure 3.3 shows the menu of refactorings available for Erlang in Wrangler. In the menu, there are a number of possible refactorings to apply, including a standard set of structural based refactorings, such as renaming, introduce new variable, function folding and function unfolding. The refactoring to introduce and tune parallelism are given in a sub menu, where the refactorings pipeline introduction, farm introduction, map introduction and introduce chunking can be applied.
3.3 User Input

3.4 The Preview Screen

The preview process in Wrangler is made up of a number of different stages.
Figure 3.3: Screenshot showing the menu of Erlang refactorings to apply, including pipeline introduction, farm introduction, map introduction and introduce chunking refactorings.

Figure 3.4: Screenshot showing Wrangler asking the programmer to commit the refactoring, preview the result, or to cancel the refactoring process.
Figure 3.5: Screenshot showing the Wrangler preview process. The code block highlighted in blue will be transformed into the code shown in Figure 3.6. The refactoring tool asks the programmer to apply the refactoring, or to quit the refactoring process.

Figure 3.6: The Wrangler preview screen showing the code that will replace the highlighted expression.

1. After selecting which refactoring to apply from the drop down menu, Wrangler displays a message to the user asking if they want to apply the refactoring to the selected code fragment. This process is illustrated in Figure 3.4, where the selected code entity under refactoring is highlighted in blue in the left column (changed from the user-selected yellow), the output of the refactoring is shown in the right hand column and the Wrangler tool asks the programmer if they want to apply the refactoring or cancel the process. This is particularly useful if the programmer accidentally selects the incorrect refactoring to perform.

2. Figure 3.5 shows a close-up of Wrangler asking the user to apply the refactoring or to cancel the process. The user responds by typing Y or N into the input box.

3. The refactored code, i.e., the result of the refactoring process is shown in the right column, illustrated in more detail in Figure 3.6.

4. When the programmer chooses to apply a refactoring, Wrangler then asks

Figure 3.7: The tool asks the programmer if they want to preview/commit/or cancel the changes to be performed.
Figure 3.8: The preview screen in Wrangler shows the original source file in the top window and the result of the refactoring process in the bottom window. The user can cycle through each affected code fragment in turn, before deciding to commit the refactoring or to cancel the process.

the programmer if they want to commit the refactoring directly to the source file, or if the programmer would like to preview the result instead. This is illustrated in Figure 3.7.

5. Finally, the preview screen is illustrated in Figure 3.8. Here the original un-refactored code is shown in the top window, with the particular program expression under refactoring scrutiny highlighted in grey. In the bottom window the result of the refactoring process applied to the highlighted code fragment is shown. The user can step through each refactoring result in the file and commit the changes at any time to exit the preview process.

3.5 Displaying the Refactored Code

Figure 3.9 shows the result of the refactored process, with the transformed source code displayed in the top window and a list of refactored (changed) Erlang files in the bottom window.

3.6 Undo

Similarly to the C++ refactoring tool, Wrangler also has a built-in undo feature, allowing the programmer to invert a refactoring a step if necessary.
Figure 3.9: Screenshot showing the result of the refactoring process
4. Pattern Candidate Browser

Pattern candidates are code fragments which are amenable to parallelization by transforming them into applications of high-level parallel patterns. One aim of the ParaPhrase–Enlarged project is to identify pattern candidates in Erlang programs. We have developed a user interface component (Pattern Candidate Browser) that is capable of presenting pattern candidates, and recommending profitable parallelization transformations. This component does not perform transformations – its purpose is to provide information (analysis results) that the programmer can use to carry out parallelization refactorings. The requirement imposed against the Pattern Candidate Browser is that it should present information without overloading its users with unnecessary details, but highlighting the key decisions that must be made.

The input to Pattern Candidate Browser is provided by another software component: the Pattern Discovery Component. This latter is being developed in WP2. It analyses Erlang source code, and produces a list of possible “transformation sequences”. A transformation sequence is a collection of related transformations that can be applied on a given code fragment. Currently the Pattern Discovery Component is capable of proposing transformation sequences that contain the introduction of pipe and farm skeletons [1, 4] as building blocks. For instance, consider the following expression in a matrix multiplication function.

```erlang
lists:map(fun(R) ->
    lists:map(fun(C) -> mult_sum(R, C) end, Cols)
end, Rows)
```

The `lists:map` function applies a function to each element of a list. The outermost occurrence of `lists:map` applies a function on the rows of a matrix, and the innermost occurrence applies another function on the columns of another matrix. This latter function computes the scalar product of a row and a column by calling `mult_sum`. This code fragment can be identified by the Pattern Discovery Component as a pattern candidate, and a proposed transformation sequence can be “farm of farm”, that is the introduction of a farm for both occurrences of `lists:map`. Note that the Pattern Discovery Component may propose more than one possible transformation sequence for the same pattern candidate.

By benchmarking the sequential execution time of subexpressions of a pattern candidate, one can make an estimation for the execution time of the transformed
(parallelized) code. The estimation is based on a cost model as described in [1, 4].

The Pattern Discovery Component is being implemented with RefactorErl [3], a static program analysis and transformation tool for Erlang, which supports some static analyses useful for pattern discovery, but not available in Wrangler. Therefore we have integrated RefactorErl into Wrangler.

The Pattern Candidate Browser is a web-based component, implemented as a set of web-services and a Google Visualization API [2] based front-end that executes in a web browser. The component supports multiple simultaneous users, and it can persist analysis results for faster data retrieval and improved user experience. For further processing or studying the results, the Pattern Candidate Browser can export data in CSV and XML formats.

Figure 4.1 illustrates the capabilities of the Pattern Candidate Browser. The upper table is an overview of the processed transformation sequences. The lower table gives a more detailed view of a selected transformation sequence, with a description of each constituting (farm or pipe introducing) transformation. A transformation sequence is characterized by a location (the entry point of the pattern candidate as a module-function-arity triple), an abstract skeletal configuration (which describes a composition of pipe and farm skeletons – for instance, a “farm of farm” is written as (!(!e230)), where the bang denotes the farm skeleton, and e230 is a generated symbolic name for an expression), and the expected speedup (for CPU and GPU). The description of a transformation adds more specific location information, a view of the corresponding source code fragment, benchmarked sequential execution time on CPU and on GPU, and estimated parallel execution time on CPU and on GPU. The data rows in the upper table are initially ordered by expected speedup (and hence by confidence in recommendation), but re-ordering by another column is also possible.
### Transformation sequences

<table>
<thead>
<tr>
<th>ID</th>
<th>Configuration</th>
<th>Module</th>
<th>Function</th>
<th>Arity</th>
<th>Expected speedup (CPU)</th>
<th>Expected speedup (GPU)</th>
<th>Recommended?</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>$(x,y)$</td>
<td>main_x</td>
<td>main_seq</td>
<td>2</td>
<td>204.82</td>
<td>1.00</td>
<td>✓</td>
</tr>
<tr>
<td>7</td>
<td>$(x,y)$</td>
<td>main_x</td>
<td>main_seq</td>
<td>2</td>
<td>235.77</td>
<td>1.00</td>
<td>✓</td>
</tr>
<tr>
<td>8</td>
<td>$(x,y)$</td>
<td>main_x</td>
<td>main_seq</td>
<td>2</td>
<td>107.08</td>
<td>1.00</td>
<td>✓</td>
</tr>
<tr>
<td>9</td>
<td>$(x,y)$</td>
<td>main_x</td>
<td>main_seq</td>
<td>2</td>
<td>307.49</td>
<td>1.00</td>
<td>✓</td>
</tr>
<tr>
<td>10</td>
<td>$(x,y)$</td>
<td>main_x</td>
<td>main_seq</td>
<td>2</td>
<td>31.02</td>
<td>1.00</td>
<td>✓</td>
</tr>
<tr>
<td>11</td>
<td>$(x,y)$</td>
<td>main_x</td>
<td>main_seq</td>
<td>2</td>
<td>4.08</td>
<td>1.00</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Details of the transformation sequence

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Location information</th>
<th>Program text</th>
<th>Sequential CPU time</th>
<th>Sequential GPU time</th>
<th>Parallel CPU time</th>
<th>Parallel GPU time</th>
<th>Expected speedup (CPU)</th>
<th>Expected speedup (GPU)</th>
<th>Used stream length</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(x,y)$</td>
<td>$(x,y)$</td>
<td>$k(x) = \text{main}(x, y)$</td>
<td>0.44</td>
<td>0.00</td>
<td>0.44</td>
<td>0.00</td>
<td>1.00</td>
<td>1.00</td>
<td>1</td>
</tr>
<tr>
<td>$(x,y)$</td>
<td>$(x,y)$</td>
<td>$k(x) = \text{main}(x, y)$</td>
<td>4.0809</td>
<td>0.00</td>
<td>2.9918</td>
<td>0.00</td>
<td>1.37</td>
<td>1.00</td>
<td>10.000</td>
</tr>
<tr>
<td>$(x,y)$</td>
<td>$(x,y)$</td>
<td>$k(x) = \text{main}(x, y)$</td>
<td>40.8556222124</td>
<td>0.00</td>
<td>154.35223994</td>
<td>0.00</td>
<td>254.81</td>
<td>1.00</td>
<td>10.000</td>
</tr>
</tbody>
</table>

Figure 4.1: The front-end of the Pattern Candidate Browser
5. Conclusion

This deliverable has described the user interfaces for the Erlang refactoring tool, Wrangler, and also the C++ refactoring tool, built into Eclipse. This deliverable has built upon the foundational work in D4.1 where the Erlang refactoring tool and subsequent refactorings are described, and also D4.2, where the C++ refactoring tool is described, together with a number of use-cases.

In Chapter 2, we presented the C++ refactoring user interface built into the Eclipse IDE for C++ programmers. Similarly, in Chapter 3 we presented the user interface for the Erlang refactoring tool, built into Emacs. Although both refactoring tools use different text editors, they both share the same user interface process. The user interface to the refactoring process requires the programmer to highlight a portion of code in Eclipse and then select the appropriate refactoring to apply from the refactoring drop-down menu. This in turn brings up a dialog box asking the user for further input parameters and the option to preview the result of the refactoring. The result of the refactoring transforms the source file directly, with the changes directly shown in Eclipse.

In Chapter 4 we gave a description of the prototype user interface for the pattern discovery tool, which locates sources of parallelism within Erlang programs (i.e., the discovery of skeleton instances), and displays a report of each pattern instance together with its predicted speedups after apply a particular sequence of refactorings (the sequence of refactoring steps to apply is also report). Each pattern instance is also displayed with its position in the source file. In the near future we hope to incorporate this pattern discovery result directly into the refactoring tools themselves.
Bibliography


