Bachelor Thesis
in RBG
Summer Semester 2007

Rechnerbetriebsgruppe
Fachbereich Informatik
TU Darmstadt

Extension of Refefree contest system

O Hung Lun
1177796

Supervisors: Dr. Guido Roessling
Martin Girschick

June 14, 2007
## Contents

**Declaration**

- III

**Abstract**

- IV

### 1 The Original version of Referee

1.1 Overview of Contest and Referee

1.1.1 Contest

1.1.2 Referee

1.2 Usability

1.3 The usability of Referee

1.3.1 Effectiveness

1.3.2 Efficiency

1.3.3 Satisfaction

### 2 Recreate Referee for use in schools

2.1 A new plan

2.2 The guideline to usable design

2.2.1 Understand and specify context of use.

2.2.2 Specify the user and organizational requirements.

2.2.3 Produce Design solutions

2.2.4 Evaluate specifications against requirements.

### 3 The new Referee

3.1 Overview of the new Referee

3.2 Offline mode

3.3 Support for Pascal

3.3.1 Problem with dcc32

3.3.2 dcc32 vs. fpc

3.3.3 Suggestion for improvement

3.4 Optimising the workflow

3.4.1 New Problem Panel

3.4.2 Template

3.4.3 Open editor from GUI

3.4.4 Manual Test

3.5 German graphical interface

### 4 Using the configuration files

4.1 Configuration files

4.2 Compilers

4.3 Known bugs

### 5 Installation Guide

5.1 Pre-installation check-list

5.2 Performing the installation

5.3 Starting the program
6 Conclusion 28

7 Acknowledgement 29

8 Templates 30

9 dcc32utility.java 35

10 Literature 36
Declaration

All helping sources that contribute towards the completion of this Bachelor Thesis are explicitly acknowledged and it has not been submitted to any examination authority.
Darmstadt, June 14, 2007
Abstract

TU Darmstadt holds an annual programming contest among her own students. In the future, the contest may extend its reach to secondary schools. The main objective of this contest is to encourage algorithmic thinking among the contestants. To facilitate the contestants in both testing and submitting their solutions, a software, by the name of Refefree, is put into use. It is created by Marcus Moll and Wolfgang Hess at TU Darmstadt. In this thesis, we seek to analyse several questions related to Refefree. To what extent has Refefree succeeded in helping the contest achieve its objective? Are there room for its improvement? In what ways do the new features, developed in course of this thesis, improve the software? In the following chapters, we shall have a brief look at the original version of Refefree. Then, we systematically expose its weaknesses and make suggestions for future development, and at the same time, motivate the new features found in the new Refefree, which are developed in course of this thesis.

An installation guide is attached at the end of this thesis. The reader of this thesis is recommended to first read the [1]study report, written by Marcus Moll and Wolfgang Hess.
1 The Original version of Refefree

1.1 Overview of Contest and Refefree

In this chapter, we shall describe what the contestants typically do in a programming contest, as they try to finish the contest problems. There are several steps that they must go through for each problem that they are trying to solve. Some of these steps are done through the interface provided by Refefree. When Refefree was developed, the target users of this software were the first semester students in TU Darmstadt. Therefore we shall evaluate it with respect to this group of users. We do not attempt to evaluate Refefree in its entirety, but rather on its core; therefore, some of its functionalities will just be briefly mentioned but not discussed in details.

1.1.1 Contest

Let’s assume that the contestants are organised into teams of three. Each team is given just one computer and is assumed to do the following in the contest.

1. start: is given the description of all contest problems

2. initialise: prepares test cases for each problem

3. design and implementation: one part of the team designs the solution while the other part implements the design in the computer.

4. test: test the solution against test cases

![Contest Model](image)
5. submit: submit the solution to the judges

At the beginning of the contest, all the three team members are involved in the design of solution. Once the design of the first problem is finished, one part of the team will implement the solution while the other part carries on with solution design for the next problem.

If the testing is unsuccessful or the judges return unfavourable judgement, the team has to redesign and reimplement the submitted solution.

1.1.2 Refeefree

Refeefree consists of one server and two clients (contestant and judge). The server is written in C++ and the clients, in Java, supported by a native component written in C++. It runs on UNIX platforms (i.e. Solaris) and supports programs written in C, C++ and Java.

For the contestants, the contestant client is of utmost importance. It is where solutions are tested and submitted. The native component is there to help the client do testing; while the server helps with transmitting solution code from contestants to judges. Now, let’s have a closer look at the graphical user interface (GUI) of the contestant client.

![Figure 2: Original Refeefree](image)

This GUI is only available in English. It has 4 tab panels:

1. submit panel: for testing and submitting the solution program
2. submission history panel: to show the history of submission
3. scoreboard panel: to show the current score of the contestants
4. clarifications panel: to show messages exchanged between judges and respective contestants.
The image above shows the submit panel with two child problem panels. Each problem panel consists, in turn, of four buttons:

1. select: to select a program file that contains the solution.
2. test: test the selected file against the predefined test case.
3. submit: submit the selected file to the judge client.
4. fetch tests: reset the testcase back to the predefined test case.

There is only one test case for each problem. Each test case consists of two files, namely, sin and sout. sin stands for standard input, while sout for standard output. Each test case is copied to a directory, named after the associated problem name. And this directory is located in the directory, where the client program is found.

The contestants may modify the content inside the test cases before pressing the test button. When they press fetch tests button later, the modified test case will be overwritten immediately by the original from a remote test base. The test base is specified in the client configuration file; it is a remote address that holds all the predefined test cases.

1.2 Usability

Now, we have a rough idea how an actual programming contest is run and what functions are offered by the original Refereefree. How useful is Refereefree in helping to run the contest? We need a definition of usability to guide us in measuring that. There is an important standard definition of usability, namely, [7] ISO 9241-11.

The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

The specified users are the first semester students of TU Darmstadt; while the goal is to assist them in developing and submitting the solution program. From the definition, we derive three criteria to measure usability, namely, effectiveness, efficiency and satisfaction.

1. effective: the users can easily figure out on their own how to use the functions inside the software to achieve these specified goals.
2. efficient: these functions take relatively short time to complete.
3. satisfaction: it can be adapted closely to the profile and preferences of each individual user.

1.3 The usability of Refereefree

1.3.1 Effectiveness

The functionalities of the contestant client can be best captured in the Use Case Diagram below:
All these functions are easily accessible through GUI. However, for those not familiar with English language, it is another story. Moreover, the labelling of the *fetch tests* button is quite misleading. It sounds as if the users must *fetch* the tests before being able to initiate the testing process. Features and functions inside a software are useless, unless the users know how to use them. It is therefore important to make the interface as intuitive as possible; the best is that the users can use the software even without consulting the manual.

### 1.3.2 Efficiency

The efficiency of Referee in carrying its functions is decisive in a programming contest, where the contestants must run against time in completing their solution for each problem. Those functions available on the problem panel are especially important, because they contain functions that are critically needed in the development cycle of a solution program. The development cycle of a solution program is typically *initialise-design-implementation-testing-submission*

**Modelling the contestants using Referee** We distinguish two types of contestants regarding the use of Referee during the contests.

The first type of contestants carry out a series of self-defined test against the solution through several cycles of test and initialisation. For every self-defined test, he accesses

Figure 3: Use Case for original Referee
and modifies the predefined test case, and then test it on the problem panel. If he wants to test the solution against the standard test case again, then he has to use the *fetch tests* button. His self-defined test case is overwritten and hence lost.

On the other hand, the second type of contestants do not exploit this testing process. They find the entire process of making self-defined test case too tedious and time consuming. Once they pass the standard test on their computer, they submit the solution to the judges. In another word, they rely extensively on the judges in the submission process to test the solution. Every submission takes at least several minutes before being evaluated by the judges. This increases significantly the development cycle of their solution programs.

A manual test panel with test case creation and selection functions could have helped the contestants do more tests by themselves in a more efficient manner.
Initialisation In the initialisation phase, a standard input and output file are copied to the client from the server. These files are already generated and saved at the server for each problem before the contest. The contestants are expected to include in their code an IO routine so that the input file could be read and processed to produce an expected output. Since the contest emphasizes more on algorithmic thinking rather than programming language, it is certainly a good idea to dispense with the IO hurdle in every contest problem. The contestants can then concentrate on solving the core of the problem.

Design and Implementation As seen from the Problem Panel, there is nothing in the client that supports design and implementation. The contestants have to create and edit their solutions outside the client. It is assumed that contestants can find a suitable editor in the given computer, which runs on UNIX. Any student who takes the first semester computer science course should have enough UNIX knowledge to find a suitable editor. For those students not yet initiated into the world of UNIX, the contest organising committee may have to provide extra human resource just to get them started.

Furthermore, the choice of programming language allowed in the contest is restricted to Java, C++ and C. Although most students learn Java in the first semester, it may not be the programming language that we are most familiar with. Ideally, the contestants should not be restricted at all in the choice of programming language, since testing the skills in a certain programming language is not the objective of the contest.

Testing Referee returns a test display panel only at the end of the testing process. In another word, the entire compiling and executing process is invisible to the contestants until the display panel pops up. The contestants can therefore not anticipate the output. With a real-time display of test output, the design/implementation process and the testing process can then take place in parallel, thus reducing the development cycle.

![Diagram](image-url)

Figure 6: Users anticipate test output
Clearly, real-time display of test output enables contestants to think about the next step of action, before the testing comes to a full stop.

Furthermore, the solution may run into an infinite loop and does not return at all. There is already a time-out function in place. The time-out can also be adjusted. Nevertheless, the testing process may take up quite some time, since it includes compiling and executing the solution code, and reading the output back to the client. Worse still if the problem requires a relatively long time to solve, even with the best solution. The best course of action is to allow the contestants to abort the testing process, as the need arises.

This simple dialog model reflects the way a contestant interact with Refree. Once a command is issued, he has to wait until its execution comes to an end by itself. The contestant has no control over the software during this period of time. On the other hand, the software hides the process of computation to the contestant. Only completed output is shown at the end. Software transparency and user control are very lacking in this model. The future developer may create a display panel, that shows real-time testing process, together with a test abortion button on it.

1.3.3 Satisfaction

Some contestants may prefer a German GUI, a different compiler or a different programming language. Refree is not configurable in all these 3 aspects. Contestants, not used...
to UNIX environment, may waste much time in finding an editor for writing the solution code. Furthermore, the contestant client must always be started with connection to server. The contestants may want to practise using the client at home, before the contest.

All these shortages are still tolerable, as long as the contestants are students from TU Darmstadt. The real motivation for a new Referee to make up for these shortages comes, in fact, from the plan to open the contest to pupils from secondary schools.

We shall in the next chapter elaborate on the ideas that guide the design of the new Referee, that is more effective, efficient and provides more satisfaction to these new users and the existing users.
2 Recreate Refefree for use in schools

2.1 A new plan

In the near future, the contestants will include pupils from secondary schools. How should the developer of Refefree responds to the new demands raised by these new users? Besides, it is clear from the last chapter that Refefree has much to be improved upon, even with respect to the original user group. In this chapter, we shall make new specification and design for Refefree in a systematic way, in response to these new challenges.

2.2 The guideline to usable design

We have introduced in the first chapter the idea of usability and shown how it guides us in gauging the usability of Refefree. Similarly, we shall introduce another ISO standard that provides a guide to usable design. In ISO 13407, a guideline towards an usable program is illustrated in Figure 8.

![Usability diagram](image)

Figure 8: Usability diagram
The activities can be briefly described as follows [7]:

Understand and specify context of use. Identify who uses the program, on which OS should the program run, what the program is used for.

Specify the user and organizational requirements. Determine the success criteria of usability for the program in terms of user tasks. Determine how much resources are available for the project at hand.

Produce design solutions. Incorporate HCI (Human Computer Interaction) knowledge of visual design, interaction design, usability into design solutions.

Evaluate designs against requirements. Reflect on how much resources are actually used for the project and if the user requirements are satisfied by the program created.

2.2.1 Understand and specify context of use.

Referefree is to be used in a programming contest, whose participants include both students and pupils from around the city, Darmstadt. Referefree should be able to run on Microsoft Windows.

2.2.2 Specify the user and organizational requirements.

We identify the needs of our new users:

- Solution templates.
- Open editor from GUI
- Manual testing.
- German graphical user interface.
- Support for Pascal.
- Create an offline version of Referefree
- Port Referefree to Windows.

New features should be included to optimise the workflow of contestants.

Support for Pascal is necessary, since it is the only programming language commonly taught in schools. Portability to Windows is also important, since it is probably the OS that they are most familiar with. The offline mode is a light version of Referefree for them to practise at home. German GUI breaks down the language barrier that the original Referefree may have. The other features are aimed to make Referefree more usable, to make up for the shortages exposed in the first chapters.

The organizational requirements consist of human resources and cost required to develop the new Referefree. Only one student is required to develop it. The cost, arising mostly from software licenses, is kept to minimum by using open source software.

2.2.3 Produce Design solutions

The design of each feature will be elaborated in the following chapters.
2.2.4 Evaluate specifications against requirements.

Evaluation of the software may be collected from the feedback made by the contestants during the contest. Through the clarification panel, they can pose questions to the judges. Some of these questions may shed light upon some usability issues of Refefree.

For the offline mode, there is no integrated function to facilitate feedback from the users. The future developer of Refefree may build a function in Refefree to encourage exchanges between offline users and the Refefree support team.
3 The new Refefree

3.1 Overview of the new Refefree

Let’s have a look at an overview of the new Refefree system, centering on the contestant client.

![New Refefree Overview](image)

Figure 9: New Refefree Overview

The green components are newly added to Refefree system, while the blue components are modified from the last version. A set of problem zip files are created for offline version of Refefree; each zip file contain one test case and 4 solution templates. The new native component supports Pascal programming language. The format of both configuration files are modified to carry more information on each problem and programming language,
such as the choice of compiler and compiler parameters for each language and the time limit of each problem. The server is modified so that these extra information can be transmitted to the clients in online mode. The new functionalities are captured in the Use Case Diagram in Figure 10.

![Use Case Diagram for new Referee](image)

Figure 10: Use case diagram for new Referee

### 3.2 Offline mode

**Motivation** In the original Referee, the client can not be started without connection to server. An offline version of Referee enables the contestant client to stand alone. Potential contestants can practise using the contestant client before coming to the actual contest.

**Design** The client reads from the configuration file a test base, from which problem-specific test data can be downloaded. In offline version, a set of local problem zip files is offered as a substitute. In each zip file, a standard input, standard output and template are included. (More details on template will be mentioned in later section.) The zip files must be placed in the current directory, where the client is located, else an error message will be shown.

Only the submit panel is created upon startup in offline mode and the submit buttons on all problem panels are dimmed permanently.
Starting the client in offline mode  To get the client running in offline mode, click the offline button on the Login Dialog, without giving the password and user name. Alternatively, one can set the client to be offline inside the client configuration file.

![Login Dialog](image)

Figure 11: login dialog

Afterwards, the client automatically searches for existing folders that contain test cases and templates. For each problem, there is a folder that contains a test case and 4 templates. If any folder is missing, a new one is created from the zip files. If any file inside these folders are missing, a new file is copied again from the zip.

“create source code from template” and “reset automatic test input” retrieve template and I/O data from the corresponding zip files respectively.

![Problem Panel](image)

Figure 12: problem panel

Implication for the online mode  In the original version of Refereee, the standard I/O data is saved in a directory named after the problem name at a remote address. In the new Refereee, standard I/O data and template must instead be saved in a zip file named after the problem name at the remote address.

Suggestions for further improvement  An entry in client configuration file may be included to indicate the path of the zip files.
3.3 Support for Pascal

Besides C, C++, and Java, the new Refereef supports Pascal. An open source Pascal compiler, fpc, is tested to be compatible with Refereef. Beside fpc, dcc32 compiler is also compatible with Refereef in supporting Pascal programming language.

Motivation  There are two well known compilers for Pascal, namely, fpc and dcc32. The user can choose between these two. The default compiler is fpc. fpc is preferred to dcc32, because it is available as free software and has several advantages from other perspectives. It is essential that the user of Refereef saves their Pascal program with .pas extension, regardless which Pascal compiler is used. With support for Pascal compilers, the users can test their Pascal programs using Refereef.

3.3.1 Problem with dcc32

If the user chooses the dcc32 compiler, then we are encountered with an output problem, as explained below. The dcc32 compiler always outputs trademark information which is impossible to turn off. The argument Q tells dcc32 to compile quietly. In spite of that, the trademark information is displayed.

[console output]
Borland Delphi fuer Win32 Compiler-Version 18.
2005 Borland Software Corporation
48 Zeilen, 0.06 Sekunden, 62124 Byte-Code, 16248 Byte-Daten.

Therefore native component returns this trademark information to Refereef during testing. On the other hand, fpc compiles silently with the parameter -v0. Such output always deviates from the expected output of a solution; hence the judges requires more time to evaluate the solution. This puts all Pascal contestants in disadvantages, since they always a feedback from the judges later than the other contestants.

Attempt  Here is an attempt to remove the trademark information is made. Although it works for the dcc32 compiler of this author, still more research and tests should be done on other version of dcc32 compilers.

Once the client receives an output file from native component, it ascertains that the compiler is dcc32. If true, it runs the dcc32utility, as attached in at the end of the thesis, to remove the trademark before sending it to the test display panel.

3.3.2 dcc32 vs. fpc

Here the advantages and disadvantages of the two compilers are compared.

fpc  is an open source compiler available for more platforms than most other Pascal compilers and allows easy cross-compiling. And there is work going on for even more platforms and processors. Additionally, a RAD IDE called Lazarus, is built on fpc. On the other hand, it is not fully compatible with Pascal code. Download available at http://www.freepascal.org/download.var.
**dcc32** is available as a part of the Borland Pascal IDE. The latest version of Borland Pascal IDE is Pascal 7. It fully supports Pascal code. However, it is a commercial compiler and runs only on Windows platform. Free trial available at [http://www.codegear.com/Default.aspx?tabid=138](http://www.codegear.com/Default.aspx?tabid=138)

### 3.3.3 Suggestion for improvement

To allow users to change the compiler at runtime. A possible design is to build a new tab that contains all compiler configuration. The path and compiler option for each language are placed in edit boxes. On startup, the edit boxes are filled with the relevant details from the configuration files. When there are changes in their content during runtime, the client will update the *Language* object of its new compiler attributes. A compiler setter method can be easily added to the Language class to enable compiler change during runtime. These new changes will also be saved in the configuration file during runtime.

### 3.4 Optimising the workflow

In the first chapter, we have mentioned several possibilities of increasing the efficiency of Refereefree. Some of them are implemented in course of this thesis.

#### 3.4.1 New Problem Panel

The design of problem panel reflects the workflow of contestants. On the first column, contestants may start with editing a template or an existing file. On the second column, testing can be carried. On the last column there is a submit button to send a solution program to the judges.

![Figure 13: New problem panel design](image)

All the buttons are renamed, to convey a clear meaning of each of their functions. In addition, in the original refereefree, the fetch test button prompts twice for overwriting a test case, one for the standard input and another for the expected output. We see no reason why the users would want to overwrite one and not the other; therefore the current *reset automatic test input* button only prompts for confirmation once.
Submit Button  The submit button forces the client to exit, if it is pressed before or after contest. This button is now by default dimmed. It is undimmed only when the contest is running.

A contest status variable is kept in each problem panel. It is updated whenever the judge changes the contest status. If this variable is set to running and the user successfully selects a solution file using the file selector, then the submit button is automatically undimmed.

3.4.2 Template

Templates are generated together with the predefined test cases on startup. The IO routine is included inside the templates, so that contestants can concentrate on solving the core of the contest problems. These templates are available in the four official programming languages in the contest, namely, C, C++, Java and Pascal. A set of sample template for a contest problem is attached at the end.

They are saved inside a problem specific zip file. These zip files also store the test cases. When the client is in offline mode, all problem zip files must be available in the current directory, where the client is located. When the client is in online mode, all the zip files must be available on the test base. The test base is specified in the configuration file.

Design  The client starts a Login Dialog. In online mode, when login data is entered, the dialog will send a signal to the client. The client starts fetching the test data and
templates from zip files, when the login is successful. In offline mode, the login signal is always set to be successful, since no login data is required at all for offline mode.
3.4.3 Open editor from GUI

There is an *Open Editor* button built into the new problem panel. It opens the selected file with an editor. The path of the editor is specified in configuration file, where each programming language can be assigned a distinct editor. If no file is selected, the user is asked to choose an official programming language and then a solution template in that language will be opened in the editor. If a file is selected, the choice of programming language is determined from the file extension and a suitable editor is opened. For example, if fibonacci.java is selected, then an editor for Java will be opened.

![Diagram showing Open editor from GUI](image)

Figure 15: Open an editor from GUI
3.4.4 Manual Test

The new GUI has a manual test button. It opens an edit panel to take user defined test input.

![Figure 16: Editor for manual test input](image)

![Figure 17: Manual test output display](image)
Design  A sequence diagram of the manual test function is shown below.

![Sequence Diagram of Manual Test Function](image)

**Figure 18: Manual test sequence diagram**

It should, however, be noted that the self-defined test cases do not include the proper output, as the predefined test cases do.
3.5 German graphical interface

The client offers GUI in two languages: English and German. The users can choose the preferred language both at startup and runtime. In the client configuration file, the startup GUI language is specified. In runtime, the two radio buttons on top of the client enable runtime change of GUI language.

![Figure 19: English GUI](image)

![Figure 20: German GUI](image)
Design Two Java libraries are used in the class `lang`:

```java
java.util.Locale;
java.util.ResourceBundle;
```

Two resource bundles are created for Referee: `MessageBundle_de.properties` and `MessageBundle_en.properties`; each contains all the externalised strings of a particular language. Using the libraries `Locale` and `ResourceBundle`, an appropriate property file is chosen, depending on which GUI language is selected by the user. In the implementation, each string that appears on GUI and in exception handling is externalised in language property files, and replaced with a corresponding key, `Lang.getString(stringname)`.¹ The language option radio button update `YAJSContestantUI.java`, the main frame, of the chosen GUI language. Then all the child panels are updated, one after another, until every GUI element is updated of its textual content.

![Sequence diagram for updating GUI language](image)

Figure 21: Sequence diagram for updating GUI language

This diagram shows how the label of the submit button is updated.

¹Externalisation of string is easily done in Eclipse. On the Menu bar, go to `Source`, then choose `Externalize Strings`...
4 Using the configuration files

4.1 Configuration files

In the client package, one finds client_config.xml, where the client can be configured. The following fields are configurable:

- **<server>** : server attributes
- **<GUILanguage>** : Language used for GUI
- **<test-base>** : test-base for online mode
- **<mode>** : can either be online or offline
- **<problem>** : problem attributes
- **<language>** : language attributes

In the server package, one finds serverconfig.xml to configure the server. Below are the fields one can configure:

- **<contest>** : the contest attributes
- **<language>** : the same content as the client configuration file, but in slightly different format.
- **<problem>** : the same as in client configuration file
- **<contestant>** : the login name and password for contestants
- **<judge>** : the login name and password for contestants

There are some overlapping with the client configuration. When the client is online, the specification of programming languages and problems follow the server configuration file.

**Starting the contestant client from console** with java -jar referee.jar will cause the default configuration file, client_config.xml, to be read. If another configuration file is to be used, append the name of the file : java -jar referee.jar configurationfilename.xml. Double click on referee.jar is equivalent to the command, java -jar referee.jar. The judge client can be invoked with java -jar referee.jar configurationfilename.xml judge. The configuration file name must be given in this case.

**Online or offline** Contestant can choose an online mode by simply logging on with a correct username and password. For offline mode, simply click offline button on Login Dialog. Offline button is dimmed for judge client. There is no offline version for judge client.

**Test base**

- **<test-base>** http://www.sample.de/private/problemSet/</test-base>

It is only relevant in online mode. The client downloads the test data from the given address instead of the local zip files. Only http and file format are supported here.
Problem

<problem>
  <id>Test</id>
  <name>Test</name>
  <time-limit>10</time-limit>
</problem>

The template name and the solution file must be named after the problem ID, while the zip file name and the directory name of that problem after the problem name. Beware that Linux is case sensitive and wrong case may cause reading failure. Time limit 10 means 10 seconds before program timeout. To avoid confusion, make both name and id the same, and name the problem zip file, problem directory and template after it.

Note: more tests have to be done across various OS, to check if the problem name and ID can contain special characters.

The extension of a template in a certain programming language follows the language ID of that language. For example, the language ID of Pascal is pas, so a template in this language must have pas as its file extension. More details on language attributes below.

Language

Below is a section configuration file describing the Pascal language.

<language>
  <id>pas</id>/file extension
  <name>pascal</name>/programming language
  <compiler>
    <name>fpc</name>
    <path>C:\Programme\fpc\bin\fpc</path>
    <option>-v0 -v0</option>
  </compiler>
  <editor>c:\WINDOWS\system32\notepad</editor>
</language>

The ID correspond to the template file extension as well as the solution program file extension. The client automatically recognises a given file as belonging to a particular language by comparing the file extension with the language ID. The native component also make use of the language ID to compile a solution file accordingly. Only “pas”, “cc”, “c” and “java” are recognised by the native component.

Editor

Please specify the complete path of your favourite editor here. For example:

<editor>c:\WINDOWS\system32\notepad</editor>

Compilers

If the 3 attributes of the compiler, in the example above, are left empty, then the default compiler fpc will be invoked. For path attribute, it is best that a complete path be given. If the compiler is in system path, then “fpc” may suffice.

<compiler>
  <name>fpc</name>
  <path>C:\Programme\fpc\bin\fpc</path>
  <option>-v0 -v0</option>
</compiler>
File separator must be given as a double backward slash.

For C and C++, if a path is given, one must always add `-o` at the end of a list of compiler options.

```xml
<language>
  <id>cc</id>
  <name>c++</name>
  <compiler>
    <name>g++</name>
    <path>c:\cygwin\bin\g++</path>
    <option>-xc++ -O2 -Wall -o</option>
  </compiler>
  <editor>c:\WINDOWS\system32\notepad</editor>
</language>

<language>
  <id>c</id>
  <name>c</name>
  <compiler>
    <name>gcc</name>
    <path>c:\cygwin\bin\gcc</path>
    <option>-xc -O2 -Wall -o</option>
  </compiler>
  <editor>c:\WINDOWS\system32\notepad</editor>
</language>

Lastly, we consider java. One adds `-d` to the end of compiler options. Below is how jikes can be used.

```xml
<language>
  <id>java</id>
  <name>java</name>
  <compiler>
    <name>jikes</name>
    <path>jikes</path>
    <option>–bootclasspath c:\jre1.6.0\lib\rt.jar –d</option>
  </compiler>
  <editor>c:\WINDOWS\system32\notepad</editor>
</language>
```

We shall elaborate further on the particularities of the compiler configuration in the next section.

### 4.2 Compilers

Taking a look at the client_config.xml, one can see that each problem has attributes called compilers, compiler option etc. Indeed, in the previous version, only javac, g++ and
gcc are allowed in the Refereee system, and they are hard coded in the native component. Now, besides javac, gcj and jikes are also compatible with the new Refereee. The choice of compilers can be modified in the client or server configuration, depending whether an offline or online version is used.

Care must be taken regarding the output compiler switch. The native component creates a temporary directory, goes inside it, builds the executable and then runs it there. Since it makes no assumption about how output switch looks like for each compiler, one has to specify it in the compiler option. Else, the executable will be created in where the solution program is located, in case of C++, C and Java; and the native component would not be able to find the executable in the temporary directory. The testing will fail.

For javac, it is -d which indicates the directory that shall contain the executable of the solution program. The user must append -d, in case of javac, to the list of compiler switch, else the testing will fail. gcj and jikes also have compiler option -d, which has exactly the same semantic meaning as that of javac. Other Java compilers to be used with Refereee must possess such output switch, though it must not necessarily be -d syntactically.

Similarly, for g++ and gcc, the user must append -o to the list of compiler switch. If other c++ or c compilers are used, an equivalent of -o must be appended.

For Pascal, it is a different story. One must not add the output switch. Because both compilers generates an executable in the current directory, not in the directory where the solution program is located. Only those Pascal compilers with the above mentioned property is compatible with the native component. The native component does not anticipate an output switch from pascal compiler. If an output switch is added, there will be an error returned by the native component.

The compiler path, entered inside configuration file, will be used by the native component to call the compiler. Default compilers, when no compiler attributes are detected, will be used. Below is a list of default compilers with compiler parameters.

1. Java : javac -d
2. C++ : g++ -x c++ -o2 -Wall -o
3. C : gcc -x c -o2 -Wall -o
4. Pascal: fpc -v0

### 4.3 Known bugs

To instruct fpc to compile silently, one uses -v0 -v0, because the native component may sometimes, due to unknown reason, ignore the first entry.
5 Installation Guide

5.1 Pre-installation check-list

We have the following installed on computer.

1. Java JDK 1.5 http://java.sun.com/javase/downloads/index.jsp
2. Free Pascal Compiler www.freepascal.org

For Windows user, go to www.cygwin.com/setup.exe. Select under the category “Devel” make 3.81, gcc-g++ 3.4.4-1 and gcc-core C Compiler. The Cygwin setup client will automatically select all those associated utilities. Do not deselect them.

5.2 Performing the installation

1. use make to build up native component and move native.exe to the directory that contains referefree.jar
2. use make to build up the server overlord. In Solaris, type make -f Makefile.solaris to build the server.

Remark: if we double click on the referefree.jar to start the program, the manual button may not work under some versions of java runtime environment. In that case, start the program from console.

5.3 Starting the program

1. Go to the server directory in the console(i.e commandline)
2. Type overlord -c serverconfig.xml
3. run referefree with this command: java -jar referefree.jar
4. Type in team1 as both password and name
5. Click on login button

If offline mode is desired, skips step 4 and 5 and click on offline button directly.
6 Conclusion

In this thesis, we notice that the usability of a software depends to a large extent on who its users are. Every software developer creates a software with a specific target user group in mind. Every function within the software is made because the developer thinks that the intended users have both the ability and inclination to use it. Some functions may be still in early stage of development and need to be improved upon. Take for example the testing function of Refefree which indeed provides basic testing function, but falls short of various useful features that the contestants would like to have.

As the target user group expands to include pupils from secondary schools, Refefree can no longer cope with the new demands raised by these new users. It has to be able to support Pascal, to run on Windows and have an interface that does not demand too much computer experience.

The new Refefree is one step closer towards the expectation of this enlarged user group. The improvement suggestions made in this thesis could perhaps shed light upon what the next version of Refefree may look like. And certainly there are still many possibilities of improvements not yet considered in this thesis and remain to be discovered, perhaps among the user feedback.
7 Acknowledgement

Here I would like to express my deep gratitude towards my supervisors Martin Girschick and Dr. Roessling Guido, for their encouragement and help. Their support is indispensable for the completion of this thesis. Martin gives me valuable advice on the design of the problem panel.

The user anticipation diagram(Figure 6) and the simple dialog model(Figure 7) are taken with modification from the book, [2] *Display use for Man Machine Dialog*. 
8 Templates

Templates to handle IO routines  Here are the templates written in various pro-
gramming language to ease the process of solving [9] Stamps problem. The contestants
need to answer how many friends have to be asked in order to collect a certain amount
of stamps.

The standard input is:

```
2
100 6
13 17 42 9 23 57
99 6
13 17 42 9 23 57
```

2 means the number of scenarios. 100 means the number of stamps need to be collected.
6 means there are six friends. In the third line, the six number represents how many
stamps the six friends have.

The expected output is:

```
Scenario #1:
3

Scenario #2:
2
```

In these templates, the IO routines are taken care of.
Java

Java template
[Stamps.java]
import java.io.*;
import java.util.*;
class Solution {
    int requiredNumberOfStamps;
    int[] stampsOfFriends;

    //make a StreamTokenizer object of the standard
    // input stream.
    static StreamTokenizer in =
        new StreamTokenizer( new InputStreamReader( System.in ));

    //return a token from the StreamTokenizer object
    static int readInt () throws Exception {
        in.nextToken();
        return (int) in.nval;
    }

    //The first parameter is the required number of stamps
    //The second parameter is an array of friends.
    //If stampsOfFriends[0]==3, then it means the first
    //friend has 3 stamps.
    public static void solve
        (int requiredNumberOfStamps, int[] stampsOfFriends) {

        //write your solution
        //here and output the result via System.out.println
    }

    public static void main ( String[] args )
        throws Exception {
        //read the number of required stamps
        int requiredNumberOfStamps=readInt();
        // read the number of friends
        int numberoffriends=readInt();

        //store in an array how many stamps each friend has
        int[] b = new int[ numberoffriends ];
        for( int i=0; i< f; i++ )
            b[i] = readInt();

        //solve the problem!
solve(requirednumberofstamps,b);
}
C++

[Stamps.cc]
#include <iostream>
#include <string>
using namespace std;

void solve(int requirednumberofstamps,
    int[] stampsoffriends){
  //write the solution here.
}

int main () {

    //--- Read testcase data.
    int need, f;
    //read the number of stamps needed and
    //the number of friends
    cin >> need >> f;
    //create an array of size f.
    int s[f];
    //assign the array
    for( int i=0; i<f; i++ )
        cin >> s[i];
    //solve the problem!
    solve(need,s);
}
Pascal

[Stamps.pas]
program Solution;
uses contnrs;

var n,f: LongInt;
var b: Array [1..60] of integer;

//The first parameter is the number of stamps needed.
//The second parameter is the number of friends.
//We assume 60 to be the upper limit of number of friends.
procedure solve (requirednumberofstamps
:longInt;stampsOfFriends: Array [1..60] of integer);
//write the solution here.
begin
  //--- Read testcase data.
  //read the number of stamps needed
  Read(n);
  //read the number of friends.
  Read(f);
  //assigning the array of friends
  // b[1]==3 means the first friend has 3 stamps.
  //b[2]==6 means the second friends has 6 stamps.
  FOR I:=1 TO f DO
    BEGIN
      Read(b[I]);
    END;
  //solve the problem!
  solve(n,b);
end.
9 dcc32utility.java

dcc32utility.java  This class starts by confirming that the first two lines of output contain keyword such as “Borland Delphi” and “Borland Software Corporation”, then it skips to the fourth line to start reading the actual output of the solution program. The third line is not checked, due to difficulty of different translations of “rows”, “seconds” and “data”.

```java
public static File compilerInfoFilter(File file){
    String ln1="Borland Delphi";
    String ln2="Borland Software Corporation";
    try {
        File outputFile=File.createTempFile("dcc32", "out");
        BufferedWriter out =
            new BufferedWriter(new FileWriter(outputFile));
        LineNumberReader rd=
            new LineNumberReader(new FileReader(file));
        if(rd.readLine().contains(ln1))
            if(rd.readLine().contains(ln2)){
                rd.readLine();
                out.write(rd.readLine()+"\n");
                String str;
                str=rd.readLine();
                while(str!=null){
                    out.write(str+"\n");
                    str=rd.readLine();
                }
            }
        out.close();
        return outputFile;
    }catch (IOException e) {} return file;
}
```
10 Literature

References


