ITC-irst, Centro per la Ricerca Scientifica e Tecnologica

STAR project – Software Technology Advanced Research

C++ Code Analysis Tool

User Guide

Paolo Tonella Alessandra Potrich

February 7, 2002

Contents

1 Installation **3**
- 1.1 Code analysis 3
- 1.2 Web interface 3

2 Directories **5**
- 2.1 Code analysis 5
- 2.2 Web interface 6

3 Configuration **7**
- 3.1 Code analysis 7
  - 3.1.1 File `config_ANALYSIS` 7
  - 3.1.2 File `config_SUFFIXES` 8
  - 3.1.3 File `config_RULES` 8
  - 3.1.4 File `config_FILTERING` 8
  - 3.1.5 File `config_DB` 9
  - 3.1.6 File `config_CONTAINER` 10
- 3.2 Web server 10

4 Execution **13**
- 4.1 Code analysis 13
  - 4.1.1 Rule checker 13
  - 4.1.2 Reverse engineering 14
- 4.2 Web server 16

Chapter 1

Installation

1.1 Code analysis

To install the code analysis tool, open the tar file (*IRSTCodeAnalysis-Tool.tar.gz*) in the installation directory (called *$INSTALL_DIR* in the following), with the command:

```
tar xzvf IRSTCodeAnalysisTool.tar.gz
```

Then, select one of the user configurations from the subdirectory *userConfig*, by creating a link to it in *$INSTALL_DIR*. For example, people from the ALICE experiment will run the following command:

```
ln -s userConfig/ALICE/ config
```

The tool comes with configuration directories for Alice, ATLAS and LHCb (empty directory to be filled in) experiments. New configurations can be created under *userConfig* and made active by linking them to *config* in *$INSTALL_DIR*.

1.2 Web interface

To install the Web interface for the reverse engineering package contained in the code analysis tool, open the tar *webreveng.tar.gz* in the installation directory (called *$INSTALL_DIR* in the following), with the command:

```
tar xzvf webreveng.tar.gz
```

Chapter 2

Directories

2.1 Code analysis

The following directories under *$INSTALL_DIR* contain the source code of the tool as well as the executable bytecode:

```
entities/
reveng/
rules/
syntax/
```

In particular, the directory `reveng/` contains also some Java libraries for flow analysis (`flow_analysis.jar`) and connection to the MySQL database (`mysql_2_uncomp.jar`), and the executable of `dot`, a program computing the layout of a graph given in the *dot* format.

Directory `userConfig` contains a set of predefined files for tool configurations.

Directory `scripts` contains some utility scripts, like `createViolation-Table.sh` (to be personalized), which creates an HTML page with all violations to the coding rules from a set of violation report files.

Directory `patch` contains the patches (Perl scripts) for the C++ features not fully supported by the tool.

Directory `dyn` contains scripts for the dynamic analyses.

Directory `doc` contains the HTML documentation of the tool automatically generated by `javadoc`.

Directory `man` contains User Guide and Developer Guide.

2.2 Web interface

The main directory (`webreveng`) contains all Perl scripts necessary for the creation of the images of the UML diagrams to be displayed inside the resulting Web pages.

Directory `CGI-BIN` contains the Perl script for the navigation inside the generated diagrams.

Directory `WWW` contains an example of Web site providing the UML diagrams extracted by the tool.

Chapter 3

Configuration

3.1 Code analysis

The configuration of the tool can be modified by editing the files contained in the directory linked to *$INSTALL_DIR/config*. Their format and the effects of the possible customizations are described in the following.

All configuration files in *$INSTALL_DIR/config* can be overridden by creating another file with the same name in the working directory (i.e., in the directory where the tool is run). If present, such a file is read by the tool instead of that under *$INSTALL_DIR/config*.

3.1.1 File `config_ANALYSIS`

This file contains a list of operations that can be enabled (tag `ON`) or disabled (tag `OFF`). It is possible to decide if the statements of functions and methods have to be analyzed, so that the related expressions are determined and made available from the *entities* package. This is controlled by the label `EXPRESSION_ANALYSIS`; if the tag is OFF some rules can not be checked). During reverse engineering, it is possible to decide whether to perform the analysis of the weakly typed containers – a quite computationally expensive analysis – to infer the type of the contained objects and improve the accuracy of the class diagram. The related label is `CONTAINER_ANALYSIS`. It is also possible to decide whether to statically extract the object diagram or not (label: `OBJECT_ANALYSIS`). Finally, it is possible to record into a database the entities visited during reverse engineering, by enabling the operation named `DB_INSERTION`. An example of `config_ANALYSIS` file with expression analysis enabled, and container analysis, object analysis and data base insertion disabled is the following:

```
EXPRESSION_ANALYSIS     ON
CONTAINER_ANALYSIS      OFF
OBJECT_ANALYSIS         OFF
DB_INSERTION            OFF
```

3.1.2 File `config_SUFFIXES`

This file contains the specification of the allowed extensions of the source files. It has two entries, for the specification of the implementation file extensions and of the header file extensions respectively. An example of `config_SUFFIXES` file, declaring that the known extensions are `.cxx, .C, .cpp` for the implementation files and `.h` for the header files, is the following:

```
SOURCE_FILE_SUFFIX cxx C cpp
HEADER_FILE_SUFFIX h
```

3.1.3 File `config_RULES`

This file contains the specification of the coding conventions to be checked, among those available in the subpackage with the specific implementation of *RuleChecker* (for example, among all Alice rules, when running *ALICERule-Checker*). It contains the list of the active rules, one for line. If, for example, only rules `RC10, RC11, RC12` are to be checked, this file will contain:

```
RC10
RC11
RC12
```

As with all other configuration files, a common way to override the specifications provided in `config_RULES` under *$INSTALL_DIR/config* is to create another file `config_RULES` in the current directory. If present, it is read instead of that under *$INSTALL_DIR/config*.

3.1.4 File `config_FILTERING`

Some of the entities present in the C++ code under analysis may be uninteresting, for example because they are generated automatically by some tool. It is possible to exclude them from the analysis by specifying the substrings of their names that allow their recognition. This is done in the `config_FILTERING` file.

To exclude a specific entity, it is sufficient to indicate its name after the label corresponding to its type (e.g., if it is a class, after `CLASS_NAME`). The simbols ^, $, % can be put before the specified strings to indicate that a particular entity has to be removed only if its name respectively starts with that string, ends with that string or contains that string.

For example, in the following case:

```
MODULE_NAME
GLOBAL_VARIABLE_NAME    $R__ ^G__
CLASS_NAME              ^UNNAMED %R__
METHOD_NAME             Class_Name
FIELD_NAME              fgIsA
LOCAL_VARIABLE_NAME
FRIEND_FUNCTIONS_NAME   operator>>
```

no filtering is applied to *modules* and *local variables*. Global variables ending with `R__` or starting with `G__` are filtered out. *Classes* starting with `UNNAMED` (a default name used for the C++ construct *struct*) or containing `R__` are excluded. The field `fgIsA` as well as the friend function `operator>>` are also excluded.

3.1.5 File `config_DB`

The configuration file `config_DB` contains all the information necessary to connect to the data base server to store or access the entities generated by the tool after parsing a source program. The information needed to perform a data base connection consists of host name, data base name, user name, and password. An example of such a file is:

```
Host: pcepaip12.cern.ch
Database: IRSTCppAnalysis
User: IRSTCppAnalysis
Password: IRSTCppAnalysis
```

If the data base management system in use is MySQL, the following commands have to be prompted to create data base and user:

```
mysql> create database IRSTCppAnalysis;
mysql> grant all privileges on IRSTCppAnalysis.* to
       IRSTCppAnalysis@'%' identified by 'IRSTCppAnalysis';
mysql> grant all privileges on IRSTCppAnalysis.* to
       IRSTCppAnalysis@<mysql-host> identified by 'IRSTCppAnalysis';
```

where `<mysql-host>` is the name of the host where MySQL is running.

To create the necessary tables in the database, type the command:

```
java entities.EntityDBManager create
```

with *$INSTALL_DIR* and *$INSTALL_DIR/reveng/mysql_2_uncomp.jar* in the `CLASSPATH`.

3.1.6 File `config_CONTAINER`

When the container analysis is activated, the reverse engineering module needs some information about the weakly typed containers in use, in order to succed determining the type of the contained objects. All insertion and extraction methods available from the container library have to be listed. Moreover, the source and target of such operations have to be specified. If the operation is an *insertion*, the source is the object to be inserted, while the target is the container. Vice versa for an *extraction* the source is the container while the target is an object (typically, reference or pointer).

When the source/target of an operation is the object on which the operation is invoked, it will be indicated as `this` in the configuration file. When it is a parameter, the convention is to indicate `param-` followed by an index (starting from 0 for the first parameter and incremented for the next ones). Finally, if the extracted object is returned, the keyword `return` is used as target.

For example, the following configuration file:

```
// <class>     <method>  <source>    <target>   <type>
TList          Add       param-0     this       insertion
TList          At        this        return     extraction
```

specifies that method `Add` of class `TList` is an *insertion* method, which puts its first parameter into the container on which the method is invoked. Method `At` of the same class returns the object extracted from the container on which the invocation is made.

3.2 Web server

In order to set up a Web server that provides access to and navigation inside the UML diagrams extracted from the code, it is necessary to follow these steps:

1. Preparation of the static data (diagrams, static pages, etc.).
2. Set up of the server programs for the dynamic access.

To prepare the static data, it is possible to copy the home page of the site `WWW/HomePage.html` from `webreveng` into the proper directory (`reveng` in the following) and rename it, so as to make it accessible as the initial page of the Web site.

The following line should be edited:

```
<BASE HREF="http://zeus.itc.it:4444/">
```

so as to refer to the base URL of the site;

A directory for each subsystem should be created, starting from the initial directory (e.g., `reveng/STEER` for the detector[1] STEER, etc.). The page `WWW/STEER/HomePage.html` should be copied into every subsystem's directory, and edited so as to contain the proper subsystem name and location. The following lines should be edited:

```
<BASE HREF="http://zeus.itc.it:4444/">
```

so as to include the base URL for the site;

```
<A HREF=cgi-bin/reveng/focus.prl?detector=STEER>
```

so as to refer to the actual CGI-BIN directory containing the script `focus.prl` (relative to the base URL, see below). The detector name is an additional parameter, still to be edited

```
<IMG SRC="reveng/STEER/classDiagram.gif" ismap>
```

[1]In the following, detector and subsystem will be used interchangeably.

so as to refer to the directory containing all reverse engineered information for a given detector (subsystem).

A directory `comp` should be created to store the component diagram. The file `WWW/comp/HomePage.html` should be copied into this directory and edited as described above for normal detectors (BASE URL and `reveng` directory).

Finally, the page `WWW/index.html` (the index displayed in the left frame) should be copied into the initial directory (`reveng`) and edited so as to reflect the current installation (actual name of `reveng` directory and BASE URL). The list of detectors may also be edited to add/remove detectors. The initial page in `WWW/HomePage.html` and the first index item in `WWW/index.html` is the component diagram. On Web servers which force the initial page to be named `index.html` (instead of `HomePage.html`), it may be necessary to rename `index.html` so that it does not conflict with it.

Finally, the server program for the dynamic interaction with the UML diagrams can be set up.

The script `CGI-BIN/focus.prl` should be copied into the proper directory (`cgi-bin/reveng` above) under the CGI-BIN directory of the server (i.e., the directory from which the Web server retrieves its scripts).

The following line should be edited:

```
$www_dir = "/ssi0/ssi/revenge/WWW/reveng/";
```

so as to refer to the directory containing the initial page (`HomePage.html`).

Chapter 4

Execution

4.1 Code analysis

4.1.1 Rule checker

To execute Rule checker on a single source file, add the directory *$INSTALL_DIR/* to the environment variable `CLASSPATH`. Then type the commands:

```
g++ -E -I... -D... -o A.i A.cxx
java rules.ALICE.ALICERuleChecker A.i [path]
```

to execute the Rule checker of the Alice experiment. Similar command lines work for the other experiments.

If necessary, the preprocessor has to be invoked with one or more `-I` and/or `-D` directives, specifying the location of the include files that are needed and providing macro definitions for the compiler.

When `path` is not specified, the tool assumes that `A.cxx` and `A.h` are in the current directory. Otherwise, they are assumed to both reside in `path`.

To execute the tool on a list of files, it is necessary to indicate them in a text file conventionally named `config_FILES_TO_ANALYZE`, which can be created either in the current directory or in the general `config` directory (typically, the former case is more practical since several such files are expected to be created by different users and for different subsystems).

Then, the tool is executed without parameters:

```
java rules.ALICE.ALICERuleChecker
```

In order for the execution to succeed, it is also required that a file named `quickExecution` exists in the current directory (it can be created by prompting the UNIX command: `touch quickExecution`). The effect of this file on the execution is explained below.

The file `config_FILES_TO_ANALYZE` respects the following format: each line of text corresponds to one file to analyze. The first two strings in each line are mandatory, and give respectively the name of the preprocessed file `A.i` and the name of the implementation file `A.cxx`. Then, the list of necessary header files follow (e.g., `A.h`). Typically, this is expected to be one file. The name of the violation report file can be specified within asterisks (e.g., `*A.viol*`); if a file is not specified the violations are reported on the standard output. Finally, if any header file name is ambiguous, because different subsystems contain a header file with the same name, it is possible to indicate the specific subsystem within square brackets.

Example:

```
A.i A.cxx [b1/b2] A.h *A.viol*
B.i B.cxx B.h *B.viol*
```

In this case, two files are analyzed (`A.cxx` and `B.cxx`). The output is printed to `A.viol` and `B.viol` respectively. To disambiguate the name of the first header file (`A.h`), the subsystem `b1/b2` is indicated.

In case the header file associated to a given implementation file (say, `A.cxx`) is unique and its name is obtained by changing the extension of the implementation file (`A.h`), it is possible to run the tool in a *quick execution mode*, skipping an initial preprocessing in which the tool determines the header files associated to a given implementation file. Execution times are thus shortened. To achieve this, it is sufficient to create an empty file named `quickExecution` in the current directory (`touch quickExecution`). Since knowledge of the user defined implementation and header files is granted also when `config_FILES_TO_ANALYZE` is used, execution on a list of files requires the quick execution mode.

4.1.2 Reverse engineering

To execute the reverse engineering module, add the directory *$INSTALL_DIR/* to the environment variable `CLASSPATH`. Moreover, the Java archive *flow_analysis.jar* from directory *reveng* has to be added if the container analysis is going to be executed, and the Java archive *mysql_2_uncomp.jar* from directory *reveng* has to be added if data base insertion is active. A typical command which achieves all of this is:

```
setenv CLASSPATH ${CLASSPATH}:${INSTALL_DIR}:
       ${INSTALL_DIR}/reveng/flow_analysis.jar:
       ${INSTALL_DIR}/reveng/mysql_2_uncomp.jar
```

Then the following command can be typed:

```
java reveng.ReverseEngineering A1.i A2.i A3.i
```

to execute Reverse engineering on the three preprocessed files `A1.i`, `A2.i`, and `A3.i`. Typically, Reverse engineering is executed on an entire subsystem. This is achieved by typing the command:

```
java reveng.ReverseEngineering subsys/*.i
```

for the subsystem `subsys`.

The result of the execution is the *dot* file `classDiagram.dot` in the current directory. If data base insertion is active, the data base will be populated with the entities processed during parsing. If object analysis is active, the object diagram is produced by a static analysis of the source code and is stored in the *dot* file `objectDiagram.dot`. Dynamic extraction of the object diagram is described below.

Once the data base is populated with entities associated to the source files in the system, it is possible to extract class diagram and component diagram from the data base, respectively by typing the commands:

```
java reveng.ClassDiagramFromDB file1.h ... fileN.h
```

and

```
java reveng.ComponentDiagramFromDB dir1/file1.h ... dirN/fileN.h
```

In the first case, the name of the directory containing the header files with the declarations of the classes to be analyzed is not important, while in the second case it identifies the subsystem, which is the basic element of the component diagram. Therefore, the indication of the enclosing directories is mandatory in the second case.

Execution of these two commands leads to the generation of the files `classDiagram.dot` and `componentDiagram.dot` respectively. They can be visualized by means of the public domain tool Graphviz by AT&T or by exploiting the Web server *webreveng*, distributed separately.

Dynamic extraction of the object diagram can be achieved by executing the following steps:

1. Edit `$INSTALL_DIR/dyn/gdb.cmd` and add the breakpoints of interest. Objects will be dumped at these points.

2. Run:
   `gdb -x $INSTALL_DIR/dyn/gdb.cmd program > trace.txt`. Terminate execution by pressing `<CTRL> D`.

3. Run:
   `$INSTALL_DIR/dyn/generateObjectDiagramFromTrace.prl trace.txt > dynamicObjectDiagram.dot`.

4. Visualize the graph by means of Graphviz:
   `dotty dynamicObjectDiagram.dot`.

4.2 Web server

All files `classDiagram.dot` should be copied into the subdirectories associated with each subsystem (e.g., reveng/STEER), together with the initial HTML pages (copy and modify `HomePage.html` from `webreveng/WWW/STEER/`). The file `componentDiagram.dot` should be copied into the subdirectory `comp`.

The shell script `create-class-diagram-pages.sh` automates HTML page creation for each subsystem, `while create-component-diagram-pages.sh` automates the creation of the pages for the component diagram.

These two scripts assume that the starting information (file `classDiagram.dot` for each subsystem and file `componentDiagram.dot` for the component diagram) is available. For convenience, they include as commented lines the commands to be issued to produce them.

Now, if the Web server (e.g.Apache) is on, the reverse engineering information can be accessed and navigated.