Columbus: A Reverse Engineering Approach

Árpád Beszédes, Rudolf Ferenc and Tibor Gyimóthy
University of Szeged, Department of Software Engineering
and
FrontEndART Software Ltd.
{beszedes|ferenc|gyimi}@inf.u-szeged.hu

Abstract

In this paper we present our approach to several common problems in reverse engineering that are built around the Columbus framework. Columbus defines several fundamental building blocks for the use in reverse engineering processes, and as such it can be an important player in the studies conducted at the workshop for Empirical Studies in Reverse Engineering. The Columbus framework proved its usefulness in the field through a number of research projects (also by independent researchers) and several industrial applications. Columbus may contribute as (1) a flexible, easily extensible tool architecture, (2) a data exchange model (C/C++ schema) and (3) as a source code analysis process.

1. Introduction

The role of reverse engineering in software evolution is evidently important. However, in order to introduce this discipline into the software engineering process, some fundamental building blocks are needed. Such are the availability of tools, data exchange settlements and a reverse engineering process.

We demonstrate the reverse engineering approach that builds upon the *Columbus* framework [4, 5, 6], which addresses these essential elements, and as such can be an important player in a study conducted to assess the existing approaches and define a general framework for empirical studies. The Columbus technology is by now recognized by the academia, in fact it is one of the reference architectures in the field. In addition to the numerous examples of its usage by research groups, it also plays an important role in a number of industrial projects.

Columbus deals with the issues mentioned above in the following way. First, it includes an extensible reverse engineering tool, currently with a source code analysis module for the C/C++ language. Second, it defines a schema for representing C++ source code as an exchange model among other tools [3, 6, 12]. This schema is also one of the reference representations according to state of the art. Finally, Columbus contributes to the reverse engineering process in the following manner. Although it does not cover all potential aspects of such a process, it deals with some of the most critical issues such as project set-up and fact extraction. We emphasize the importance of the former, since in many situations collecting the artifacts to be analyzed is not trivial, and being a fundamental activity, it affects all remaining parts of the process. The most important aspect that needs special attention of the latter issue, fact extraction, is that it needs special design consideration regarding code analysis, with respect to traditional methods such as those used by compilers.

We overview fact extraction using Columbus in the second section, while data exchange is discussed in Section 3. Section 4 deals with representing the extracted data in another form and providing input to other tools, and in the final section we summarize our contributions to the workshop.

2. Fact extraction with Columbus

To comprehend a software system we need to know many different things about it. We refer to this information as *fact*s about the source code. *Fact extraction* is an automatized process during which the subject system is analyzed with analyzer tools to identify the source code's various characteristics and their interrelationships, and to create abstract representations of the extracted information. The form of these representations is prescribed by *schemas*, which are descriptions of the form of the data in terms of a set of entities with attributes and relationships. A *schema instance* is an embodiment of the schema which models a concrete software system. To make the results of fact extraction widely usable, we further process the schema instances to take various new formats.

Columbus is a reverse engineering framework developed in cooperation between the University of Szeged, the Nokia

Research Center and FrontEndART [9]. The main motivation behind developing this framework was to create a toolset which supports fact extraction and provides a common interface for reverse engineering tasks in general. The graphical user interface of the framework is called *Columbus REE* (Reverse Engineering Environment). Further tools are also incorporated into the framework (mostly command line), which actually do the C++-specific tasks, like analyzing the source code and further processing the results.

The extraction process within the Columbus framework is outlined in [4]. The process is very similar to the traditional compilation process. It consists of five consecutive steps (see Figure 1) where each step uses the results of the previous one.

The steps of the process are the following:

1. Acquiring project/configuration information
2. Analysis of the source – creation of schema instances
3. Linking of schema instances
4. Filtering the schema instances
5. Processing the schema instances

These steps may be performed in different ways: using the visual user interface of the Columbus REE, using the compiler wrapper toolset (see below), or using only the command-line programs by themselves. An important advantage of the presented steps is that they can be performed incrementally, that is, if the partial results of certain steps are available and the input of the step has not been altered, these results need not be regenerated.

Acquiring project/configuration information is indispensable to carry out the extraction process. The source code of a software system is usually logically split into a number of files and these files are arranged into folders and subfolders. Furthermore, different preprocessing configurations can apply to them. The information on how these files are related to each other and what setting apply to them are usually stored either in *makefile*s (in the case of building software with the *make* tool), or in different *project files* (in the case of using different *IDE*-s – Integrated Development Environments).

The Columbus technology employs a so-called *compiler wrapping* technique for using makefile information and two different approaches for handling IDE project files: IDE integration and project file import.

Compiler wrapping. Makefiles can contain not only the references to files to be compiled and their settings but can also contain various commands, like invoking external tools. These powerful possibilities are bad news for reverse engineers, because every action in the makefile must be somehow simulated in the reverse engineering tool. This can be extremely hard or even impossible in some cases. We approached this problem from the other end and solved it by "wrapping" the compiler. This means that we temporarily hide the original compiler, and this way if the original compiler should be invoked our wrapper program will start instead of it, which executes first the original compiler, and second, it invokes our analyzer tools as well. These are invoked with the appropriate parameters in the same environment to build up the required schema instances. This way all we have to do is to build a software system as usual (but with the wrapper switched on).

IDE integration. In this case our tool appears as a new toolbar within the IDE and its operation is very similar to the usual build process. The active project is analyzed and the output can be transformed into any format supported by the Columbus framework. (Currently Microsoft Visual Studio 6.0 and .NET are supported.)

Project file import. The Columbus REE is able to parse Microsoft Visual C++ 6.0 and .NET project files and to import all relevant information from them to be able to analyze the project. All Columbus REE features can be used in this case.

Manual setup. Besides these possibilities the project can be built up by hand also in the Columbus REE (for instance if no project information is available at all). A so-called *Project Setup Wizard* is available to help in this task.

The Columbus environment currently contains a C/C++ source code analysis front end, which is constituted of a specially developed preprocessor and language analyzer. The analyzers were designed especially to meet the requirements of source code analysis in the scope of reverse engineering. The employed technology makes possible, for example, parsing incomplete code, source-complete detailed analysis, and the handling of language dialects.

3. Data exchange

Successful data exchange is crucial among reverse engineering tools. This requires a common *format*, which is applicable in various reverse engineering tools such as front ends and metrics tools. A standard schema must be found. We described our approach to this important topic in [3, 6, 8], which is since then known as the *Columbus Schema for C++*, and it describes the C++ language details. An extension to the C++ schema is the schema for capturing the *preprocessor* related facts as described in [12].

The Columbus schemas capture the C++ and preprocessor languages at low detail (AST) and also contains higher-level elements (e. g. semantics of types). The description of the schemas is given using UML Class Diagrams, which permits their simple implementation and easy physical representation (e. g. using GXL). Their modularity provides additional flexibility for any future extension/modification. The implementation of the classes belonging to the schemas provides an Application Programming Interface for access-

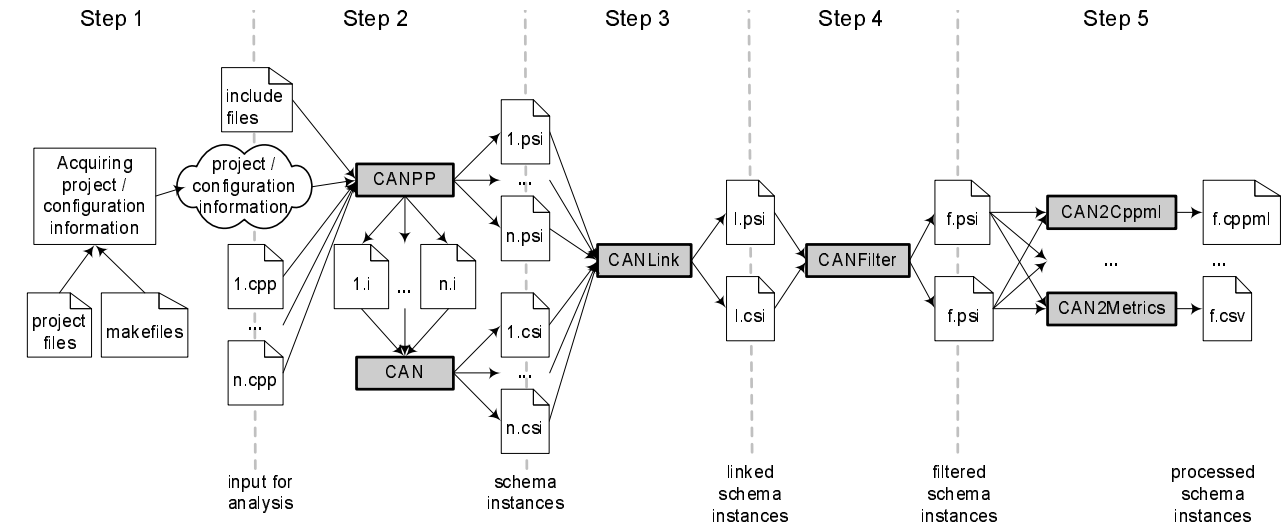

Figure 1. The fact extraction process

ing the internal representation, which is used both by the Columbus framework itself and independent developers as well.

Apart from defining the content of the data to be represented and potentially exchanged with other reverse engineering tools, it is also necessary to settle down common physical formats. To this end, the Columbus schemas can be represented in various external formats, including the increasingly popular GXL format. The external formats are overviewed in the following section.

4. Presentation of data and tool interoperability

Because different re- and reverse engineering tools use different schemas for representing their data, the schema instances must be further processed to achieve tool interoperability. The processing may consist of transforming the schema instance into another format and/or applying further computations on it. Currently the following are included in the Columbus REE:

PPML and CPPML. This transformation permits the creation of XML documents (called *PPML* – Preprocessor Markup Language and *CPPML* – C++ Markup Language) that have structures based on the corresponding Columbus schemas. The exported documents conform to their Document Type Definitions, as described on FrontEndART's homepage [9].

GXL. With this transformation GXL representations can be created from the extracted information. *GXL* (Graph eXchange Language) [10] is an XML-based graph-description format. Since the Columbus schemas basically define graphs, this format is suitable for representing them in a convenient way. The call graph of the analyzed system can be also created in GXL form.

UML XMI. This processing allows the creation of standard UML XMI documents from the Columbus Schema for C++. The XMI document contains the class diagram of the analyzed project which can be further processed with XMI enabled tools (like IBM Rational Rose, Borland Together ControlCenter, etc.).

Famix XMI. With this processing a *Famix* [2] XMI representation of the extracted information can be created. This format can be utilized in the *CodeCrawler* tool for visualization and metric calculations.

RSF. Three transformations are available for creating *rigi* RSF [11] documents: (1) a graph based on the Columbus Schema for C++, (2) a call-graph and (3) a UML class diagram-like graph. All of these use different rigi domains which can be created with Columbus as well.

HTML documentation. This processing can be used to create a hypertext documentation of the extracted project in *HTML* form. The generated documentation presents the project in a browsable and user-friendly fashion. All the necessary information is presented about the classes and other elements in a structured way. Three types of browser frames are also supplied, with which a project can be easily navigated. These present the classes using (1) their names in alphabetical order, (2) the scoping structure and (3) the inheritance relationship.

Apart from these, so to say, simple transformations, we have been experimenting with other kinds of processings of the extracted fact representations. These include the calculation of *object oriented metrics* [7], recognition of *design patterns* [1] and *bad smells* (for refactoring purposes) in the code.

Furthermore, we have developed a special code auditing tool based on Columbus – called *SourceAudit* – which is able to investigate source code and check it against rules that describe the preferred properties of the code. These rules mostly involve issues related to coding style, but in some cases they extend the warning reporting capabilities of the compiler. The checked rules are organized into rule packages and the tool can be freely extended with new packages. The tool can be used in command line and integrated with popular IDE-s (e. g. Microsoft Visual Studio and Borland C++Builder).

5. Relevance to the Workshop

We believe that the Columbus reverse engineering technology may serve as one of the fundamental approaches to be investigated under the scope of the workshop for Empirical Studies in Reverse Engineering. It may contribute as a tool, a data exchange model and as a source code analysis process.

The Columbus Reverse Engineering Environment employs a highly flexible architecture that is essential for tool reuse in various environments where reverse engineering is needed. Its C/C++ source code analysis front end includes the necessary analysis technologies that are specific to the purpose of reverse engineering.

The extracted facts about the source code are stored and further manipulated according to a language schema that, together with common formats like GXL, provides a basis for successful data exchange among reverse engineering tools. Columbus supports a number of external formats that are most commonly used by the reverse engineering community, and this interoperability helped being utilized in several research and industrial projects.

One of the most important assets of Columbus is that it performs the analysis tasks according to a proven source code analysis process. It includes several ways of setting up the reverse engineering projects, because we believe that this is one of the most critical issues in successful code analysis. The compiler wrapping technology proved its simplicity and usefulness in a number of successfully performed analyses of real size software systems, like the open source Mozilla and StarOffice systems.

References

[1] Z. Balanyi and R. Ferenc. Mining Design Patterns from C++ Source Code. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM 2003)*, pages 305–314. IEEE Computer Society, Sept. 2003.

[2] S. Demeyer, S. Ducasse, and M. Lanza. A Hybrid Reverse Engineering Platform Combining Metrics and Program Visualization. In *Proceedings of WCRE'99*, 1999.

[3] R. Ferenc and Á. Beszédes. Data Exchange with the Columbus Schema for C++. In *Proceedings of the 6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pages 59–66. IEEE Computer Society, Mar. 2002.

[4] R. Ferenc, Á. Beszédes, and T. Gyimóthy. Extracting Facts with Columbus from C++ Code. In *Tool Demonstrations of the 8th European Conference on Software Maintenance and Reengineering (CSMR 2004)*, pages 4–8. IEEE Computer Society, Mar. 2004.

[5] R. Ferenc, Á. Beszédes, and T. Gyimóthy. *Tools for Software Maintenance and Reengineering*, chapter Extracting Facts with Columbus from C++ Code, pages 16–31. Franco Angeli Milano, 2004.

[6] R. Ferenc, Á. Beszédes, M. Tarkiainen, and T. Gyimóthy. Columbus – Reverse Engineering Tool and Schema for C++. In *Proceedings of the 18th International Conference on Software Maintenance (ICSM 2002)*, pages 172–181. IEEE Computer Society, Oct. 2002.

[7] R. Ferenc, I. Siket, and T. Gyimóthy. Extracting Facts from Open Source Software. In *Proceedings of the 20th International Conference on Software Maintenance (ICSM 2004)*, pages 60–69. IEEE Computer Society, Sept. 2004.

[8] R. Ferenc, S. E. Sim, R. C. Holt, R. Koschke, and T. Gyimóthy. Towards a Standard Schema for C/C++. In *Proceedings of the 8th Working Conference on Reverse Engineering (WCRE 2001)*, pages 49–58. IEEE Computer Society, Oct. 2001.

[9] Homepage of FrontEndART Software Ltd. `http://www.frontendart.com`.

[10] R. Holt, A. Winter, and A. Schürr. GXL: Towards a Standard Exchange Format. In *Proceedings of WCRE'00*, pages 162–171, Nov. 2000.

[11] H. A. Müller, K. Wong, and S. R. Tilley. Understanding Software Systems Using Reverse Engineering Technology. In *Proceedings of ACFAS*, 1994.

[12] L. Vidács, Á. Beszédes, and R. Ferenc. Columbus Schema for C/C++ Preprocessing. In *Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR 2004)*, pages 75–84. IEEE Computer Society, Mar. 2004.