

Case: Rational Rose
81940 A Seminar on Reverse Engineering
Software Systems Laboratory
Tampere University of Technology
Fall, 2000

1. Introduction

As a part of the TUT Software Systems Laboratory's project *Smart/Kahvinheitin* I studied the Rational Rose's reverse engineering capabilities. This paper is a brief report of that study.

2. What is Rational Rose

Rational Rose is a commercial case-tool software. It supports two essential elements of modern software engineering: component based development and controlled iterative development. Models created with Rose can be visualized with several UML diagrams. Rose also supports Round-Trip engineering with several languages. Here we discuss the code generation and reverse engineering of the C++-language.

3. Why and where was Rational Rose used

The usage of Rational Rose was due to a subgoal of the project *Smart*. The goal was to become familiar with several products of Rational Software Corporation. The retrieved knowledge was also used to hold a presentation on this seminar about reverse engineering. Rational Rose was used as a case-tool in the project *Kahvinheitin* where the idea was to create a software for a microprocessor based coffee maker. Project *Kahvinheitin* can be considered as a subproject of the project *Smart*. In the project Rose was used visually to create class-, state- and packet diagrams. Rose's Round-trip engineering capabilities were also examined.

4. How was the tool used.

The tool was used to create the packet hierarchy, structure and dependencies of classes', state diagram and an example of the control flow (sequence diagram) of the coffee maker software.

C++-code was generated from the created model using the Rose's C++ code generator. This created code was then altered (due to changes to requirements) and reverse engineered to a new model or to an update of the existing model.

5. About the round-trip engineering

The code was generated from the Rose's class diagram. The generation process was quite easy: one needs to select the desired classes, choose destination directory for files and start the code generation. Several properties can be modified to control the code generation. The generated code seemed to be a bit obfuscating – Rose generates it's own comment lines with special tags. These comment lines which help Rose as it reverse engineers these files can look a bit cryptical.

Reverse engineering with Rose requires the usage of a special tool, C++ Analyzer. This tool is a part of Rose's reverse engineering – though it is a separate tool from Rose. With the C++ Analyzer the C++ source files were analysed and reverse engineered. Analyzing was quite simple too. First the files to be analysed were selected. Then the actual analysing was done. After that the user can select if he/she wants to create a normal Rose model (reverse engineering) or an update to an existing model (round-trip engineering). The retrieved output was then ready to be observed in Rose. Also the C++ Analyzer has several properties which can be used to modification of the created models. A view of the C++ Analyzer is shown on figure 1.

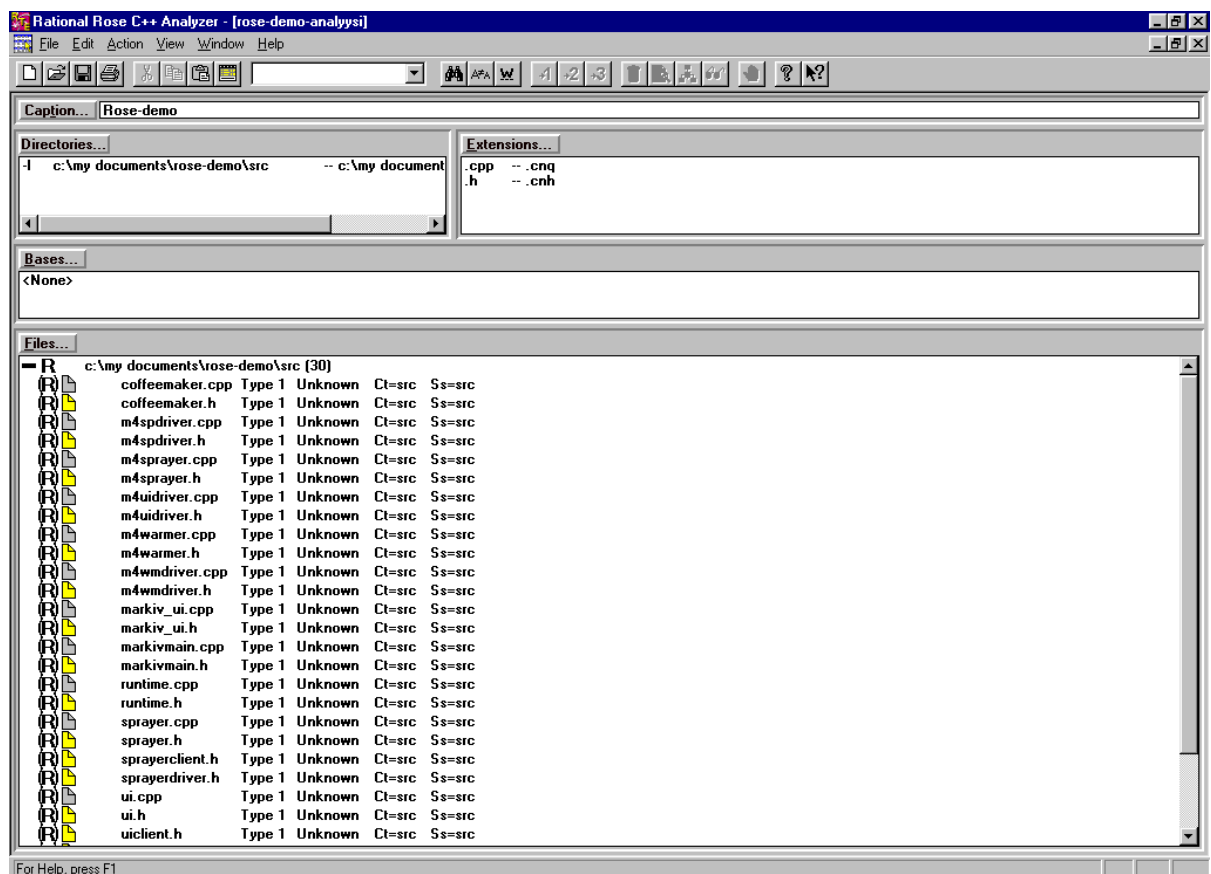


Figure 1: View of the C++ Analyzer

Reverse-engineered model can be layouted with Rose's layout-manager. The layout can be UML, OMT or Booch. The work of the layout manager was found out to be satisfactory - but not good. Usually it was necessary to filter out some dependencies from the model to make it more clear. Rose's support for this dependency-filtering seemed to work ok. When the amount of reverse-engineered classes was big enough (say over 50 classes) the created class diagram was quite tangled. Example of reverse-engineered class diagram is shown on figure 2.

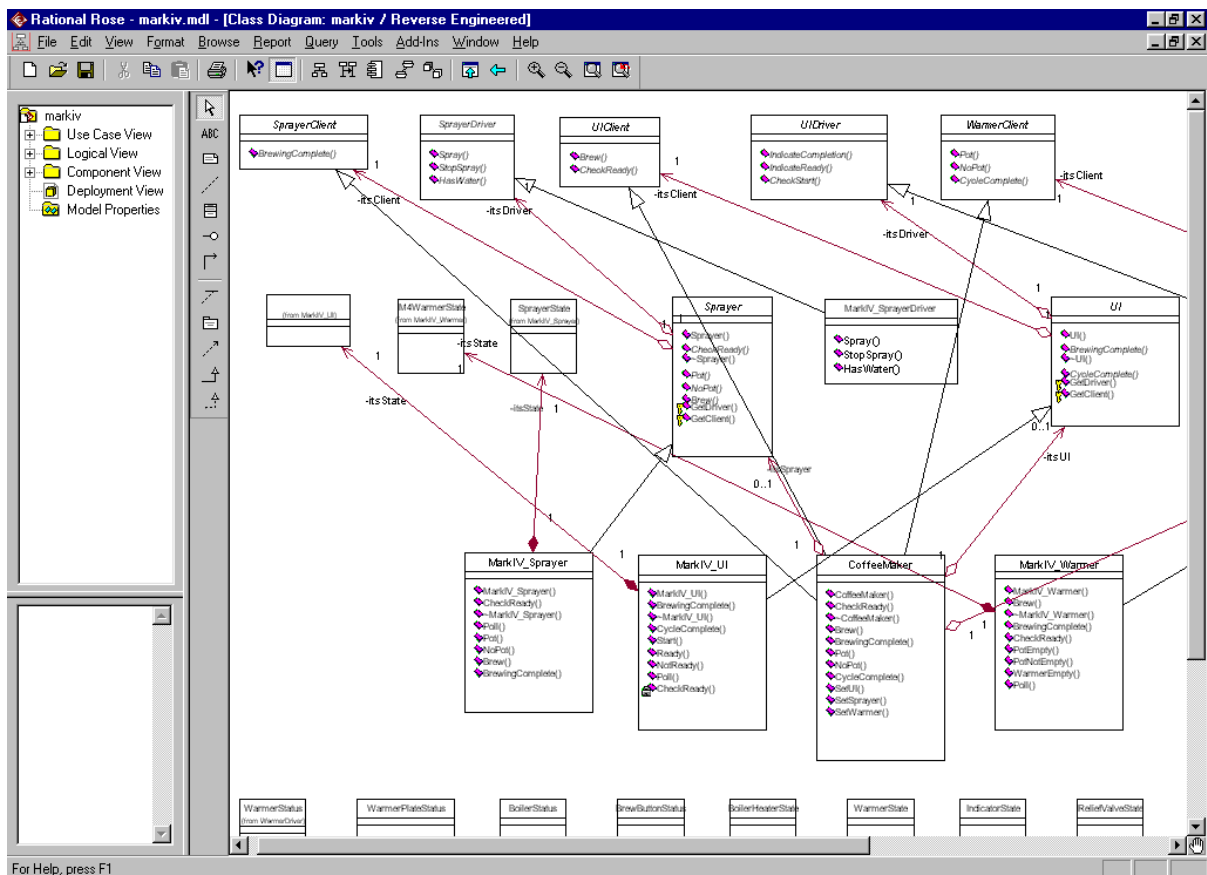


Figure 2. Reverse engineered class diagram.

6. Summary of the Rational Rose

Positive factors

- The tool itself was quite easy to install.
- The creation of the different diagrams can be learned quite fast.
- Code generation is simple.
- C++ Analyzer was also easy to use (though it's functionality could be included in the Rose itself)

Negative factors

- At first the tool seems to be quite complex.
- Some minor bugs were found.
- Separate tool had to be used (and learned) to reverse-engineer files.
- Layout manager could have been a bit more effective.
- Generated code was a bit obfuscated.

Conclusion

The tool is useful. A little bit learning is needed to figure out the basic features of the tool. Round-trip engineering was quite easy and definitely helps to keep model and source codes up to date. For a small project this tool may be a bit too powerful – meaning that the learning may take more time that is relatively beneficial in a small project.