RevEngE

Using an Integrated Reverse Engineering Environment

University of Toronto
McGill University
University of Victoria
IBM Centre for Advanced Studies

Goal

Reverse engineering concerns the analysis of existing software systems to make them moreunderstandable for maintenance and evolution purposes. The importance of reverse engineering has grown tremendously as corporations face mounting maintenance and re-engineeringcosts for large legacy software systems.

The goal of this three-year project, carried out in conjunction with IBM Software Solutions Toronto Laboratory Centre for Advanced Studies (CAS), is to develop and apply an integrated reverse engineering environment.

In particular, the project addresses:

- software analysis technology,
- tool integration technology,
- software engineering repositories, and
- program visualization.

Milestones for the first year focused on building the individual components: reverse engineering tools and the software repository. Year two focused on integrating the tools with the repository. Year three focused on identifying useful analysis scenarios and applying the toolset to understand particular programs. These milestones have been met, resulting in an integrated and extensible environment with a diverse range of program understanding capabilities.

Reference System

CLIPS (C Language Integrated Production System), an expert system shell developed by NASA, was used as a reference program for testing the RevEngE tools and repository. CLIPS consists of about 60 files containing over 700 functions (around 30 KLOC). Although it is small by industrial standards, it is a nontrivial production application that may typically be assigned to a software engineer. Particular analyses of CLIPS are detailed in the demos.

Research contributions

Our reverse engineering environment supports data and control integration through a loosely coupled architecture, a common schema, and a repository developed by the University of Toronto. Pattern-matching and search algorithms are provided by McGill University. These components are complemented by a graphical editor developed by the University of Victoria.

University of Toronto

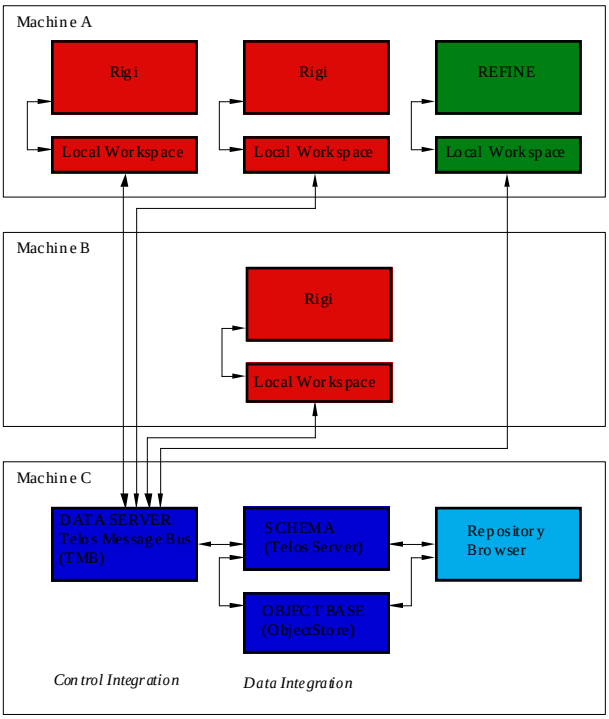
The team at the University of Toronto developed an open, extensible, and modular system architecture. This consists of two components: a repository with a global schema and a data server. The schema, defined with the object-oriented information model *Telos*, supports data integration among the tools. The data server is responsible for transporting information among tools either through the repository or directly. *ObjectStore* is used as the persistent storage manager.

The repository accommodates all data handled by any one of the integrated tools (currently REFINE and Rigi). The common and consistent conceptual schema is a superset of the subschemas required by the individual tools. The use of metaclasses makes it possible to support new tools and new types of information as they appear. Dynamic schema evolution is also supported.

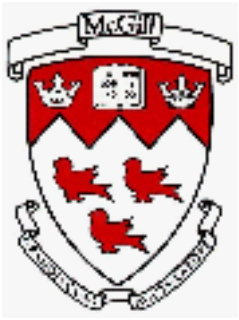
Scalability of the interface to the repository became an issue as larger programs such as CLIPS were loaded into the repository. Although the repository is able to store large amounts of information, the tool workspaces are more limited. In practice, only small portions are actually needed for particular analyses. Additional operations were added to retrieve smaller portions of the repository information and to deal with partial data. Some optimizations were made to save objects in the persistent store more efficiently.

For more information, contact:

Martin Stanley
mts@ai.toronto.edu



McGill University



The McGill team developed a collection of statement-level pattern matching algorithms using the Software Refinery. This effort is known as Ariadne, complementing the high-level structural pattern matching work done by the University of Victoria team.

The current McGill focus is on developing efficient algorithms for statement-level pattern matching. The algorithms form data bindings or common references across procedures, calculate tuples of quality and complexity metric values for the functions, select functions with similar complexity, match code through dynamic programming, analyze accesses to variables, and perform system clustering.

For more information, contact:

Christos Magdalinos
magdal@cs.mcgill.ca

University of Victoria

The team at the University of Victoria enhanced a system for reverse engineering known as Rigi. Rigi provides a parsing system suitable for imperative programming languages, offers a programmable graph editor, and supports conceptual modelling of the application domain.

Through these extensible capabilities, Rigi realizes a meta reverse engineering tool that can be adapted to understand large information spaces, such as source code, documentation, and the Web. Rigi is instantiated for a particular domain by the analyst actively forming a conceptual model, extending the core functionality, and personalizing the user interface.

Since one of the main goals of RevEngE is to ease integration among multiple tools, a high degree of flexibility and scalability in the editor is of paramount importance. Toward this goal, Rigi supports end-user programming whereby an analyst can control many aspects of the editor using scripts written in Tcl. In particular, this interface was used to informally integrate ART (Analysis of Redundancy in Text), a tool developed by J. Howard Johnson at the National Research Council in Canada.

For more information, contact:

Michael J. Whitney
mwhitney@csr.uvic.ca