

Tools for Extracting Software Structure from Compiled Programs

Hwei Sheng Teoh and David B. Wortman
Department of Computer Science
University of Toronto

hsteoh@quickfur.ath.cx , wortman@cs.toronto.edu

This research investigates how much useful information about the structure of software systems can be obtained using existing software tools to analyze compiled programs. This research arose from the observation that the compiled *object units* for a software system must necessarily contain enough structural information so that the object units can be successfully linked into a complete executable program. We developed a suite of tools that extract this structural information directly from the object units.

The tools that we have developed differ from most previous efforts in several ways: **They are source code independent and multi-lingual** because they process compiled programs and thus avoid all programming language specific issues. **They are time and space efficient.** because they avoid the complexities of processing source code our tools require at most minutes to analyze large software systems. **Our tools describe the software as built.** Tools that process the source code describe the program *as written*. By processing the software after it has been compiled our tools describe the software after all compiler transformations and optimizations. This means that our tools can more accurately account for various compiler generated artifacts (e.g. template expansion in C++).

The tools we developed use the `objdump` utility from the GNU `binutils` package to extract all of the available structural information from a collection of object units. Our information tools process this output to produce various structural descriptions of the software. The `dot` program from the AT&T Graphviz package is used to produce graphical output.

We found that there was enough structural information in the compiled object units of software systems to allow us to produce accurate descriptions of the dependencies between the various components (e.g. variables, functions, or object units) in the software. Higher-level components such as modules or subsystems can be abstracted from these basic components. We implemented tools to provide several different graphical views of a software system. Our tools support user-specified *elision* which produces a subgraph of the full graph, which contains only the nodes of interest, and

the edges between them. This allows the user to focus on a particular set of functions, variables or object units. The backend processing tools that we have developed include:

Object Unit Dependency Tool produces a dependency graph for all of the object units in the system.

Symbol Dependency Tool produces a dependency graph of all the symbols (global variables and functions) in the system.

Code Symbol Dependency Tool produces a dependency graph for all of the code symbols in the system. For most purposes this is the function call graph for the system.

Weak/Common Groups Tool produces a graph of global variables defined in the system and the modules that use those variables.

Zoomed In Module Dependency Tools produces a graph that shows the symbol dependencies between a collection of modules. This tool is useful for determining *why* one module depends on another module.

Query Tool This non-graphical tool produces a table of information about the symbols defined in the system.

We tested our tools on the then current Linux kernel (2.2.17), approximately 2 millions lines of source code. It took us about 20 seconds to extract all of the object unit information (3.1 Mb) on a relatively slow 333MHz Pentium processor with 512 Mb memory. A worst case example, running the Query tool to extract all of the information into text form took 9 seconds. An important issue with this type of tool is the accuracy of the information that they provide. We compared our tools with a published set of accuracy criteria for extraction tools and found that our tools were as accurate as source based tools and in some cases more accurate.

The tools that we developed demonstrate that processing object units instead of source code is an efficient and accurate way to obtain structural information about software systems. For a more detailed description of this work see the first author's M. Sc. thesis: Hwei Sheng Teoh, *An Architecture- and Language-Independent Method of Extracting Software Structure*, M.Sc thesis, Department of Computer Science, University of Toronto, 2001