An IL Converter and Program Database for Analysis Tools

Kathleen Lindlan, Janice Cuny, Allen D. Malony, Sameer Shende
Department of Computer and Information Science
University of Oregon, Eugene, OR 97403
{klindlan, cuny, malony, sameer}@cs.uoregon.edu

Peter Beckman
Advanced Computing Laboratory
Los Alamos National Laboratory
Los Alamos, NM 87545
beckman@acl.lanl.gov

1. ABSTRACT

Developers of static and dynamic analysis tools for C++ programs need access to information on functions, classes, templates, and macros in parsed C++ code. Existing tools, such as the EDG display tool, provide that access, but in an unsuitable format. We built a converter that prunes and reorganizes the information into the appropriate format. The converter provides the information needed for our TAU (Tuning and Analysis Utilities) tools and, in more general terms, provides C++ developers considerable opportunities for automating software development.

The EDG Front End parses a C++ source file and creates an intermediate language (IL) tree representing this file. The constructs of this tree correspond closely to the analogous constructs of the C++ language. The EDG display tool walks the IL tree and reports information on each IL entity that it encounters.

Not needing the entire EDG intermediate language tree, we created a converter to report only the information typically needed by analysis tools, such as function call graphs and class hierarchies. Our initial strategy was passive: report on functions and their calls, and classes and their members, as the related structures are discovered during the traversal of the IL tree. We realized, however, that not all related information could be grouped as we wanted, e.g. function calls with the calling function. Our strategy, then, became more aggressive. When processing a function construct, we also traverse that part of the tree -- in another memory region -- containing the IL for its executable code, and report the routine calls encountered there. The class hierarchy information is scattered in structures pertaining to base classes and their derivations: we follow the necessary pointers and compress the information given to just the direct parent classes.

The IL converter processes the intermediate language file for a given source file and creates a human-readable file. This file contains all information on functions and classes, including template instantiations, as well as that for templates and macros. The routine section lists source identification, parent class and access, signature, characteristics, and functions called for each routine. The class section specifies source information, characteristics, direct parent classes, member function IDs, and information on other members. The template and macro sections will report source information, type, and the complete text of each entity.

The resulting program database is useful for a variety of analysis tools. We will use it for TAU, upgrading the class browser, call graph display, and tracing/profiling mechanism. Further, our converter will allow the developers of the ACTS toolkit to create tools that selectively query the database. Such tools can, e.g., automatically generate the HPC++ stubs that are required to interface directly to POOMA objects from perl. A template instantiation browser can be developed that will enable scripting at run-time. These C++ developers now have new options to automate various aspects of the software development process.

The prototype of the TAU IL converter will be available shortly. This version will provide information on functions, classes, templates, and macros. Future versions will offer improved efficiency, greater flexibility with command-line options, and perhaps a functional interface.