Program Transformations for Re-Engineering C++ Components [OOPSLA/GPCE]

Robert L. Akers, Ph.D.
lakers@semdesigns.com

Ira D. Baxter, Ph.D.
idbaxter@semdesigns.com

Michael Mehlich, Ph.D.
mmehlich@semdesigns.com

Semantic Designs Inc.
12636 Research Blvd. #C214
Austin, Texas, USA 78759-2200
512-250-1018

ABSTRACT

Component-based software engineering enables applications to be assembled from component parts that adhere to a component-style specific interface specification and protocol. Components available for one style are not available for another. Component styles evolve, too, which can obsolete components using a legacy style. This creates a demand for migrating components from one style to another, which can require complex changes to the component source code. For a large component library, doing this manually is likely prohibitive. An alternative is to apply automated program transformations to carry out the changes.

Using source-to-source transformations on real code requires a scalable, robust program transformation technology. Such technologies are difficult to justify for single applications. DMS®[1] is a commercial program transformation system which has been used to transform many programming languages, including C++, C#, Java and ObjectPascal. It is parameterized by language and desired task, enabling its infrastructure costs to be amortized across many different software analysis or change applications.

This demonstration shows a concrete example of DMS program transformations being used to migrate legacy C++ components from a Boeing distributed avionics software system, using a Boeing proprietary component format, to a CORBA component style. The conversion requires nontrivial understanding and manipulation of the C++ source code. It will explain the component migration problem to be solved, show some of the transformations, and actually convert a component.

Categories and Subject Descriptors

D.1.2 [**Programming Techniques**]: Automatic Programming – *Automating analysis of algorithms, Program Modification, Program Synthesis, Program Transformations.* D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer-aided software engineering (CASE).* D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement – *Restructuring, reverse engineering, and reengineering.* D.2.13 [**Software Engineering**]: Reusable Software – *domain engineering.* D.3.4 [**Programming Languages**]: Processors – *Parsing, Translator writing systems and compiler generators, Code Generation.*

[1] DMS is a registered trademark of Semantic Designs (SD).

General Terms

Algorithms, Management, Design, Economics, Languages

Keywords

Software transformation, software analysis, migration, component architectures, legacy systems, C++, compilers, re-engineering, abstract syntax trees, patterns, rewrite rules.

1. The DMS Software Re-Engineering Toolkit

The DMS Toolkit provides an infrastructure for software transformation based on deep semantic understanding of programs. Programs are internalized via DMS-generated parsers, available for most languages. Analyses and modifications are performed on abstract syntax tree (AST) representations of programs, and transformed programs are printed with generated prettyprinters. The Toolkit is capable of defining multiple, arbitrary specification and implementation languages (*domains*) and can apply analyses and transformations to source code written in any combination of defined domains. Transformations may be either written as procedural code or expressed as source-to-source rewrite rules in an enriched syntax for the defined domains. Rewrite rules may be optionally qualified by arbitrary semantic conditions. The DMS Toolkit can be considered as extremely generalized compiler technology, offering these facilities:

- A hypergraph foundation for capturing program representations (e.g., ASTs, flow graphs, etc.).
- Complete procedural interfaces for processing ASTs, etc.
- Means for defining language syntax, and deriving parsers and prettyprinters to/from DMS internal ASTs.
- Support for defining and updating arbitrary symbol tables holding name/type/location information.
- An attribute evaluation system for encoding arbitrary analyses over ASTs. One use is constructing symbol tables.
- An AST-to-AST rewriting engine that understands algebraic properties (e.g., associativity and commutativity).
- The ability to specify and apply syntax-specific source-to-source program transformations. Such transforms can operate within a language or across language boundaries.
- A procedural framework for connecting these pieces and adding arbitrary code.

C++ is one of many domains defined for DMS, and the system contains preprocessors, parsers, prettyprinters and symbol table building for both the ANSI and Visual C++ 6.0 dialects. Unlike a compiler preprocessor, the DMS C++ preprocessor preserves both the original form and expanded manifestation of the directives within the AST so that programs can be manipulated, transformed, and printed with their preprocessor directives preserved, even in the presence of preprocessor conditionals.

DMS has been used for a variety of large scale commercial activities, including cross-platform migrations, domain-specific code generation, and construction of a variety of conventional software engineering tools for dead and clone code elimination, test code coverage, source browsing, and static metrics analysis. A more complete discussion of DMS is presented in [1]. DMS-based tools are described on the Semantic Designs web page [2].

2. The Boeing Migration Tool

Boeing's Bold Stroke C++ based embedded avionics component software architecture is based on the best practices of the mid 1990's [3]. Component technology has since matured, and the CORBA component model has emerged as a standard. The U.S. Government's DARPA-PCES program and OMG are sponsoring development of a CORBA-inspired standard real time (CCMRT) embedded system component model [4], which offers standardization, superior encapsulation and interfaces for ongoing development of distributed, real time, embedded systems, as well as a base for tools for design and analysis of such systems. Boeing wishes to modernized its software to use PriSm, a proprietary CCMRT variant. The task of converting components is technically straightforward and now well understood, but a great deal of detail must be managed with rigorous regularity and completeness. Since Bold Stroke is implemented in C++, the complexity of the language and its preprocessor requires careful attention to semantic detail. With thousands of legacy components now fielded, the sheer size of the migration task is an extraordinary barrier to success. With the use of C++ libraries, approximately 150,000 lines of C++ source contribute to a typical component, and a sound understanding of the component's name space requires comprehension of all this code.

To deal with scale, semantic sensitivity, and regularity issues, DARPA, Boeing, and Semantic Designs (SD) decided on an automated approach using a DMS-based tool, dubbed "BMT" for "Boeing Migration Tool". DMS and its C++ front end was uniquely qualified as a migration tool. Automating the migration assures regularity of the transformation across all components.

The legacy component structure was essentially flat, with all the methods contributing to a component collected in a very few classes. One principal piece of the migration involves factoring a component into facets, which would form distinct classes reflecting different areas of concern. Some facets encapsulate various functional aspects; others capture protocols for inter-component communication.

Factoring a component into functional facets requires human understanding. Essentially, the legacy interface methods must be sorted into bins corresponding to the facets, and indicative names given to the new facet classes. To provide a clean specification facility for the Boeing engineers using the BMT, we developed a simple facet specification language. For each component, an engineer simply names the facets and uniquely identifies which methods (via simple name, qualified name, or signature if necessary) comprise its interface.

The BMT translates components one at a time. Input consists of the source code, the facet specification for the component being translated, and the facet specifications of all components with which it communicates, plus a few bookkeeping directives. Conversion-related input is succinct.

The facet language is defined as a DMS domain, enabling easy parsing by DMS. A DMS-based attribute evaluator over the facet domain traverses the facet specifications' ASTs and assembles a database of facts for use during component transformation.

After processing facet specifications, the BMT parses and does full name and type resolution on the C++ source code, including files included by any of the components in play, building complete symbol tables for all files involved in the component. An accurate table enables accurate name lookup, which is the key point that defeats scripting languages as C++ transformers. While a considerable number of transformations are required to achieve the component conversion, two particular transformations typify what the BMT does to perform the component migration:

- New classes for facets and their interfaces are generated based on the facet specifications. C++ code corresponding to the designated methods are found, moved to each facet, and all access paths are adjusted to account for movement of the method, other called methods, and component data.
- Newly generated "receptacle" classes provide an image of the outgoing interface of a component to the other components whose methods it calls. Constructing the receptacles involves searching all of a component's classes for outgoing calls and generating code to serve each connection accordingly.

The transformations are coded as a combination of DMS source-to-source transforms, attribute evaluation, and procedural code.

The demo will describe DMS in sufficient detail to understand how the components are transformed, and show a legacy component being converted.

3. ACKNOWLEDGMENTS

We give our thanks to our collaborator in this effort, the Boeing Company, and to the DARPA PCES program for its funding.

4. REFERENCES.

[1] Baxter, I. D., Pidgeon, C., and Mehlich, M., DMS: Program Transformations for Practical Scalable Software Evolution. *Proceedings of the 26th International Conference on Software Engineering,* 2004.

[2] Semantic Designs, Inc. web site, www.semanticdesigns.com.

[3] Sharp, D. C., Reducing Avionics Software Cost Through Component Based Product Line Development, *Proceedings of the 1998 Software Technology.*

[4] Gidding, V., Beckwith, B., Real-time CORBA Tutorial, OMG's Workshop on Distributed Object Computing For Real-Time and Embedded Systems, *www.omg.org/news/meetings/workshops/rt_embedded2003.html*, 2003.