A Framework for Component-based Compiler Development

Xiaoqing Wu
Advisors: Barrett R. Bryant, Jeff Gray, Marjan Mernik
Department of Computer and Information Sciences
The University of Alabama at Birmingham
http://www.cis.uab.edu/wuxi/research/ Email: wuxi@cis.uab.edu

1. Problem Statement

Current compiler development practices suffer from lack of modularity.

- No decomposition of language definitions
- Large grammars may run into thousands of productions (e.g., Cobol 85 has 2500 lines of specification with more than 1000 variables).
- There is no modular grammar support in most parser generators (e.g. YACC).
- No clear separation of compiler construction phases.
- Syntax and semantics: The communication between syntax (in formal specification) and semantics (in programming languages) makes the specification and code tangled together.
- Among different semantic phases: In pure object-oriented design, code scatters throughout the syntax tree class hierarchy.
- Lack of modularity usually yields poor comprehensibility, reduced reusability, and hampers independent development of the language implementation.

2. Background

Component-Based Software Engineering (CBSE)

The goal of CBSE is to build a software system by assembling small components in the same manner as hardware composition. A software component is generally considered to have the following properties:

- Information hiding
- Explicit interface
- Context independency

Aspect-Oriented Programming (AOP)

- Aspect-Oriented Programming provides special language constructs called aspects to modularize crosscutting concerns in conventional program structures. AOP offers two new types of formalisms:
  - introduction to existing structure
  - insertion of group behaviors (i.e., join point models).
- In AOP, a translator called a weaver is responsible for merging the aspects into the base language at compile time.

3. Research Objectives and Major Contributions

- Objective I: Large languages should be developed based on small language implementations.
- Objective II: Inside a single language implementation, each compiler phase should be separated with each other.

4. Component-based development framework

Grammar Components:

- A grammar component G is a quintuple (N, T, C, P, S)
  - N: nonterminal symbols
  - T: terminal symbols
  - C: other grammar components
  - P: LR productions
  - S: start symbol
- Function decomposition:
  - Structuring a large language into a set of smaller languages, each of which is first implemented separately and then assembled together.
  - Using grammar components (pluggable components) that can be merged at compile time via Aspect-Oriented Semantics.
- Architecture:
  - Each grammar component generates a parser component and an aspect-oriented syntax tree.
  - To conduct parsing, the root parser invokes its sub-parse trees recursively.

5. Case study: the Java language implementation

Java language components:

- Implementations of java.util classes are expressed as aspects.

6. Framework Overview

- Structure decomposition: CLR decomposes a large language into a set of smaller languages, each of which is first implemented separately and then assembled together.
- Function decomposition: Object-Oriented Syntax (OOS) and Aspect-Oriented Semantics (AOS) are used jointly within each language implementation to facilitate Separation of Concerns (SoC), where syntax and semantics as well as semantic phases themselves are isolated into different modules.
- Information hiding:
  - Aspect-oriented semantics employed in the framework supersedes the object-oriented Visitor pattern by unrestricted method definitions and maintenance nature of node classes, as well as the flexibility in tree walking and phase composition using join points.