Using Object Oriented Techniques at Implementing Compilers

Zoltán Porkoláb

Object Oriented Programming became well-accepted technology of Computer Science in the recent years. Object Oriented Methods of analysis and design are to standardize and supported by CASE tools. Programming languages - such C++, ADA95 and JAVA - are in everyday use helping the programmer to implement object oriented techniques with classes, using inheritance, exception handling, etc. In a few fields of Computer Science, however, Object Orientation is still not a mayor player. In specially, in the world of compilers, translators, etc. we can see mostly old-fashioned, not object oriented programs. This is perhaps originated in the event-driven attitude of using finite automaton.

In this article I try to explain another way of writing translators - the object oriented way. In this case the object structure of the program is driven from the internal data structures of translation. Lexical tokens are detected by lexer, and passed to the parser as an object. The parser is building a syntax tree using Predictive Descending Algorithm (Aho, Sethi, Ullman: Compilers. Addison-Wesley, 1986), recursion was eliminating with while loops. However this syntax tree is containing objects from a certain object hierarchy. Each node in the tree is an instance of a class corresponding to a certain operator-group. This way of typing the nodes of syntax tree makes late binding in effect at type checking and code generation.

Type checking makes function overloading possible. Rules of function argument matching are separated in distinct algorithms. A few case of polymorphic function needs special handling. Code generation also use virtual functions producing ready to compile JAVA source. Exception handling is used not only to break syntax analysis at error but also to generate warnings. Error and warning information is encapsulated in classes and can be caught.

This translator is a part of a real-word project, implemented in JAVA 1.1. In the project 4GL (SuperNOVA) source code is translated to JAVA (applet) source. The translator is validating the 4GL expressions and generate JAVA code. Since this 4GL is based on interpreter and allows run-time evaluation of expressions, the class structure of the translator is also a part of the applet. This way the translator is used in compiling time to check the most of the source and generate static JAVA code, but also activated in run-time if a certain expression is using run-time evaluation. The project has finished in the last year and now the translator is ready to use.