Specification and Implementation of a Tree-Abiding Interface for Ada
(Extended Abstract)

Martin Hartwig, Eckhard Stein, Roland Strobel
Academy of Sciences of the GDR
Central Institute of Cybernetics and Information Processes (ZKI)
Kurstraße 33, Berlin, 1086, GDR

1. Introduction

Ada is a language of high complexity concerning both sytax and semantics. To manage this complexity with respect to the communication between compiler phases and environment tools there is a strong need for formal specification and automated handling of interfaces.

A number of available Ada compilers rely on DIANA, the Descriptive Intermediate Attributed Notation for Ada /1/. This interface is expressed using the Interface Description Language IDL, a formal means to describe abstract data structures as typed, attributed, directed graphs /2/.

In the ZKI Ada project we decided to deviate from the DIANA interface and tried to apply highly efficient traditional compiler techniques in the framework of IDL, e.g. symbol table stacks and reverse polish notation.

The evolving TRee ABiding Interface (TRABI) provides a solution to the following implementation problems:

- reduction in storage requirements (solution: syntactically directed I/O of subtrees as opposed to virtual node interface in conventional implementations),
- simplification of I/O-routines (solution: the general directed graph model of the DIANA design is replaced by a strictly tree-oriented data structure),
- efficient attribute access upon semantic processing (solution: the functional interface used in conventional implementations is on a local level replaced by Ada's basic operations such as indexing, selection, and slicing).

The proposed techniques are of general use when implementing languages with a high degree of complexity such as, e.g., VHDL or CHILL.

2. Design Principles of TRABI

Whereas DIANA represents the source code information as a directed graph, we use a strictly tree-abiding data structure.:

- only the abstract syntax tree (AST) is presented as an IDL-graph,
- attributes reflecting semantic links (e.g. references from applied to defining occurencies) are handled by a private type.

Values of this private type are implemented as indices to a *semantic vector* that in turn contains references to declarative nodes in the AST. The separate handling of semantic references outside the structure specified by IDL offers the following chances:

- It is easier to handle a tree than a general directed graph. This concerns traversals (no need to check for cycles) as well as partial I/O of the interface.
- The semantic vector simplifies the inspection of certain sets of declarations.

With TRABI, we avoid virtual node management in favour of syntactically directed tree-I/O. Candidates for partial I/O are statements, local declarative regions, and contexts needed solely upon code generation for generic instantiations and inline expansion of subprograms. We no longer employ any attributes for accessing these trees but rather explicitly trigger an input operation as soon as the corresponding structures are needed by the processing algorithm.

In order to indicate the corresponding side effects of these input operations on the semantic vector, we enriched the I/O facilities of IDL by features to

- specify input and output conversion functions for private types,
- specifiy subprograms to be called before output or after input of nodes or attributes, respectively.

The evolving specification is supported by an IDL tool case that generates Ada packages defining and operating the interface. As a result, we get rid of a functional interface in favour of Ada's basic operations such as indexing or component selection. Syntactically directed I/O compensates for the missing virtual node layer, class and attribute transformations solve the problem of mapping internal/external references.

3. Practical Results

The TRABI specifications of the ZKI Ada compiler frontend is in size comparable to DIANA. From a total of 80_000 Ada source lines, 20_000 are automatically generated. The generated packages provide for:

- declarations to describe the internal representation of the interface structure,
- allocation/deallocation of nodes,
- binary input/output of trees,
- ASCII input/output of trees (testing only), printing a (sub)tree (documentation and debugging only).

The IDL tool case consists of about 9_000 source lines. Implementation language is Ada-0, an Ada-subset including dynamic types and separate compilation /4/.

Much of the TRABI design effort had to be spent on synchronizing the compilation phases (parsing, analyzing, and code generation) with respect to syntactically directed I/O.

Due to the automated IDL support changes to the interface posed no problems. The use of Ada's basic operations payed off by a high speed of compilation (3000 source lines per CPU minute on a VAX 8600). This result is the more remarkable as it has been achieved prior to any tuning and without suppressing Ada's time-consuming constraint checks.

References

/1/ G. Goos et al: DIANA. An Intermediate Language for Ada. Lecture Notes in Computer Science no. 155. Springer 1983

/2/ J.Nestor et al: IDL-Interface Description Language Formal Description. Draft Revision 2.0, Computer Science Department Carnegie-Mellon University 1982

/3/ T. Smith et al: A Standard Interface to Programming Environment Information. In: Ada in Industry. Proceedings of the Ada-Europe International Conference Munich 1988. Cambridge University Press 1988.

/4/ R. Strobel, M. Hartwig, E. Stein: Das Ada-0-System des ZKI. Nutzerhandbuch. ZKI Informationen. Sonderheft 2/1987