THE APEX DESIGN SYSTEM

Robert Marmelstein, 1Lt, USAF

Air Force Wright Aeronautical Laboratories (AFWAL/AAAF)
Wright-Patterson Air Force Base, OH

This paper describes the APEX code generation tool for the Ada language. APEX was developed at the USAF Avionics Laboratory as a result of productivity studies performed on Computer Aided Software Engineering tools for Ada. The lessons learned from these studies helped define the requirements for APEX. APEX provides the programmer with three different, yet consistent, graphical views of the Ada program under development. Once the graphical specification has been developed, Ada code can be generated from APEX. In addition, APEX can reverse engineer the graphical specification from existing Ada code in order to support software reuse.

I. Background

In recent years there has been a surge in the number of Computer Aided Software Engineering (CASE) tools used in the development of USAF software. This surge has primarily been in response to the increasing amount and complexity of DOD software and its corresponding documentation. The use of CASE tools is touted as a likely solution to help managers and engineers deal with the ever increasing software load. This paper will be a discussion of research done in-house at the Air Force Wright Aeronautical Laboratories (AFWAL) Avionics Laboratory on graphics based code generation tools for the Ada language.

II. Why Graphical Programming?

The use of graphical programming tools offers several potential advantages over traditional forms of software development. Perhaps the biggest advantage these tools offer is the ability to generate both Program, Design Language (PDL) and documentation from a central graphical representation. When changes are made to the central database, automated updates are made to the PDL and documentation. The automated generation of documentation can take advantage of the inherent overlap between similar sections of

different documents. A "smart" database will also have the ability to propagate changes made to one part of the program to different, yet related parts of the program. This feature will keep the overall design consistent. These features are geared toward decreasing the cost of making a change to a design.

Another area where graphical programming is useful is in the rapid prototyping of a design. The PDL generated from a detailed design can often be modified, compiled, and executed in much less time than starting to code from scratch. By executing this code, an early prototype of a design can be tested and analyzed. In this manner, rapid prototyping can uncover flaws in the design which can be isolated and fixed at an early state of development.

Graphical programming can also aid in the reuse of software. Some CASE tools are starting to provide the capability to reverse engineer existing software (developed without their system) into the graphical design notation supported by their product. This capability can help the programmer more easily reuse existing software, since it is often easier to interpret a design than to understand the source code that implements the design. The reuse of the program design affords more flexibility to the programmer than does the reuse of the source code itself. In addition, the programmer won't have the burden of rewriting the existing software with the CASE tool being used. The combination of these factors illustrates how graphical programming tools can potentially improve software development.

III. Prerequisites for Success

The advantages outlined in the previous section summarize the potential of this technology. Unfortunately, the current state of the art has not lived up to this promise. Over the past 3 years, AFWAL has gained a wealth of experience with these tools based

on several in-house evaluations of graphical programming tools for Ada. The most notable of these tools was the Interactive Ada Workstation (IAW) developed by General Electric under AFWAL sponsorship. As a result of our in-house evaluations, we have identified some general weakness inherent in these products:

- *Design methodologies are too complex.*
- *Limited support for essential language constructs.*
- *Limited code generation for subprogram bodies.*
- *Poor quality of generated code.*
- *Different views of program are not consistent.*
- *Overhead required to maintain the graphical design is large.*

Since those who do not heed past mistakes are condemned to repeat them, the requirements for APEX were driven by the shortcomings of the tool's evaluated. The APEX requirements are outlined below.

1. The design methodology employed must be simple to learn. As such, it must not burden the user with overly complex ways of defining data structures, subprograms, or relationships between program components. Redundant operations should not be required to add or delete the same information from the program model.

2. The tool should provide a user friendly interface and perform operations within a reasonable amount of time.

3. The tool must produce both PDL and documentation from its representation. The PDL produced from the graphical representation must be of the quality one would expect from an experienced programmer.

4. The design methodology must support several views of the program. High level languages have three fundamental views: the architecture view (describing the relationship between program units), a data structure view (describing the representation of data) and the control flow view (displaying program logic). Because these views of the program often contain overlapping information, they must be consistent with each other. Information presented in one view must not contradict information presented in another view. In order to keep all views consistent, the tool should have a unified database for its program representation. In addition, each view should have separate rules for interpreting the database. Without multiple views of the program, it is difficult to produce anything more than a one dimensional program. For example, some Ada design tools produce package declarations, but provide no

means for generating the code for the subprogram body. The utility of such a design aid is extremely limited.

5. The design methodology must be optimized for the target language. Many code generation tools tout the fact that they support more than one language. In such cases, the design methodology is so generic that it does not allow the programmer to take advantage of specific language features in his design. For example, if there is no way of expressing inter-process communication or exception handling in a design tool for Ada, then the methodology has left the programmer without the ability to use key features of Ada to his advantage during design. *The fact that Ada has implicit language constructs to perform functions traditionally reserved for operating system calls demands that any design methodology allow the programmer to express these constructs during design. Indeed, this aspect of Ada has blurred the once clear distinction between the design and coding phases of software development.*

6. The tool should support the reuse of existing software through reverse engineering of code to the graphical representation.

IV. The APEX Approach

As a result of the limitations of existing graphical programming tools, an in-house development project was initiated. The objective of this effort was to produce a new Ada design methodology which did not suffer from the identified weaknesses. In addition, a prototype graphical design tool was constructed to illustrate the effectiveness of this new design methodology. This tool is known as APEX. The APEX system consists of a top level editor (the APEX editor) and two peripheral editors (Data Structure, and REM-Net editors). These editors, when used together, provide the programmer with three different, yet consistent, views of his program. Each view is optimized for its specific purpose. Once an application has been graphically designed using APEX, the system will generate both the Ada code (PDL) and the documentation which describes the program. The following sections provide a brief description of the three APEX editors. An illustration of each editor and the resulting code generated is provided in the final pages of this paper.

1. APEX Editor

The purpose of the APEX editor is to provide a mechanism to clearly represent package dependencies and contents. The APEX editor is used to represent the Ada package declarations only. The editor allows the

subprograms defined within that package appear in separate, scrollable windows (see Figure 1). In addition, APEX is used to define the appropriate package dependencies. The APEX editor establishes the program structure; the other editors are used to elaborate on that structure.

2. Data Structure Editor

This editor gives the programmer the capability to edit his data structures (created using the APEX editor). When data structures are initially created (in the APEX editor), their description is sometimes incomplete or may need modification later; the Data Structure editor provides a mechanism for this. For example, the definition of a record type can be modified in the Data Structure editor through the addition or deletion of fields in that record. In addition, the Data Structure editor can be used to declare variables within subprograms visible to the chosen package by making instances of the available data structures. The Data Structure editor is illustrated in Figure 2.

3. REM-Net Editor

The purpose of the REM-Net editor is to provide a means of designing the subprogram body to support the generation of structured Ada code. One drawback to many current design methodologies is that they fail to support the specification of many Ada language constructs. The REM-Net editor provides support for the following Ada constructs:

```
-- If/Then statements
-- Accept/Select statements
-- Delays
-- Entries
-- Exception Handlers
-- Guards
-- Loops (While, For, and
  unconditional)
-- Code Blocks, Procedures,
  Functions, and Tasks
```

The editor has three main icons; these are states, events, or slots. A state can be classified as a code block, subprogram call, rendezvous call, or an exception handler. An event represents either a select or accept statement (a delay is optional). Slots represent entries into a task. Once an slot has been defined in a task, other subprograms (that the task is visible to) can make a rendezvous call by linking to the existing slot.

Connections between icons have different meaning depending on the context. A

connection between a slot and an event implies a component of a select statement (or an accept statement if only one slot is attached). Any connection between a state and an event represents an unconditional flow of control. A connection between a state and a slot represents a rendezvous call. The system is sufficiently "smart" that illegal situations will not be allowed to complete (for example, a transition between a slot and a state not classified as a rendezvous call will be rejected). Transitions between states can be defined as loops, if statements, or as an unconditional flow of control. An example of a REM-Net representation can be found in Figure 3.

V. Reverse Engineering

APEX has some limited support for reverse engineering of Ada code to graphics. A file containing the specification for an Ada package can be loaded into the APEX editor and converted into its semantic equivalent in graphical form. Currently APEX only supports this capability for the package specification, not for the package body. Despite this shortcoming, this capability is still extremely useful for reusing existing software. When building a new application, the user can load in one or more existing packages in this manner. Once these packages are loaded, the user can use their existing data structures and take advantage of the operations they have which are already written. Because the code already exists, it is not necessary to completely reverse engineer the entire package, rather only the specification the package provides to interface to other program units.

VI. Document Generation

APEX also supports the generation of documentation. The Data Structure editor has a comment window which allows the user to associate comments which program structures (packages, subprograms, data structures, record fields, and variables). These comments are used in the generation of a data dictionary which describes the function of program components. In addition, a document is generated which lists the place in the code where each subprogram is referenced from.

VII. Current Status

The current APEX system is written in C and is hosted on a Sun 3/110 workstation running Sun OS version 4.0. An earlier version of APEX was prototyped on a Symbolics 3600 Lisp Machine.

Figure 1.

Figure 2.

Figure 3.

Figure 4.