Architecture of Languages for Large CAD Systems Development

Clive K. Liu
GE-Calma Company
Research and Development
527 Lakeside Dr.
Sunnyvale CA. 94086

Abstract

This paper reviews current approaches to designing and implementing the User Programming Language (UPL) for computer-aided design. The requirements of the UPL, Application Programming Language (APL) and System Programming Language (SPL) are summarized. Some alternatives to language architectures for users at different levels are analyzed, and a model is proposed for the next generation of CAD systems. The implementation issues of this architecture are also addressed.

Introduction

A CAD system generally includes different modules for at least three levels of users, such as, system operators, application programmers and system programmers. Computer languages and development environments are required for these users to perform various tasks. The UPL for system operators (technicians, designers, and engineers) is used to operate the system and customize the CAD software for the needs of particular installations. More frequently, these users will also tailor the system to meet their practices and styles with this language. I will define this language that serves as the outermost level of dialogue between users and the CAD system to be the UPL. Similarly, APL is used by application programmers to customize CAD systems for different applications, e.g., the application of a piping layout. SPL is used by system programmers to implement the system on some hardware and operating systems. With these definitions, the UPL is used by technicians to record the design layout, by designers and engineers as casual programmers to define the design artifact and its semantics, to verify the design and to customize the user interface. Because UPL is used by a variety of users with different purposes and knowledge of programming, it is difficult to come up with a good solution that meets all the requirements. However, it is important to define such a single and consistent language since a user usually performs more than one type of task (e.g., record the design layout and write a simple program for his own needs) during the design evolution. It is in the users best interest to have a simple yet powerful (in terms of expressiveness) CAD language at this outermost level.

This paper reviews current approaches to designing and implementing the UPL for computer-aided design. Different architectures of the language for users at different levels are analyzed, and a model is proposed for the next generation CAD systems. The implementation issues of this architecture are also addressed.

Current Approaches and Related Work

Most CAD systems provide a UPL so users can customize the system. A UPL is provided because of a very strong demand from the marketplace. For example, Graphics Programming Language (GPL) and Design Analysis Language (DAL) [2] [3] are two user-oriented languages for Calma's GDSII and DDM systems. Grip is the graphics programming for McAuto's Unigraphics system. Formtek Drawing Language (FDL) is the language for Form:Draw users [7]. These languages are all used as an integral part of the underlying CAD systems, i.e. GPL for GDSII, DAL for DDM, Grip for Unigraphics and FDL for Form:Draw. Given the language, users are able to access most of the procedures and functions provided. These languages typically consist of the

following features:

- Interactive commands
- Vocabularies including identifiers, numbers, strings, etc.
- Manipulation of primitive data types such as coordinate, line, circle, array, etc.
- Declarations of variables and macros
- Arithmetic, logical, relational operations and other expressions
- Control statements such as branching, looping and conditional execution
- Functions to group items into some logical structures such as layers, set, etc.
- Functions to deal with user interface such as system messages and menus
- Graphical manipulation including translation, rotation, etc.
- Display functions for manipulating windows, views, panning and zooming
- Geometric constructions that are available in the underlying interactive system
- Annotation functions including texts (different fonts), dimensioning, etc.
- Operating system utilities such as file I/O

Some vendors offer rather simple command languages that include only a portion of the above features. Others offer a general purpose programming language with utilities that are provided by the underlying CAD systems. These languages allow users with little programming knowledge to customize the system and develop some simple packages of their needs. Vendors who do not offer this sort of language are most likely trying to come up with one in a short period of time.

In the research environment, many researchers also recognize the need to provide a simple yet powerful language for CAD users. Glide2 [5] is a CAD language (a superset of Pascal) that provides facilities such as database support and geometric modeling needed for producing large integrated CAD systems. Glide was developed at Carnegie-Mellon University to be used for CAD applications development. CAEADS (Computer-Aided Engineering and Architectural Design System) [6] was a prototype system actually implemented in Glide2. Ideograph [9], another language developed at C-MU, was designed as a drawing language that allows the semantics of drawings to be defined and maintained. Ideograph contains most features offered by a programming language. It also includes many functions that are required by a drafting/drawing system. End users can easily operate and extend the drafting system using this language. End users may also define a new form through the inheritance mechanism for a class of drawings. The semantics of the form are maintained and enforced by the operations defined in the form. The language allows programmers to encapsulate the data types. End users should only concern themselves with how to manipulate the drawing objects; they should not have to know how the data types are implemented.

Another interesting language for CAD applications is an object-oriented language TM [8] that includes full database capabilities required in a CAD environment. The language offers various characteristics, e.g., object-oriented approach, encapsulation, extensibility, attribute inheritance, public and private definitions, communication among objects through message passing and response adding.

Language offered by CAD vendors are intended to be the UPL for end users. Most of these languages only offer function abstractions. Function abstractions are usually specified by input-output relations that allow users to apply most utilities provided by the underlying systems without knowing how those utilities are actually

implemented. With only function abstractions, users will have to deal with complicated data structures and make sure of the correctness of the data to be operated by the function. This requires pretty extensive knowledge and programming experience of end users. Therefore, most of these languages restrict users from defining new data structures and they do not support the type notion to avoid language complexity. However, this approach certainly limits the capability of extending semantics of the data that end users would like to describe. The languags developed in the research environment offer data abstractions in one form or another. Glide2 has frames to encapsulate the data structures and functions. Ideography has forms that allow users to use the abstraction without concerning what are inside the form. Users are also allow to create a new form (abstraction) by simply inheriting other data structures and functions from previous defined forms. TM has classes, objects and inheritance for functions and attributes. Objects in TM are communicated through messages. These languages provide a good encapsulation mechanism for end users to manipulate and extend the complicated objects offered by the CAD system. However, these languages may not be designed only to address the needs of end usres. Several issues of how the language can be integrated well with other language environments for other types of users such as application and system programmers have remained untouched.

Architecture of Languages for CAD Systems

Four basic architectures of CAD systems languages are as follows :

* One single language used for SPL, APL and UPL (Fig.1(a)):

  The advantage of this approach is the continuity of interfaces among different layers from hardware to the end user. However, one language may be suitable for one development layer but not for others. Fortran is the most commonly used language for this architecture.

* One language is used as SPL and APL; a different language usually implemented in the SPL/APL is used as UPL (Fig.1(b)):

  This architecture has been undertaken by the most current CAD systems [10]. Examples are Applicon's command language that is the UPL for AGS/880 system; DAL is the UPL and Fortran is the APL and SPL for Calma's DDM system. The UPL of this architecture is usually a command language such that each command gets interpreted and appropriate subroutines are called. Users may combine a set of commands and later refer to them as a whole. The advantage is that this command language can be designed to be easily used by end users. However, this language often does not contain some important features offered by general programming languages such as type extension. Another approach of this architecture has been to extend an existing programming language to include some keywords for graphics. This language inherits all features provided in the host language such as Fortran and Pascal, however the language is too complicated and addresses some features beyond the scope that CAD operators and casual programmers can comfortably handle.

* APL and UPL are defined to the same language that is implemented in SPL (Fig.1(c)):

  There is another approach to defining an application-oriented language that will meet the specific needs required by the applications. Examples are languages for general drafting applications, and languages for VLSI layouts. This approach can produce a language that is good for the application in mind. Users

(application programmers and end users) will have a better chance of writing code expressed in their terms. The FDL in Formtek's Form:Draw system [7] is designed specifically for the users doing drafting and building floor plan layout. The language is implemented in an extended version of Pascal which is the SPL for the Form:Draw system. Since a large portion of the system will be coded in APL (and UPL in this case), the performance, maintenance and portability issues will require very special attention.

* Three different languages for SPL, APL and UPL (Fig.1(d)):

Some systems took this architecture to provide different languages for different layers of system use. For example, in Calma's GDS II sytem, the assembly language of Data General's hardware (Eclipse machine) is used as the primary SPL. XGL (a language similar to Algol) is used as the APL and GPL is used as the UPL that is an intepreted language. The advantage of this architecture is that each language can be chosen or designed so it is more suitable for one particular layer than another. It is important, however, to provide good interfaces between these layers in order to come up with an integrated environment so that users of different layers can move easily from one layer to another and all the

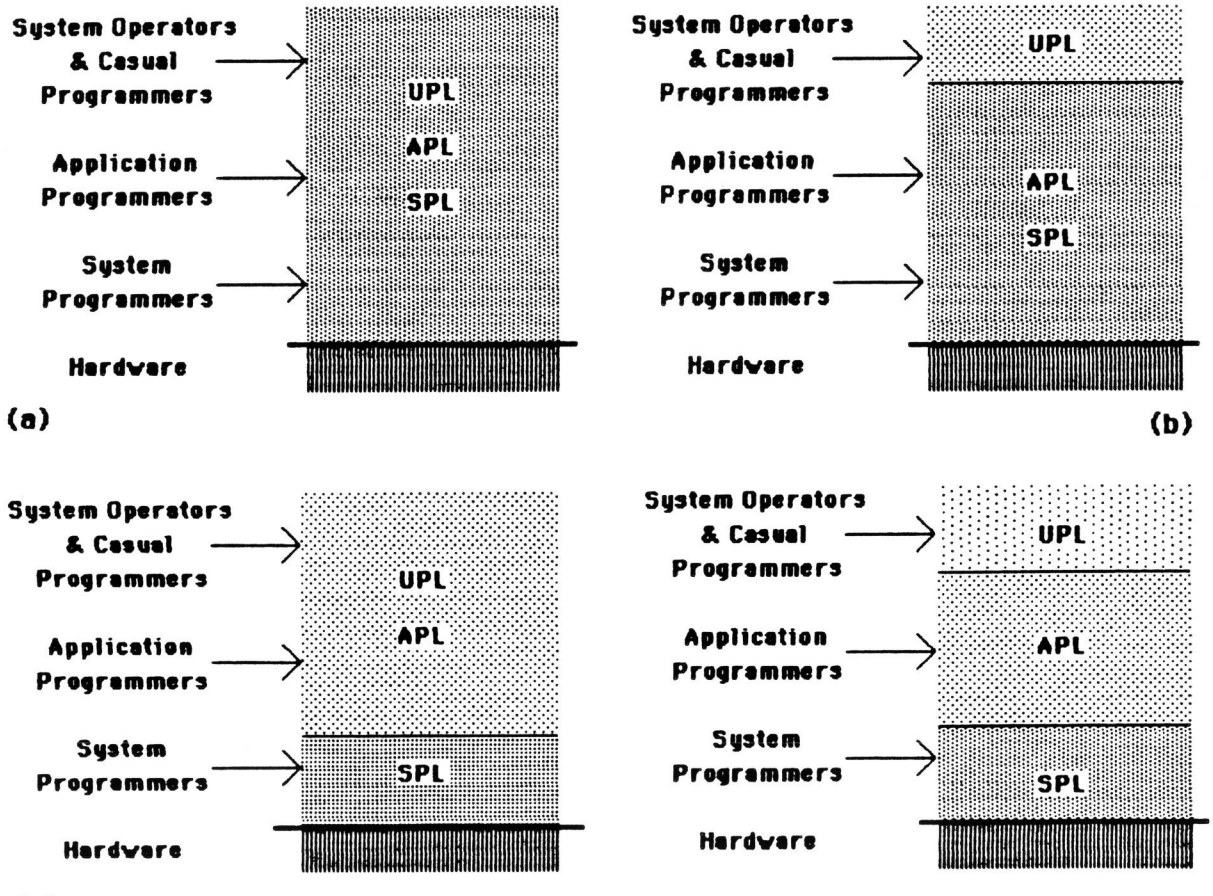

Fig.(1) Language Architectures

utilities provided beneath the layer can be accessed naturally.

Although these four different architectures, consist of different languages for different requirements, they currently provide the only capability of function abstraction. Almost none of the existing commercial systems, to the author's knowledge, provides data abstraction in UPL. Without the capability of data abstraction, the complexity of the UPL (e.g., type extension) increases so that the language is a useful one. Unfortunately, when the language is complicated, it no longer remains as a good UPL.

A Language Architecture for Large CAD Systems Development

There are hundreds of programming languages available, and still many languages are under design and development. When a new project starts, it always calls for a sound programmatic decision relative to the choice of a language for the system development. It is also required to provide a language as the interface (whether a graphical, dialogue-based or programming-based interface) to the system. Typically, the decision is simply determined by the hardware used for running the system and the language used in existing code. However, a CAD system development is usually a large effort and it involves a very long lifetime. The decision is thus not so straight forward. It requires the considerations of language simplicity, portability, reusability, maintainability, efficiency and machine-level accessibility. These factors are all important and somewhat conflicting. Therefore, it is unlikely to decide or define a single language that meets all the requirements. A practical architecture is probably the "mix and match" model that suggests three primary programming languages for UPL, APL and SPL separately (see Fig.2). The use of a language may be extended to the level beneath or above it, depending on individual cases. For example, a large portion of system programming may be performed by APL programs, if the APL has good performance and easy access to the underlying hardware. Within a layer, there may be a few other languages used besides the primary one. The most important consideration, however, is that data abstractions should be offered so the "mix and match" languages can be integrated.

The purpose of the UPL is to aid in producing prototypes that can be refined into production quality systems. Therefore the language should make program writing easy (shorter programs do equivalent tasks, greater reusability of existing code etc.); turn-around time for debugging and trying new ideas should be faster (interactive language, source level debuger with graphics aid and being

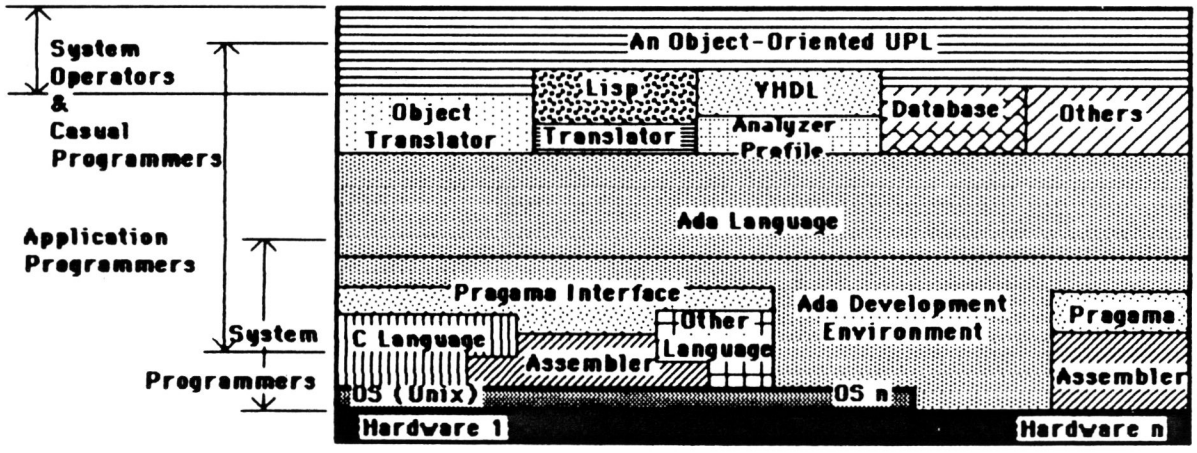

Fig.(2) A Language Architecture for Large Electronics CAD/CAE Systems Development

able to resume after fixes, dynamic linking if compiled); the language should be rich enough to access all data structures and functions offered by the underlying system. These requirements almost conclude that the UPL has to be an object-oriented language. This language is the "shell" above all other language environments and the underlying CAD system, i.e., through the UPL, users will be able to access all the objects (data structures and functions) created in the APL or SPL and link with other objects created in the UPL, but not vice versa. Smalltalk [11] and TM [8] are two languages that are close to these requirements. However, they are designed to be general purpose programming languages. A good UPL for CAD users is yet to be evolved.

A primary protion of the entire system is probably implemented by writing programs in APL in order to keep the system independent of the target hardware and the machine level instructions. Since the nature of the system is large and long-lived, APL should address issues such as readability, maintainability, reusability and portability. The language should also generate efficient code for execution. The choices for the APL are Ada, Modular-2 and Mainsail. They all support strong typing, separate compilation, data abstraction, exception handing, etc. Ada deserves more considerations due to its strong standardization (supported by DoD) and many unique features such as generic, tasking and overloading.

The SPL is the closest language layer to the operating system and the underlying hardware. This languages should provide operators for bit manipulation such as turning bits on and off, perform left and right shifts etc., low level I/O such as sequential file I/O, random access I/O, storage allocating and freeing. Ability to generate very efficient code is the most important consideration. The choice of this language depends primarily on the underlying operating system and hardware. A few choices are C, Ada, Modular-2 and Mainsail. Quite frequently, some coding done in the assembly language for the underlying hardware is inevitable in order to get more speed.

Conclusions

Most CAD vendors provide the UPL simply as an extension to allow users accessing functions in the underlying system. Their objectives and requirements of offering these UPLs are not clear, as a result, most of these languages are the interpreted version of a simplied high level language with access to underlying routines. They may be easier to use compared to the APLs, but are limited as macro languages.

In this paper, several current language architectures are analyzed and the rationale of designing a new UPL for CAD is summarized. An architecture is proposed to allow the "mix and match" of various languages that are integrated through the data abstraction that allows objects (data structures and functions) implemented in the lower level to be accessed by the higher level environment. A few good candidates of languages are suggested with a more detailed model for electronics CAD/CAE systems. These languages are recommended based on the features and current implementations of the languages without further evaluations of individual business considerations. A good report [1] can be used as a guideline that includes other factors to be considered when selecting a particular language for application or system development. The design and implementation of a UPL requires further investigation. Some important issues are: a simple object-oriented language but powerful enough to produce a prototype system quickly; a good semi-automatic mechanism that translates the UPL code into APL and gains performance after the prototype programs are ready; good interfaces among the UPL to other application specific languages (e.g., VHDL [4]) that are later translated to APL.

References

[1] D. Baker, "Ada Decision Matrix," The Aerospace Corporation, El Segundo, Ca., March 1984.

[2] Calma, "Design Analysis Language Product Specification," GE-Calma Company, Santa Clara, Ca., 1982.

[3] Calma, "GPL-II Reference Guide," GE-Calma Company, Santa Clara, Ca., 1983.

[4] A. Dewey, "VHDL: Implication of a Modern Hardware Description Language," IEEE International Conference on Computer Design: VLSI in Computers, New York, 1984.

[5] C. Eastman and R. Thornton, "A Report on the Glide2 Language Definition," preliminary draft, IPP, Carnegie-Mellon University, Pgh., PA, Mar. 1979.

[6] C. Eastman and Y. Yasky, "The Integrated Building Model and Database Schema for the 2nd Phase of Integrated CAEADS," CAD-Graphics Lab., IBS., Carnegie-Mellon University, Pgh., PA, Jan. 1981.

[7] Formtek, "Formtek Reference Manual, Form:Draw," Formative Technologies Inc., Pgh., PA, 1983.

[8] J. Gerzso and A. Buchmann, "TM- an Object-Oriented Language for CAD and Required Database Capabilities," IEEE Workshop on Languages for Automation, New Orleans, Nov. 1984.

[9] C. Liu, "Drawings as Models for Design: a Computer Drawing System to Build Models Supporting Design Process through Abstractions," Ph.D. Dissertation, Carnegie -Mellon University, Pgh. PA, April 1984.

[10] C. Liu, "A Study of Graphics Programming Language for Computer-Aided Design and Drafting," IEEE Workshop on Languages for Automation, New Orleans, Nov. 1984.

[11] Special Issue on Smalltalk , Byte, Aug. 1981.