

AD-A252 462



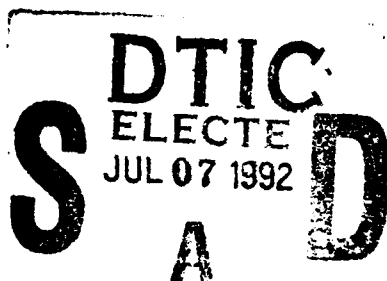
1

Integrating Commercial Off-The-Shelf Tools for Custom Software Development

MTP 92B0000002

June 1992

D. S. Blodgett
D. J. Phair



"Original contains color plates: All DTIC reproductions will be in black and white"

This document has been approved for public release and sale; its distribution is unlimited.

92-17593



MITRE

Bedford, Massachusetts

92 7 08 081

Integrating Commercial Off-The-Shelf Tools for Custom Software Development

MTP 92B0000002

June 1992

D. S. Blodgett

D. J. Phair

Contract Sponsor ESD
Contract No. F19628-89-C-0001
Project No. 589A
Dept. D072

Approved for public release;
distribution unlimited.

Accession For	
NTIS CRA&I	
DTIC TAB	
Unannounced	
Justification	
By	
Distribution /	
Availability Codes	
Dist	Availability Codes
A-1	Special

MITRE

Bedford, Massachusetts

ABSTRACT

Fourth-generation, object-driven languages (4GLs) (i.e., Hypermedia) have been used effectively in requirements analysis prototyping and human-machine interface development and have served as a front end to more complex applications. This paper describes the process of evaluating, selecting, and integrating 4GL tools for specific applications running on a variety of microcomputer platforms. To explore these different options, we will present an example of a computer-based training system created in a 4GL and describe how numerous commercial off-the-shelf software tools were integrated for added functionality. The logical extension to the current suite of Hypermedia products is tools capable of producing device-independent source code, which in turn is capable of being compiled into stand-alone applications. The impact of using a set of nonhomogeneous tools will be discussed in terms of source code control, supportability, and tool enhanceability.

ACKNOWLEDGMENTS

The authors of this document would like to thank John Wilson for conceiving and supporting this effort. Additionally, we appreciate the extensive management support and helpful suggestions that we received from Ed Fitzgerald, Steve Harris, Stu Jolly, and Dave White. We would also like to acknowledge the review and support of ESD. Finally, we would like to thank Pamela Guild and Brenda Proctor for early editing and formatting of the document. Additional thanks go to Roberta Carrara (J103) for making suggestions resulting in the final version.

TABLE OF CONTENTS

SECTION	PAGE
1 Introduction	1
2 Custom Software Development	2
2.1 Language Classifications	2
2.2 Advantages and Disadvantages	2
3 COTS Tool Kits	4
3.1 HMI Development Tools	4
3.2 Code Generators	7
3.3 Custom Development Platforms	8
3.4 Cross-Platform Development Tools	9
3.5 Multimedia Tools	9
3.6 Database Tools	10
3.7 Low-Level Interfaces	14
4 COTS Tool Integration	18
4.1 Integration Criteria	18
4.2 Core Requirements	18
4.3 Derived Requirements	19
4.4 Selection Process	19
4.5 Custom Development	20
4.6 Experience Base	21
5 Example of an Application Using COTS Tools	22
5.1 Ultrasonic Inspection Trainer	22
5.2 Real-Time Signal Generation Using Xcmds	24
5.3 Digitized Voice	24
5.4 Context-Sensitive Help System	25
5.5 Real-Time Digitized Video	27
6 Summary	29
7 Glossary	30

LIST OF FIGURES

FIGURE	PAGE
1 Sample <i>HyperCard</i> Application, Chem Inventory, 2.1	5
2 <i>Director</i> Working Environment	11
3 <i>Oracle</i> System Stack	13
4 <i>SQL*Plus</i> Screen	14
5 <i>HyperBasic</i> Xcmd Script	15
6 <i>CompileIt!</i> Development Environment	16
7 <i>CompileIt!</i> Debugger	17
8 COTS Tools Used in the Ultrasonic Inspection Trainer Prototype	22
9 Sample Radome Inspection Display	23
10 <i>MacRecorder</i>	25
11 <i>InterFACE</i> Agent Editor	26
12 <i>MediaGrabber</i> Environment	28

LIST OF TABLES

TABLE	PAGE
1 Sample Programming Language Classifications	3
2 Examples of Hypermedia Products	6
3 Examples of Code Generators	8
4 Sample Cross-Platform Development Tools	9
5 <i>Oracle</i> Support Utilities	12

SECTION 1

INTRODUCTION

Integrating commercial off-the-shelf (COTS) tools has proved successful in expediting the development of custom software applications. With the introduction of fourth-generation, object-based languages (4GLs) and the expanding array of commercially available tools, many developers have found an efficient alternative to traditional software development. That is, the requirements for many applications can be satisfied by selecting and integrating the appropriate set of commercial tools. In contrast, complete custom software development is costly, risk-intensive, and time-consuming. The integration of COTS tools has been successfully used in requirements analysis, prototyping, and human-machine interface (HMI) development.

In recent years, the price-performance ratio between workstations and microcomputers has closed dramatically. This has resulted in the widespread availability of economical tools capable of producing high-quality applications quickly and easily. We will describe the process of evaluating, selecting, and integrating 4GL tools that run on a variety of microcomputer platforms and discuss the impact a set of nonhomogeneous tools will have on development, supportability, and application enhanceability.

Also, we will present an example of a computer-based training system that is created in a 4GL language and describe how numerous COTS tools were integrated to add functionality. This example will provide insight into the selection and integration of COTS tools in the areas of multimedia, real-time processing, help systems, and simulation.

SECTION 2

CUSTOM SOFTWARE DEVELOPMENT

All software, to some extent, is custom software. Even if an application is constructed entirely using COTS tools, some customization will be necessary to tie the various tools together. A thoughtfully selected suite of COTS tools supplements, but does not completely replace, custom-developed software.

2.1 LANGUAGE CLASSIFICATIONS

Custom software normally refers to an application that is constructed using a traditional programming language such as C or FORTRAN. C and FORTRAN are third-generation (3GL) programming languages. There are four distinct classes or generations¹ of programming languages. A first-generation programming language refers to the stream of ones and zeros understood by the central processing unit (CPU). For example, 10011110 instructs an 8086 microprocessor to store the AH register in the flag register. Second-generation (2GL), or assembly languages, take the groups of ones and zeros known to the CPU and associate a mnemonic with each function. Assembly languages, the "lowest" level where practical programming can occur, are particularly useful for tasks which are hardware-specific or speed-critical. Low-level languages are arbitrarily grouped as assembly language and below; high-level languages are all others. The 3GLs, or procedural languages, use English-like statements to cause specific processing to occur and easily support structured programming, multiple data types, and complex processing algorithms. Most importantly, 3GLs remove the programmer from the details of hardware implementation. The highest level of programming is done with 4GL object-based languages. An object is a building block, a section of code that can be easily customized for a wide variety of tasks.

2.2 ADVANTAGES AND DISADVANTAGES

Assembly languages are used today primarily for real-time, performance-critical tasks. Assembly language programming, a tedious, error-prone task, lends itself poorly to structured programming and maintainability. Further, assembly language programming is cumbersome, since detailed knowledge of the underlying hardware implementation is required. As a result of their flexibility, the 3GLs are often the language of choice. Programming with 3GLs, because of their long-time popularity, is often referred to as traditional programming. The 4GLs reuse tried and proven objects to develop new applications, thereby making software development easier and more predictable. Programming is necessary only to connect the objects to the application, not to construct the objects or the application. Because less programming is required, applications can be created faster and more economically than using traditional

¹ The term generation refers to the era in which this class of language was most popular.

programming methodologies. The trade-offs with 4GLs are slower execution speed, lack of flexibility to implement complex algorithms and data structures, and the inability to process real-time data.

Each generation's language has a purpose for which it is best suited. In general, there is a direct relationship between the level of the language and the amount of control the programmer has over the CPU. Low-level languages offer the most control over the hardware and system resources, while 4GLs have the least. In most applications, an inverse relationship exists between the level of difficulty to accomplish a task and the language level. High-level languages support many more controls and building blocks than do low-level languages. Thus, the higher the language, the easier it is to accomplish most tasks. Also, an inverse relationship exists between development time and language level. Because low-level programming is more difficult and involved, a task developed using a high-level language takes less time to implement than one developed using a low-level language (see table 1).

Table 1. Sample Programming Language Classifications

Generation	Level	Code	Function	Language	Development Time
First	Low	10011110	Store the AH register in Flag	8086 machine language	Impractical
Second	Low	cmpa.l (a2),a1	Compare the contents of the value at the address pointed to by register a2 with the value stored in register a1	68000 assembly language	Long
Third	High	strncpy(f1,f2,(strlen(f2));	Cause the string f1 to contain the contents of f2	C	Average
Fourth	High	Put the date into line 1 of cd fld 1	Cause the date to be printed in a field on the screen	HyperTalk	Short

SECTION 3

COTS TOOL KITS

Before selecting the correct suite of COTS tools, it is important to be aware of the available technologies. In order to provide a brief overview of some of the most popular technologies and tool sets available today, we will focus on HMI development tools, code generators, cross-platform tools, multimedia tools, database tools, and custom-compiled modules. We have limited our discussion to microcomputer-based development tools, specifically Microsoft Windows 3.0 and Apple Macintosh-based applications. Although the principles discussed in this section apply to all platforms, we have found that these two platforms offer significant development power at an economical cost. Further, the proven graphical nature of these operating systems makes them naturally suitable for modern application development.

3.1 HMI DEVELOPMENT TOOLS

In recent years, 4GLs have been used extensively for HMI development. Many of these graphically based 4GL COTS tools are known as Hypermedia products. Hypermedia tools are comprised of a set of intrinsic objects that can be used to create sophisticated front ends for complex applications, develop useful stand-alone applications, or act as the glue between various COTS tools.

Claris Corporation's *HyperCard* (Macintosh), Silicon Beach Software's *SuperCard* (Macintosh), and Microsoft Corporation's *Visual Basic* (Windows 3.0) are three of the most popular Hypermedia products. On the Macintosh, products support the notion of projects², or stacks³, and cards. Under Windows 3.0, forms are the base object. Generalizing, a stack represents the application; a card or form represents one display within that application. User input is manipulated using predefined objects, such as buttons, fields, lines, and graphics. Under Windows 3.0, there is a slightly different set of objects which are referred to as controls. These objects all have different properties and can be set to generate messages based on user input. Unlike a traditional programming language, where all code resides in one place, each object within a Hypermedia product can have its own "script." A script is a set of actions that the programmer wants to occur when the object is activated. Each tool has its own proprietary scripting language. For example, *HyperCard* uses HyperTalk; *Visual Basic* for Windows 3.0 uses BASIC.

Figure 1 shows a sample *HyperCard* application. The hand cursor arrow points to the right arrow button whose HyperTalk script is shown. When the *HyperCard* application "catches" the mouse action, then its script is executed. In this case, the next entry in the inventory is displayed.

² *SuperCard*.

³ *HyperCard*.

Chem Inventory 2.1

Compound Name: (R)-(-)-Ethyl 2-pyrrolidone-5-carboxylate

Formula: C₇H₁₁O₂N **Mol. Wt.:** 157.17

Company Name: Aldrich **Catalog No.:** 30,978-8

Where is it? Lab 22 shelf #9

How much on hand? 5 g **Date Acquired:** 11/29/91
Date Reordered: 12/2/91

**Comments/
Safety notes:** In red bottle.

I/O? 0
MSD sheet? x
Usage record: Never used

New Entry **Return** **Find** **Delete Card**

Script of bkgn button id 55 = "right arrow"

```

on mouseUp
  visual effect wipe left
  go next card
end mouseUp

```

Figure 1. Sample *HyperCard* Application, Chem Inventory 2.1⁴

⁴ Chem Inventory 2.1, © 1990 Dr. Dan Swartling, 5640 S. Maryland Ave. #BSMT, Chicago, IL 60637.

Hypermedia is an extremely effective tool for implementing proof-of-concept rapid prototypes as well as operational prototypes. Hypermedia is used most effectively in requirements analysis prototyping with a small development team (one or two engineers) with frequent design reviews. The speed through which applications may be developed by one individual allows various ideas to be explored and prototyped. Further, because Hypermedia products exist in an interpretive environment, ideas can be explored in real time during design reviews. The small initial cost, short learning curve, and small development team make Hypermedia tools extremely cost-effective. A limitation to all Hypermedia products and 4GLs, in general, is their ability to process real-time data. To compensate, most Hypermedia products have "hooks" for externally developed commands, external commands/functions (Xcmds/Xfens) (Macintosh) and DLLs⁵ (Windows 3.0), to facilitate this function. An Xcmd is a section of custom software, usually written in C or assembly language, which performs a specific function such as hardware polling. (Table 2 lists sample Hypermedia products and our experience with the tools.)

Table 2. Examples of Hypermedia Products

Application	Platform	Flexibility	Ease of Use	Popularity	Performance
<i>HyperCard</i> Claris Corp.	Macintosh	★ ★ ★	★ ★ ★	★ ★ ★ ★	★ ★ ★
<i>PLUS</i> Spinnaker SW	Macintosh/ Windows 3.0	★ ★ ★	★ ★	★	★ ★
<i>SuperCard</i> Aldus	Macintosh	★ ★ ★ ★ ★	★ ★	★ ★ ★	★ ★ ★
<i>Toolbook</i> Asymetrix	Windows 3.0	★ ★	★	★ ★	★ ★
<i>Visual Basic</i> Microsoft	Windows 3.0	★ ★ ★ ★	★ ★ ★	★ ★ ★ ★	★ ★ ★ ★

Hypermedia is not without drawbacks. As applications become large, Hypermedia becomes difficult to maintain and expand. Because of the lack of one central program (i.e., the scripts are distributed under various controls), in many cases it is easier to rewrite than it is to modify. The fragmentation of scripts also results in Hypermedia applications that do not lend themselves well to large development teams. Another drawback is the lack of freedom to express complex data structures. In general-purpose processing, this is not an inconvenience; however, this imposes serious limitations as the complexity of the application increases. The trade-off for rapid development time and ease of implementation is lack of execution speed. Because Hypermedia is interpreted and not compiled, execution times are longer than with 3GLs. As microcomputers become successively faster, this limitation becomes less relevant.

⁵ Dynamic link library.

Additionally, the skill and experience level of the programmer can have a dramatic positive effect on the speed of the application. Finally, to increase performance in speed-critical applications, an Xcmd/DLL⁶ can be integrated smoothly. The final criticism of Hypermedia is that it tends to develop unrealistic customer and management expectations. While it may be possible to develop a semifunctional application in an extremely short period of time, it may not be possible to produce all applications using this productivity rate. For the reasons previously discussed, most applications cannot be completely developed exclusively with Hypermedia tools.

Although Hypermedia is not appropriate for every application, its future is extremely promising. As tools and development platforms become faster and more powerful, the arguments against Hypermedia become less compelling. Moreover, when applications call for a high degree of integration with other cutting-edge technologies such as multimedia, there is no better formula for developing a successful application than Hypermedia.

3.2 CODE GENERATORS

Code generators allow programmers to construct an elegant HMI for an existing or new application. Using a "what you see is what you get" (WYSIWYG) editor, that in many cases is similar to a Hypermedia development environment, an individual with little programming experience can construct and modify an application HMI. The way in which code generators differ from Hypermedia is that with a few keystrokes, the (third generation) source code, usually ANSI C or Pascal, is produced. Once the application has been converted to code, the programmer adds small sections of code to connect application code to the generated code. This code may then be compiled and linked together with existing or new code to form a new application. Early code generators would not save user-supplied code if the graphical portion of the application had been changed. Most modern code generators now support this feature so that the operator has only to add application-specific code once. While connecting the custom-developed code to the generated code is a nontrivial exercise, it is certainly easier than developing the interface from scratch. The main advantage of using a code generator is that it eliminates the need to master the implementation details of the graphical user interface (GUI) libraries (i.e., the Macintosh desktop, the Windows 3.0 Presentation Manager). Neuron Data's *Open Interface* will allow a programmer to develop an application on one platform and produce source code for virtually any other GUI platform. For example, a programmer could use *Open Interface* on a VAX/VMS-based machine and produce source code capturing the "look and feel" of an Apple Macintosh application. (See table 3 for examples of code generators.)

Code generators are relatively easy to use and provide a viable alternative to custom HMI development in cases where Hypermedia is unavailable or not appropriate. Code generators are particularly useful for porting an application from one platform to another (i.e., X-windows to Windows 3.0). For example, a program which calculates the number of source lines of

⁶ These small sections of code are nontrivial to develop.

Table 3. Examples of Code Generators

Application	Platform	Flexibility	Ease of Use	Cost	Performance
<i>Open Interface</i> Neuron Data	Multiple	★★★★★	★★	\$\$\$\$\$	★★★
<i>Prototyper</i> Smethers-Barnes	Macintosh	★★	★★	\$	★★
<i>Windows Maker</i> Blue Sky Systems	Windows 3.0	★★	★★★	\$\$	★

code in a program module written in ANSI C⁷ could be ported easily from Windows 3.0 to the Macintosh. Using a code generator, a programmer could construct an elegant HMI, produce the source code, and link together the application modules. Other uses for code generators are facilitating demonstration and unit testing (by developing test drivers) of embedded technology prior to integration. Further, code generators can be a useful starting point for programmers prior to application development.

While code generators are beneficial, they can be expensive and have drawbacks. Some of the more elaborate code generators such as *Open Interface* can cost upwards of \$5000 (depending on platform). Additionally, the code these tools produce imposes a structure on the development effort which is often difficult to build upon and maintain. Another drawback is that these tools often lack the flexibility found in Hypermedia. Currently, no commercial products exist that support functionality like Hypermedia and have the ability to produce device-independent source code. While *Open Interface* is capable of producing device-independent source code, it lacks the ability to attach actions to objects.

3.3 CUSTOM DEVELOPMENT PLATFORMS

The development environment that is used for custom software development can be one of the most critical decisions made during the entire course of the development effort. Exploring the vast range of implementation languages is beyond the scope of this discussion. The language that we have had the most success with is C; we now are experimenting with the object-oriented language C++. Our experience has shown C to be flexible and reliable over a variety of tasks and applications. Moreover, validated ANSI C compilers are available for almost any implementation platform.

⁷ Code written in ANSI C implies that the routines should compile without objection using any standard C compiler.

3.4 CROSS-PLATFORM DEVELOPMENT TOOLS

The future of computing is a more unified development environment. In the future, tools will be available that will allow applications to be developed independent of the GUI platform. As previously discussed, some of these tools are available today, notably *Open Interface*, *PLUS*, and *Oracle*. The advantage of such tools is obvious: applications can be developed and maintained on multiple platforms using the same source code. Currently, vendors decide which community to support with a product (e.g., VMS, OS/2, DOS, Macintosh). With platform-independent applications, vendors no longer will have to make those decisions.

In theory, there are no disadvantages to cross-platform applications. (See table 4 for names of tools and applications.) Actual implementations may not capture the "look and feel" of an interface, may be slow, or may have implementation "bugs." As the industry matures, cross-platform development tools will become more prevalent, functional, and cost-effective. A similar parallel can be drawn by comparing the state of multimedia computing five years ago to current cross-platform development tools.

Table 4. Sample Cross-Platform Development Tools

Application	Class	Features	Ease of Use	Cost	Performance
<i>DB-Vista</i> Raima Corp.	Database	★ ★	★ ★	\$\$	★ ★
<i>Open Interface</i> Neuron Data	Code Generator	★ ★ ★ ★	★ ★ ★	\$\$\$\$\$	★ ★ ★
<i>Oracle</i> Oracle Corp.	Database	★ ★	★	\$\$\$\$\$	★ ★
<i>PLUS</i> Spinnaker Soft	Hypermedia	★ ★ ★	★ ★	\$	★ ★

3.5 MULTIMEDIA TOOLS

Multimedia has become the buzz word of the early nineties. Although many theoretical definitions of this emerging technology have been formed, the most simple and inclusive definition can be found in its roots. Simply put, a multimedia application integrates various media forms to produce an end result. The media forms are typically live video, computer animation, and digitized sound. Because incorporating various media requires specialized hardware and software, multimedia development from scratch would be prohibitively time-consuming and expensive. Never before has the use of COTS tools been so warranted than in this area.

Multimedia has found its way into education, marketing, and presentations. In the educational arena, multimedia captures the student's attention and facilitates learning. For example, in

the early days of computer-based training, computers were used for drill and practice. The computer would display a question and ask the student to select the correct answer from a list of choices. State-of-the-art trainers now present the subject matter in a highly visual and engaging environment rather than in one that bores the student with a rote question-and-answer dialog.

Multimedia has also made its way into the corporate world. Multimedia presentations are used to convey ideas, sell products, and capture the attention of the audience. To do this effectively, multimedia saturates the audience with as much information as possible in the shortest period of time. By doing so, the audience, be it customers, corporate management, or others, will retain more information than with traditional briefing strategies. This technique is analogous to the television commercial, where an enormous amount of information is compressed into a 30-second spot.

Because personal computers have become more powerful in the last five years, multimedia applications are exploding into the matured mainstream of PC development. As hardware advances have been made, so too has the software. Multimedia support tools allow for the synchronization of digitized video, sound, and special effects to create stunning end products. Figure 2 shows an example screen from MacroMind Inc.'s *Director*. This tool allows the developer to create "scores" that coordinate various media and establish how a sequence will be played. The sequence can then be saved as a movie file and played from 4GLs such as *SuperCard* or *HyperCard*.

When selecting a multimedia development environment, one must look at the target application. If a self-contained system is desired (no external devices such as a VCR or Laser disk), real-time digitized video would be the most viable option. However, this type of video requires huge amounts of secondary storage as well as a powerful central processor. If more than a few minutes of video is required in conjunction with digitized sound, a computer-controlled VCR or Laser disk would be the most appropriate solution. Although computer-controlled video equipment adds to the overall cost of the system, the extra investment outweighs the performance degradation inherent with real-time digitized video. Computer-controlled video also allows for voice and music to be played without requiring additional computer resources.

3.6 DATABASE TOOLS

Often times, as applications become more complex, the amount of associated data grows. Data can take on many forms--it can be as simple as names and addresses of clients and as extravagant as facsimile images. Most computer applications share data between users and computers; therefore, database engines must be sufficiently sophisticated in design and function. Because sophisticated database functions are complex to implement, many developers have looked to COTS database packages to manage their data.

Today, most commercially available databases integrate easily with 3/4 GLs and have tools to define, create, and populate the database. Also, many manage multiuser accessing of the data by providing protection and locking mechanisms. Before selecting a COTS database package, you should answer the following questions to define your needs and allow you to choose the most suitable database for the application.

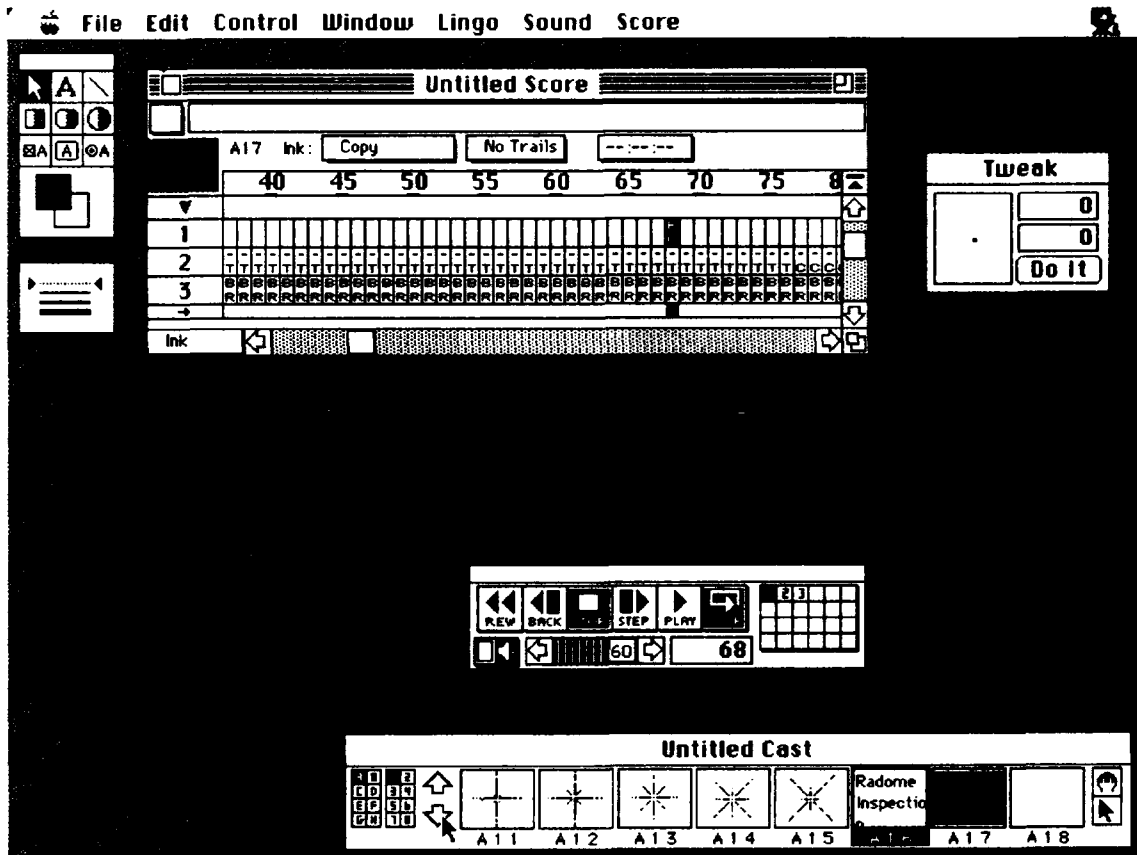


Figure 2. Director Working Environment

- How many users will be accessing the database concurrently?
- What level of data protection is needed?
- Will the database be centralized or distributed?
- Will the database run on different platforms?
- What type of licensing is needed?

In summary, if the application is targeted for use in a multiuser environment, networked or accessed on multiple platforms, you should consider taking advantage of the commercially available packages; otherwise, custom development is a viable option.

Although many commercial packages are available today, one of the most popular is Oracle Corporation's *Oracle*. *Oracle* provides a multiuser/multiplatform package that uses a standard

query language (SQL) as its access language. This package provides tools to create and manage custom databases based on the relational model⁸. On the PC, *Oracle* is supplied with a variety of support tools. For example, *SQL*Plus* provides an environment for writing scripts to perform queries, updates, report generation, and data structure definition tasks. *Pro*C* enables the use of embedded SQL commands in applications written in C and includes a set of function calls that can directly invoke *Oracle* processes. *Oracle* also comes with a complete set of utilities.

On the Macintosh, *Oracle* can be used in many ways. First, it can be called from a 4GL such as *HyperCard* or *SuperCard* using embedded *Hyper*SQL* commands in *HyperCard* scripts. Second, it can be integrated into C source code using embedded SQL commands and the *Oracle* pre-compiler *Pro*C*. Last, it can be called from the newly released front-end package *OracleCard*. (See table 5 for some examples of *Oracle*'s uses.) Each method has advantages and disadvantages. Using *Oracle* with *HyperCard* or *SuperCard* allows for rapid development, but suffers in performance. Calling the *Oracle* engine from a conventional programming language such as C increases performance, but lengthens development time. A strength of *OracleCard* is the ability to run on both the PC and the Macintosh. As with *HyperCard*, this method lacks performance for large database development applications.

The first step in creating an *Oracle* database on the Macintosh is to define the database structure using the System Stack tool, which allows you to manage user access, tables, views, and the database from within *HyperCard*. Figure 3 shows a sample screen from the System Stack tool. In this figure, the table name "Defect Info" is defined. Fields and attributes are defined by using the "Add Column" tool. Once the database is defined, it can be populated, queried, or updated via a *HyperCard* script.

Table 5. *Oracle* Support Utilities

Name	Purpose
CCF	Expanding the database
IMP, EXP	Backing up and reloading the database
SQL*Loader	Loading data that is stored in formats other than the ORACLE proprietary database format (SQL*Loader)
IOR	Performing database administration functions such as starting, stopping, and initializing the database

⁸ The relational model is based on the premise that data is perceived as tables, and operations generate new tables from old.

System Stack

Create a New Table

Table Name
Defect Info

ADD COLUMN

New Column:

Column Type

☐ Character
☐ Number
☐ Date
☐ Raw
☐ Long
☐ Long Raw
☐ Not Null

Add Column

Cancel

Column	Type	Null?
Defect Name	Char (12)	
width	Number (5,1)	Not Null
height	Number (5,1)	Not Null
graphic	Raw (1022)	
Date Created	Date	Not Null
Last Modified	Date	Not Null
Left	Number (3)	
Top	Number (3)	
Right	Number (3)	

OK

Cancel

Log Off

Options

User Management

Table Management

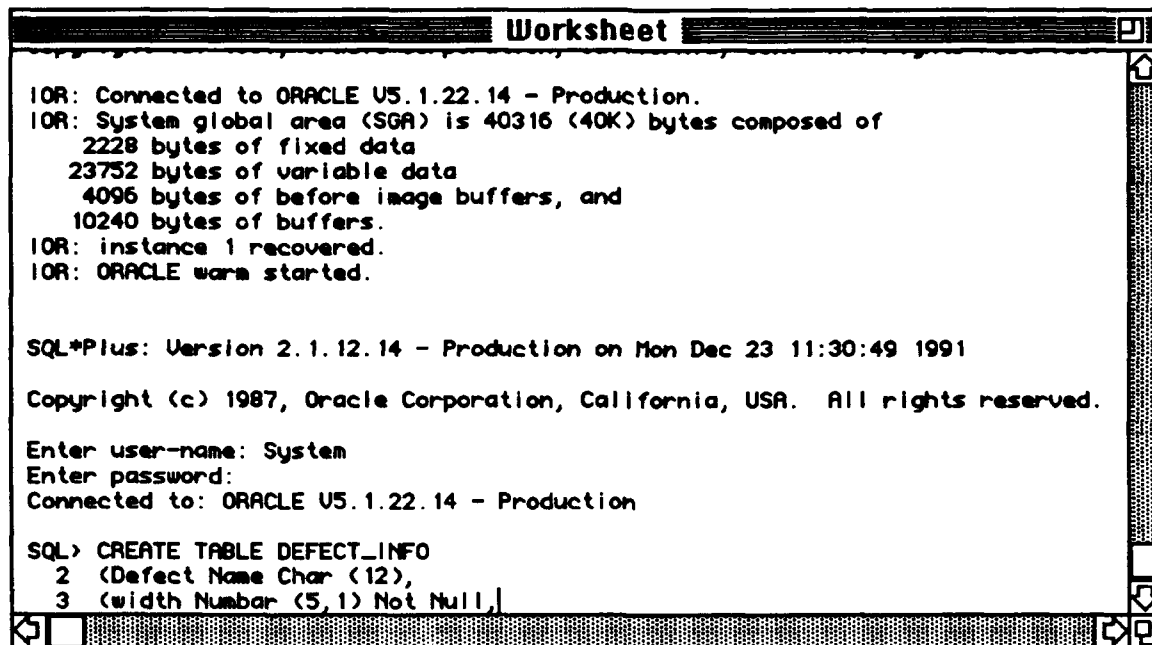
Database Management

Connectivity

SQL*Lab

Figure 3. Oracle System Stack

On the PC, the database table would typically be created using *SQL*Plus*, where the tables are created and populated via SQL commands. Figure 4 shows a sample *SQL*Plus* session. Notice that all commands are entered in textual SQL format.



```
Worksheet

IOR: Connected to ORACLE U5.1.22.14 - Production.
IOR: System global area (SGA) is 40316 (40K) bytes composed of
    2228 bytes of fixed data
    23752 bytes of variable data
    4096 bytes of before image buffers, and
    10240 bytes of buffers.
IOR: instance 1 recovered.
IOR: ORACLE warm started.

SQL*Plus: Version 2.1.12.14 - Production on Mon Dec 23 11:30:49 1991

Copyright (c) 1987, Oracle Corporation, California, USA. All rights reserved.

Enter user-name: System
Enter password:
Connected to: ORACLE U5.1.22.14 - Production

SQL> CREATE TABLE DEFECT_INFO
    2 (Defect Name Char (12),
    3 (width Number (5,1) Not Null,
```

Figure 4. *SQL*Plus* Screen

3.7 LOW-LEVEL INTERFACES

As applications have become more sophisticated, many vendors have realized the importance of using an open architecture in their software design. The trend in third-party software development is to find an unexplored niche in the software market and perfect it, rather than to develop a package that "does it all." As this trend continues, software packages increasingly will have to form with each other to create a successful application.

On the Macintosh, the concept of Xcmd addresses the issue of linking various COTS packages together. The Xcmds are external code resources which adhere to a standard calling convention and can be invoked from many 4GLs. Although originally developed to provide a link to compiled functions, Xcmds are now used as gateways into third-party software. Many Xcmds are now available for integrating packages and providing expanded "add-in" functions. Further, Xcmds can give software the ability to control hardware devices, perform complex animation sequences, and play computer-generated music. For example, an Xcmd supplied with NEC's *PC-VCR* allows the software to control the VCR via a call to the Xcmd. In the past, sample C or Assembler code would have been included in the manual, and the

programmer would have had the cumbersome task of making it work. With the advent of Xcmds, programmers simply move the Xcmd resource to the application's resource fork⁹ and call it from the application.

Figure 5 shows an example Xcmd script developed with *HyperBasic*. The script is written in a superset of the BASIC language and can be compiled to produce an Xcmd resource. *HyperBasic* provides full support for the Macintosh Toolbox¹⁰ and supports structures such as arrays, which most 4GLs do not. One drawback to developing Xcmds is that all variables are volatile and lose their values upon exiting the Xcmd. Because of this, techniques must be developed to allocate and manage shared memory. These techniques will be described in detail in the application example presented at the end of this paper.

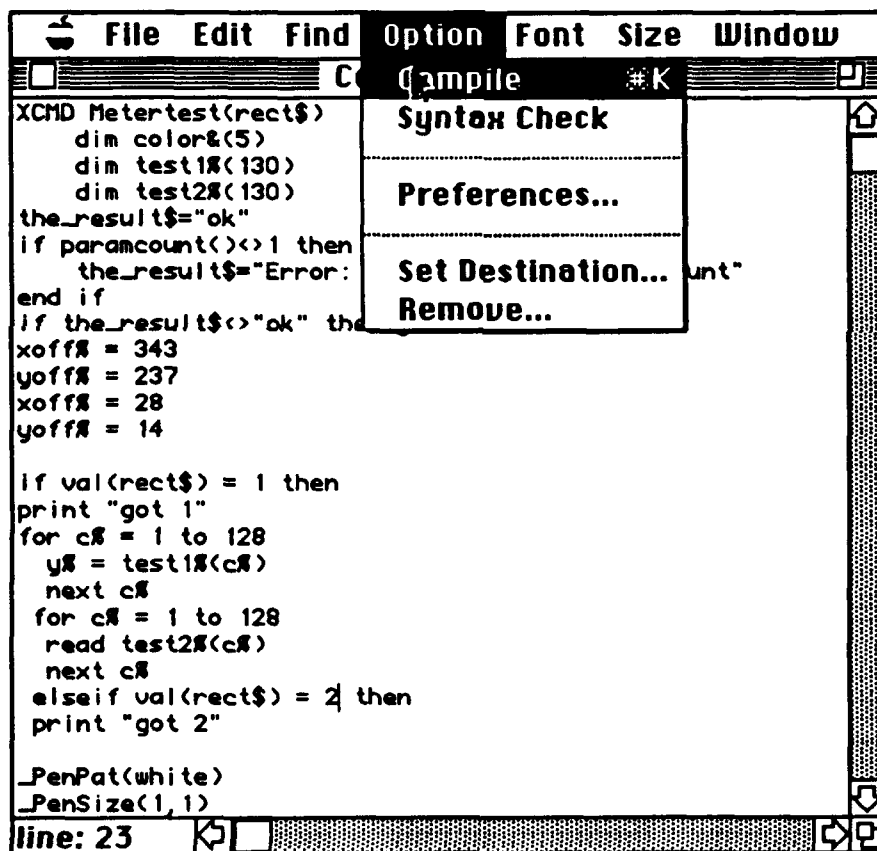


Figure 5. *HyperBasic* Xcmd Script

⁹ The resource fork contains objects termed resources such as menus, buttons, icons, and code segments.

¹⁰ The Macintosh Toolbox is a set of 900 routines located in ROM that help programmers design user-friendly applications and simplify access to the internals of the Macintosh operating system.

Besides bridging the gap between various third-party software, Xcmds provide a method for developing time-critical code which can be called from a 4GL script. Typically, 4GLs are interpreted and have poor performance when compared to compiled code. The Xcmds allow the programmer to develop compiled functions that can be called from a script, thus increasing the execution speed of the software.

Not long ago, Xcmds were considered to be a "magical" area of Macintosh software development; but recently, tools have become available that have greatly eased the task of Xcmd creation. The Xcmds can be written in C or Pascal or can be developed with packages such as Teknosys' *HyperBasic* or Heizer Software's *CompileIt!*. Developing Xcmds using 3GLs is a nontrivial task requiring a deep understanding of the Xcmd's calling conventions and parameter block. Developing Xcmds using tools created specifically for their creation is significantly easier.

Another tool that greatly eases the task of Xcmd development is *CompileIt!*. Its developers took a novel approach to Xcmd programming by allowing Xcmds to be written in HyperScript, a language similar to a 4GL script (see figure 6). With this approach, programmers do not have to learn a new language to develop Xcmds.

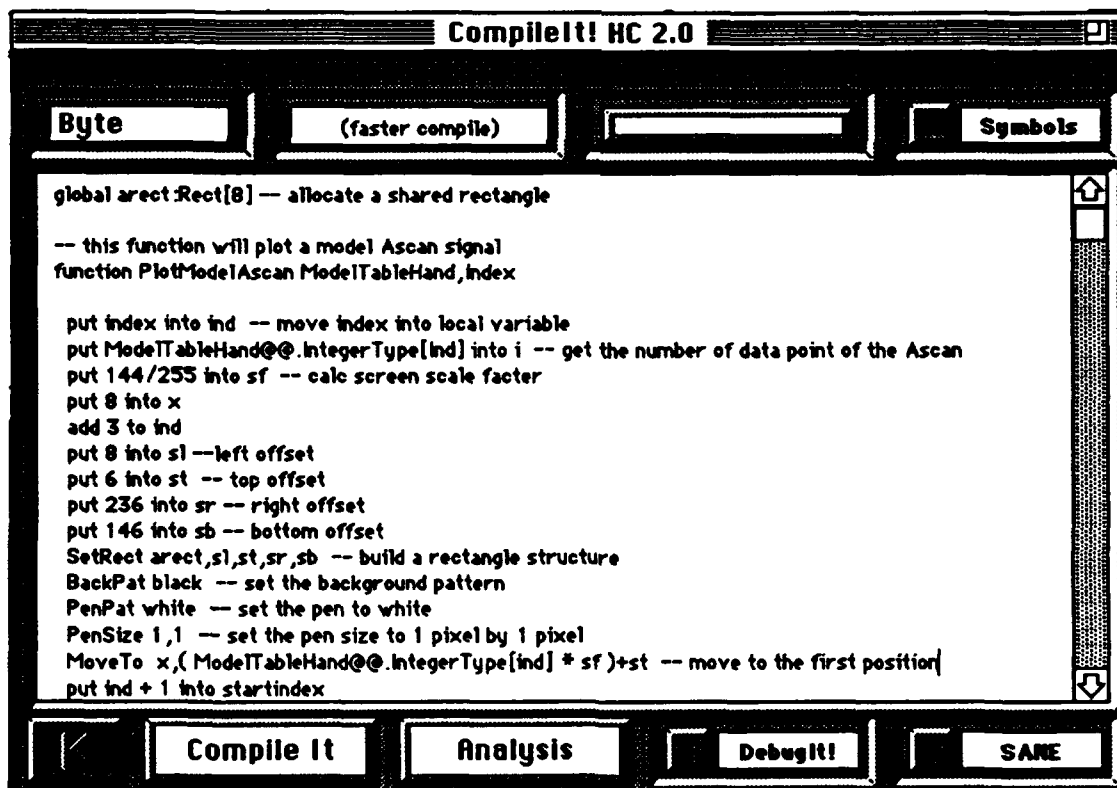


Figure 6. *CompileIt!* Development Environment

Like *HyperBasic*, *CompileIt!* provides for advanced structures as well as full compatibility with the Macintosh Toolbox. *CompileIt!* also supplies tools to analyze program deficiencies. One of the slowest mechanisms in Xcmd execution is what is termed a "text callback." A text callback occurs when the Xcmd needs the services of the calling program. When the services of *HyperCard* are needed, such as checking if the mouse is within an object, the Xcmd passes the "mousewithin" query back to *HyperCard*. *HyperCard* then answers with a True or False. In a text callback, many conversions must take place, resulting in slow processing. *CompileIt!* allows the user to analyze callbacks and points out areas that can be optimized. Another feature unique to *CompileIt!* is its sophisticated optional debugging facility. This utility attaches a symbolic debugger to the application's code resource that is activated upon entering the Xcmd. Figure 7 shows a sample screen of *CompileIt!*'s debugger. Within the debugger, the programmer can set and clear breakpoints, view and modify containers (variables), and review callbacks. In short, *CompileIt!* has opened Xcmd development to the novice Macintosh developer.

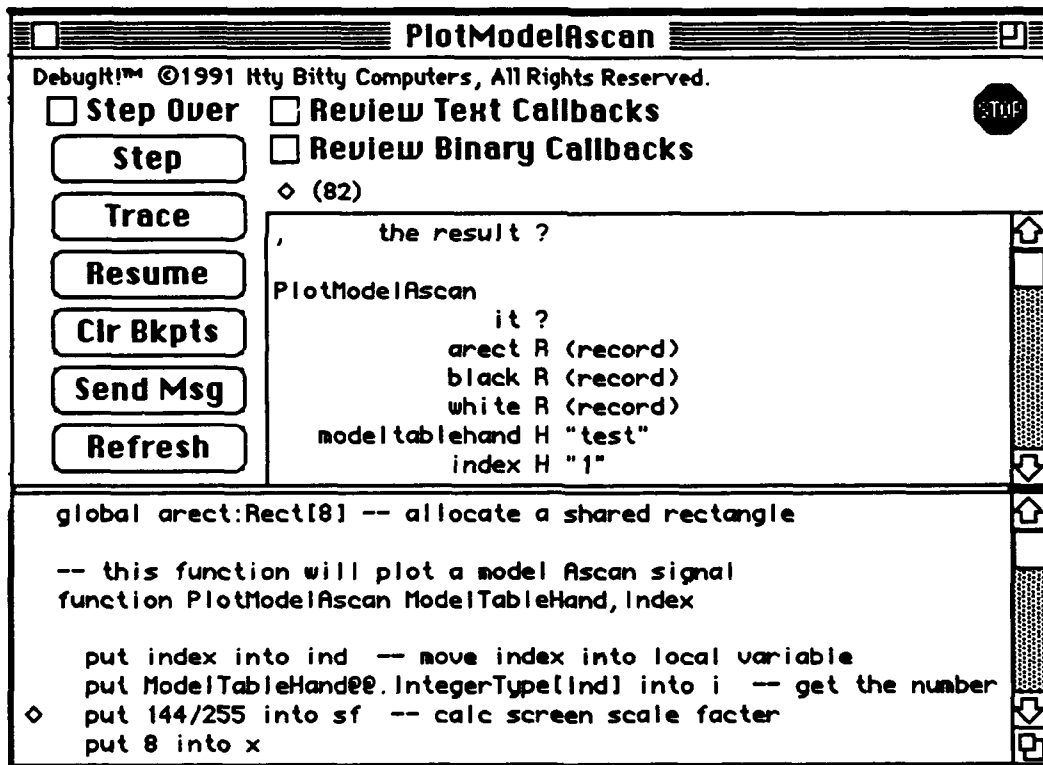


Figure 7. *CompileIt!* Debugger

SECTION 4

COTS TOOL INTEGRATION

The selection of a COTS tool should be driven by two related concerns: application requirements and tool suitability. The COTS integrator must provide a solution that satisfies performance and presentation requirements, and stays within the bounds of available funds and schedule. Application requirements identify the need that will be fulfilled by the new application. Tool suitability refers to how well the selected COTS tool fulfills the application requirements. The choice of tool set should be an iterative process. Application requirements will drive tool selection, while tool selection will impact application requirements. When employed thoughtfully, COTS tool integration capitalizes on the strengths and minimizes the weaknesses of individual tools, while providing the system designer an economical and efficient means of application development.

4.1 INTEGRATION CRITERIA

COTS tool integration is appropriate for most applications; however, it is important to only integrate proven tools which provide useful services. Applications designers live and die with the consequences of the tool set chosen. Poor performance and glitches in the COTS tool are inevitably mirrored in the integrated application. It is a poor practice to force functionality out of a COTS tool. The more convoluted the process, the more this suggests that the functionality should be implemented in another way. The only tools which should be integrated are those for which the vendor has provided a documented and graceful means of integration. To do otherwise only increases the chances of application failure or of producing an unusable application.

Custom development should be used as the application engine, the glue that ties all the COTS tools together. Custom-developed software should also be used, in most cases, for low-level data acquisition and speed-intensive tasks. It is possible to think of a custom-developed assembly language routine as a building block, like a COTS tool, to be integrated by the driver application. The driver application connects the various tools together and handles exceptions and abnormal processing. Finally, custom software development should be used in the event that no suitable COTS tool is available.

4.2 CORE REQUIREMENTS

From a computer science perspective, software provides a solution to some quantifiable problem in terms of core application requirements. Core requirements describe the basic function(s) of the system in very simple yet inflexible terms. For example, long-distance, telephone-switching equipment primarily connects callers from one point to another. In the telephone-switching example, a software designer could minimize the connection time using algorithms derived from advanced queuing theory. As the system design becomes more mature, the distinction between design and requirements blurs.

4.3 DERIVED REQUIREMENTS

Since design requirements are much less rigid than core requirements, the system designer is permitted much more freedom to tailor the system to the available tools and technologies. Tool selection can also feed the development of new core application requirements. The following examples demonstrate how new technologies employed in COTS tool-based design can alter and expand the focus of the application.

- Case 1: A research scientist wants to monitor an experiment that is connected to a data acquisition board and output the results to a video display in a user-friendly manner. The core requirement is real-time data acquisition; the design must support user-friendliness. A logical solution would be to develop custom (or interface to vendor-supplied) data acquisition routines in a high-level language such as C or assembly language. These routines could then be called through Xcmds from a 4GL language such as *SuperCard* or *Visual Basic*. This would exploit the rapid development time and flexibility of the 4GL language, while minimizing the impact of its inability to process real-time data.
- Case 2: Suppose the situation described above requires that the data be collected, stored, and analyzed over time. The solution in Case 1 could be expanded to provide an interface to a relational database such as *Oracle* or *DBVista*. The 4GL language would act as the front end to both the data acquisition tasks and the database. This capitalizes on the ability of a relational database to quickly create multiple views of data, while maintaining the user-friendly front end provided by the 4GL language.
- Case 3: Suppose the results of this experiment are also to be used for instructional purposes. For example, the scientist (who is also a college professor) now wants to present a lecture based on results of an experiment conducted previously in the lab. The system described above could be expanded by adding a multimedia product such as *Director* to play back the results, perhaps speeding up months of collected data into several minutes. With the addition of sound and other video effects, the professor could emphasize key points and provide context.

The three cases demonstrate the ability of tool selection to influence application requirements. Through the use of COTS tools, the application designer indirectly provided a smooth means through which the system could grow quickly and economically. While the original requirements never changed, the addition of cutting-edge technology, built around the original functionality, spawned a considerably enhanced application.

4.4 SELECTION PROCESS

The process of selecting a suite of COTS tools is both a technical and managerial problem. The choice of development tools is one of the most critical decisions that will be made during the entire course of a project's life cycle. There are no hard and fast rules for selecting a suite

of COTS tools; however, some application-independent heuristics can be applied. The first step is to assess the project environment using some "rules of thumb" considerations such as available staffing resources, staffing expertise, available financial resources, and project schedule.

An important consideration that is often overlooked is the staff experience base. Management should factor a learning curve into the application of any new technology. The more complex the tool, the more the experience with the COTS tools becomes relevant. Finally, while nonprogrammers can master specific software tools over time, it is an order of magnitude more difficult to master a programming language such as C or assembly language. Often, the success of COTS tool integration hinges on developing complex custom software to integrate the tools.

Much of the selection process is based on experience and the understanding of the application requirements. The relative ease and seamlessness of tool integration should not be taken for granted. A hands-on trial is the best test of ease of tool integration. Often, vendors will supply 30-day evaluation copies of software for just this purpose. Choosing a set of COTS tools should not be an arbitrary decision; it should be an iterative process. Finally, during the tool selection period, avoid jumping to conclusions about the relative suitability of a class of tools based on poor experience with one tool. Some considerations that should be weighed during the tool selection process are:

- Are the core application requirements satisfied?
- Have any new application requirements surfaced?
- Can the project realistically meet schedule and budget constraints?
- How compatible is the tool?
- Is the product maintainable?
- How smooth is the integration?
- How committed is the COTS tool vendor to supporting the chosen tool?
- How easily can the tool be "extracted" and be replaced in the future?

The chosen suite of COTS tools must provide an effective solution for the problem defined by the core application requirements. The degree to which integration can be accomplished depends on the application and must be evaluated on a case-by-case basis. From an integration perspective, the maintainability and seamlessness of the COTS tool implementation affect the quality of the end product most. The relative maturity of a technology or a particular product often determines the stability of the tool. Also, the reputation and size of the COTS tool vendor are often good clues to the quality of the product. In general, the larger the company, the more likely the product will be of high quality and available for the foreseeable future.

4.5 CUSTOM DEVELOPMENT

When a COTS tool satisfies an applications requirement, it is often advantageous to use it as a building block. However, there are occasions when integration is not the best solution. In cases where the integration will be less than seamless, integration should be avoided for two reasons. First, tools that are poorly integrated typically present an awkward HMI and are

difficult (and unpopular) to use. Second, poorly integrated applications are often characterized by "work-arounds" and "bugs." One of the worst possible scenarios is for the customer to make a mistake and get caught in between tools (i.e., unable to return to a known point). Poorly integrated applications are difficult to maintain and are often subject to unexpected behavior. Finally, any task that performs safety-critical processing should not be done by anything other than custom software.

4.6 EXPERIENCE BASE

Commercial software development is an inherently risky proposition. Successful software development organizations have developed large experience bases and are able to learn from failures and repeat successes. On a smaller scale, individuals and groups can apply this same logic. Individuals should be encouraged to experiment and innovate; organizations should strive to stay abreast of current technology trends and develop large experience bases. Trade shows and journals are excellent sources for this information.

SECTION 5

EXAMPLE OF AN APPLICATION USING COTS TOOLS

5.1 ULTRASONIC INSPECTION TRAINER

In this section, we present an example that demonstrates integrating numerous COTS tools to produce an end application. This example will focus not on the application but on the tools used in its creation.

The Air Force Ultrasonic Inspection Trainer is being developed to train aircraft radome inspectors. The objectives of this task are to improve the reliability of radome field inspection, to make the trainer engaging and easy to use for novice computer users, and to develop the trainer with minimum resources. The software is being developed on Apple Macintosh IIfx and Quadra computers with eight megabytes of memory. The trainer capitalizes on commercially available state-of-the-art multimedia and software tools. Figure 8 depicts the tools used in developing this application. Because of the large number of tools contained in this project, only a portion of the tools integrated will be discussed.

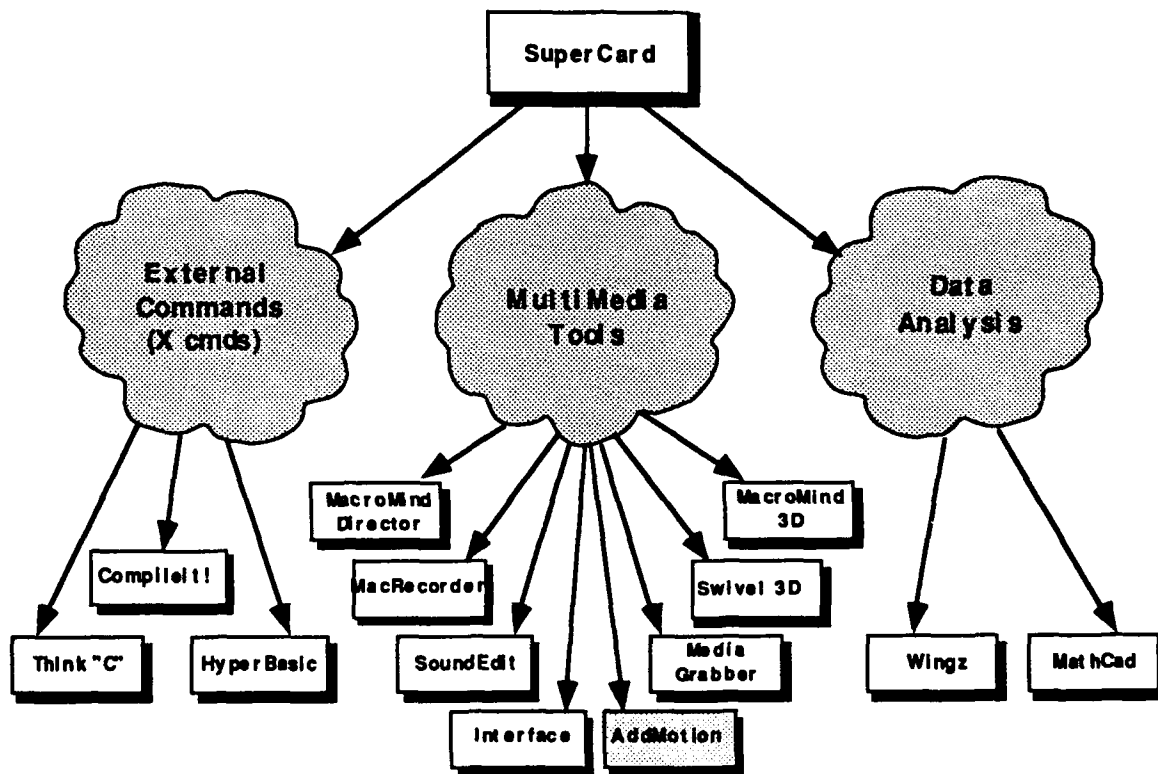


Figure 8. COTS Tools Used in the Ultrasonic Inspection Trainer Prototype

Figure 9 highlights several features of the *SuperCard* development environment. At the top of the display is a menu bar that allows the user to navigate to other displays and request specific functions. Within the vertical and horizontal scroll bars exists the enlarged view of the radome object, which allows the user to scroll to other areas of the radome image. The simulated ultrasonic meter shown in the bottom-right corner of the display is selected to show only the ultrasonic signal. The user can enlarge the meter view to access the various controls on the front panel via a menu selection. The signal that is displayed in the simulated meter is generated via a call to an Xcmd, a mechanism developed to achieve the required signal update time resolution. A small tool palette on the left side of the display provides the student immediate access to rulers, a defect marking tool, a defect logging tool, and the ultrasonic transducer. This figure shows that the student has selected the rulers and the defect logging tool to enter the attributes for a defect.

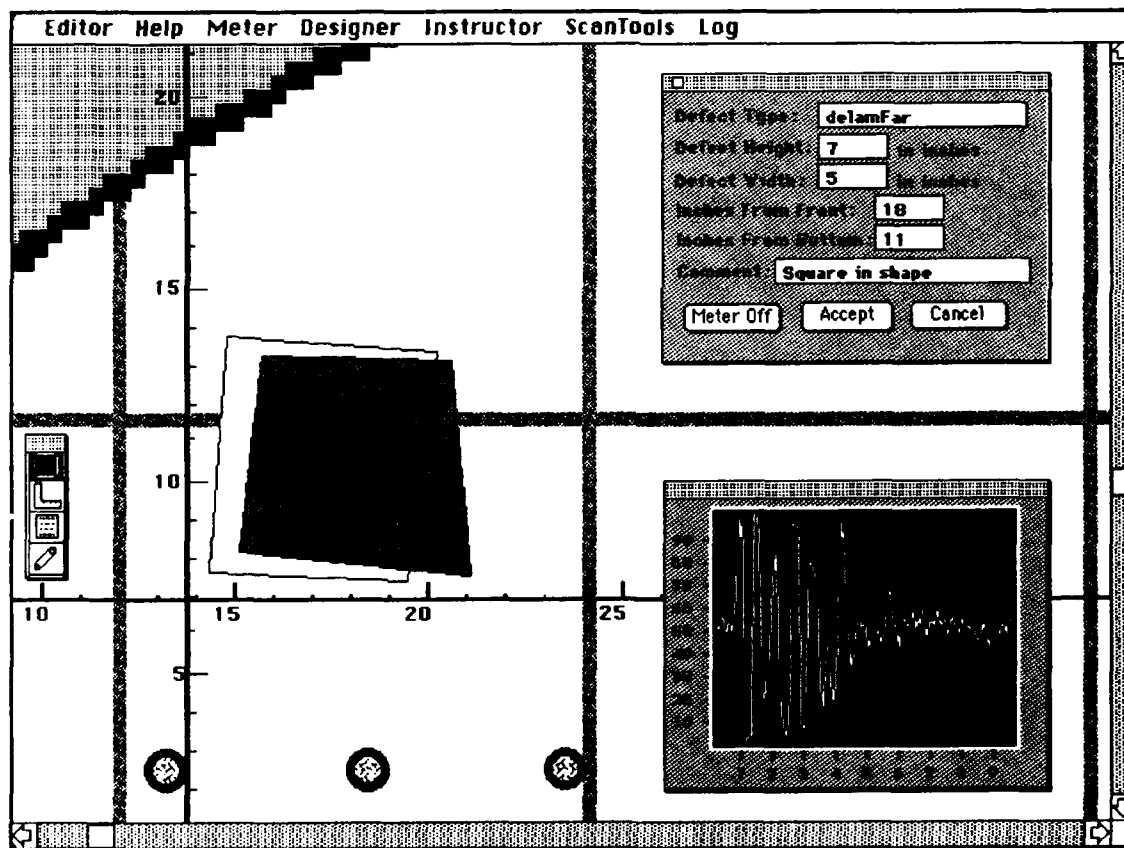


Figure 9. Sample Radome Inspection Display

SuperCard is the main tool used to develop this software. *SuperCard* is a 4GL similar to *HyperCard* but has expanded features such as sophisticated color objects, a powerful script editing environment, and a symbolic debugger. Three classes of COTS packages were used to develop the trainer in figure 8. First, Xcmd tools were used for real-time signal processing and other time-critical tasks. Second, multimedia COTS tools were added for the context-sensitive help system and animation for opening screens and tutorials. Last, data analysis tools were used for preprocessing the ultrasonic data obtained in the field.

5.2 REAL-TIME SIGNAL GENERATION USING Xcmds

This application incorporates data collected from field inspections into the trainer software. By doing so, the student can evaluate "real" signals prior to performing an actual inspection. Because actual data is used, the signal-generation software has to be plotted in real time. This requires developing Xcmds that would apply filters to the signal. The main tool for Xcmd development is *CompileIt!*. Because Xcmds cannot retain their value and because passing large amounts of data between *SuperCard* and the Xcmd is slow, a technique was developed to allocate, initialize, and dispose of the data points. This technique makes extensive use of Macintosh Toolbox calls to manage memory.

The first step in creating the Xcmd is to allocate and initialize memory with the data points. To do this, a handle¹¹ is created that refers to the pointer of the first data point. Because *CompileIt!* can manage structures such as arrays, accessing the data point is easy. As mentioned earlier, Xcmds cannot retain any values, so the handles themselves must be passed back to *SuperCard* for safe-keeping. The advantage to using handles, as opposed to passing points, is that only eight bytes, rather than thousands, need to be passed. Once the array of data points has been set up, *SuperCard* can call the Xcmd passing the handle a parameter and manipulate any point via *CompileIt!*'s built-in structure conventions. One drawback to this technique is if the software unexpectedly quits without disposing the handle, problems will arise because of dangling pointers.

5.3 DIGITIZED VOICE

To make the trainer software captivating and user-friendly, digitized sound is used extensively throughout the software. To accomplish this, Farallon Computing's *MacRecorder* and *SoundEdit* are used. *MacRecorder*, an external hardware device used to capture digitized sound, is accompanied with a sound editor and an Xcmd utility for playing the sound via a 4GL. Figure 10 shows the *SoundEdit* screen which allows the developer to record and manipulate the sound. *SoundEdit* also provides a wide array of effects (filters) that can be applied to the sound. For example, an echo filter can be applied to the voice sample to create an echo chamber effect. The developer can also rearrange words by simply cutting and pasting the sound spectrum.

¹¹ A handle, a safe double de-referencing mechanism used to access data, is maintained by the Macintosh memory manager.

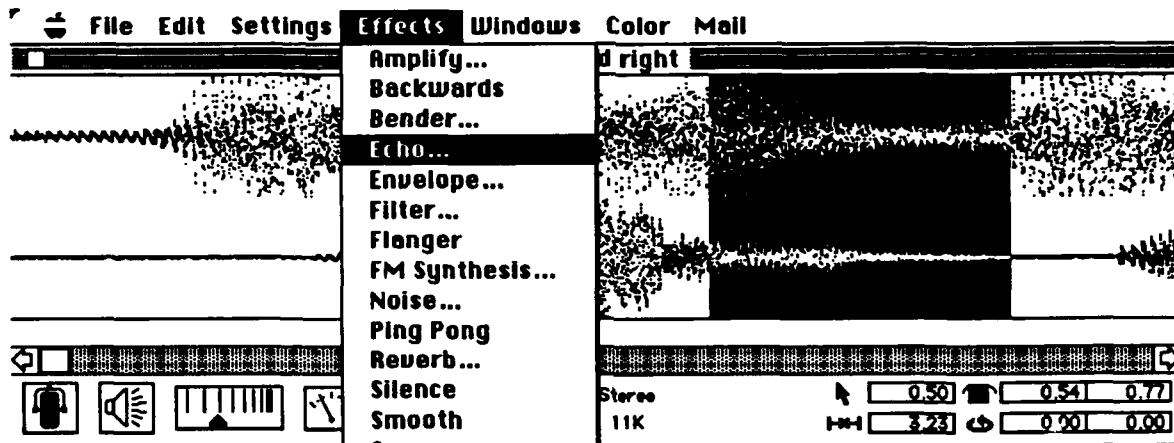


Figure 10. *MacRecorder*

Once the sound has been defined, it is saved as a “snd resource” and imported into *SuperCard* via a menu selection. Next, an Xcmd for playing sound is imported into *SuperCard*. To play the sound, the programmer simply enters the following script: `PLAYSND` “INTRODUCTION.” The number of sounds is limited only by the amount of available memory; but caution must be used, because digitized sound requires huge amounts of space.

5.4 CONTEXT-SENSITIVE HELP SYSTEM

In keeping with the objective of providing a captivating interface for novice computer users, we used a package called Bright Star Technology’s *InterFACE* (INTERactive Facial Animation Construction Environment) to develop an on-line help system. *InterFACE* provides tools to create on-screen talking agents that interact with and guide users through the training system. The *InterFACE* package consists of two major products: a complete graphical editing environment that assists developers in creating the customized on-screen agent, and voice synchronize. Agent images can be drawn with these supplied tools or created by importing digitized images captured with a video camera or scanning device. To create a new agent, the developer draws or captures the agent’s face with the mouth position resembling the supplied template. Figure 11 shows the agent editor with the agent’s mouth position saying the letter “F.” For each letter or sound, the developer must draw the agent with the correct mouth position. A total of 32 agent images must be created to produce reasonable, simulated mouth-to-voice synchronization.

Once all the agent’s expressions have been drawn, the developer may then synchronize the agent’s facial movement with digitized voice. To do this, the developer simply imports the digitized voice segment and types in the spoken sentence. *InterFACE* converts the text to phonemes and then to its internal language. This internal language consists of phonetic symbols plus facial expression codes. Finally, the user must test the end result and perform

timing adjustments as necessary. If digitized voice is not required, the *Macintalk* voice synthesizer software could be used. Although the *Macintalk* driver software does not produce the same degree of definition as digitized voice, it uses far less memory and requires little or no synchronization.

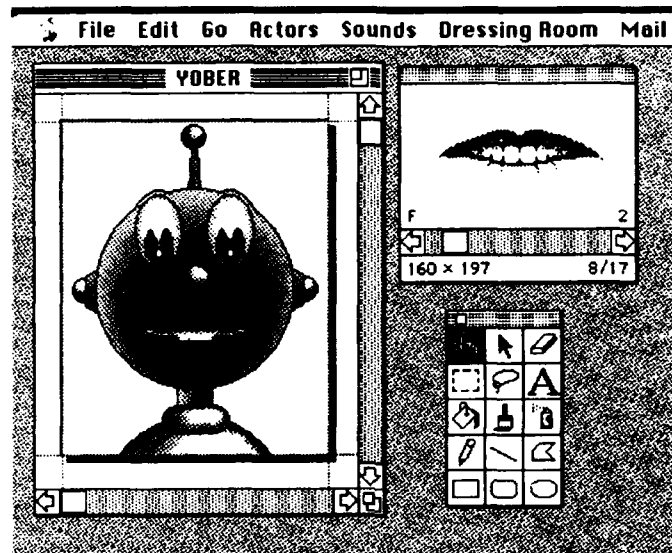


Figure 11. *InterFACE* Agent Editor

InterFACE may also be called from traditional programming languages such as C and Pascal or from 4GLs. A driver, called *RAVE*, is placed in the system folder of the Macintosh and is activated via low-level calls. In *SuperCard*, the supplied Xcmd is called to activate the talking agent. For example, to wake the agent and have it say, "Hello can I help you," the following command would be invoked:

RAVE{Hello Can I Help You.}.

The *RAVE* Xcmd will then call the *RAVE* driver, which will control the computer-generated voice and synchronization of facial expressions. Another useful feature of the package is the ability to have multiple agents interacting on the screen simultaneously. Thus, agents can be programmed to talk with each other during tutorials or informational sessions.

5.5 REAL-TIME DIGITIZED VIDEO

Real-time digitized video is used to create tutorials for the trainer software. The tutorial shows proper scanning procedures and test equipment calibration steps. To create real-time digitized video, a special video card and supporting software must be used. We used the RasterOps' 24XLTV card. This card supports 24-bit color and has the ability to capture real-time video from video cameras and VCRs. The RasterOps card also comes with *MediaGrabber* software for capturing frames of video to secondary storage. Figure 12 shows a sample screen of the *MediaGrabber* utility.

In this example screen, 10 frames of video are being captured at the rate of 1 frame per second. Depending on the bit depth of the image, up to 15 frames per second can be captured. (The standard frames per second of a VCR or video camera is 30 frames.) Compression boards are slowly becoming available to capture true real-time video. Even with compression, digitized video is extremely memory-hungry. Five minutes of 24-bit video captured at 30 frames a second can take almost a gigabyte of disk space. Though compression technology has made advances to reduce this figure, caution should be used when using real-time digitized video.

Once the video frames have been captured on disk, *Director* combines all captured frames into a video sequence. Overlaid titles and sound can then be added to the video sequence to create the desired effect. The combination of digitized video segments, sound, and titles is then saved as *Director* "movies," which can be played through an Xcmd movie player supplied with *Director*.

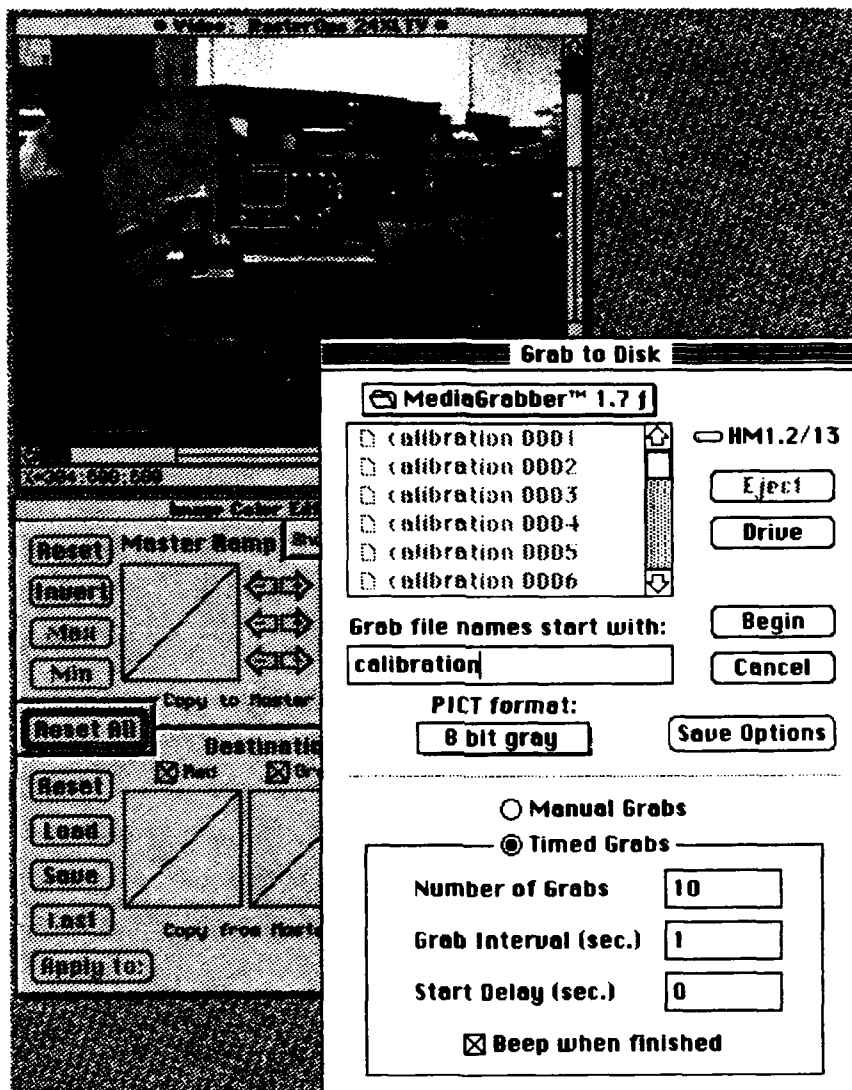


Figure 12. *MediaGrabber* Environment

SECTION 6

SUMMARY

We described some of the tools commercially available at the time this paper was published. Given the recent focus on COTS tools, a vast new selection of tools will undoubtedly become available in the near future. Our intent is not to promote a particular product, but to give insight by example. When selecting COTS tools for your own project, consider the following:

- What are the memory requirements of the tool?
- Does the tool provide an interface to the controlling shell software?
- How will the tool perform with other COTS tools?
- What are the licensing requirements?
- Is any special hardware needed?
- How dedicated is the company to the product (enhancements, upgrades, technical support)?
- What time period is required to come up to speed *with the tool*?
- What is the likelihood that the tool can be applied to other projects?

Having answered these questions, you will select the appropriate tool confidently.

GLOSSARY

COTS	commercial off-the-shelf
CPU	central processing unit
DLL	dynamic link library
GUI	graphical user interface
HMI	human-machine interface
<i>InterFACE</i>	INTERactive Facial Animation Construction Environment
ROM	read-only memory
SQL	standard query language
WYSIWG	“what you see is what you get”
Xcmd	external command
Xfcn	external function
2GL	second-generation (programming) language
3GL	third-generation (programming) language
4GL	fourth-generation (programming) language