

**AN AUTOMATED AGENT-LIKE
SYSTEM FOR EDUCATORS TO
DEVELOP EDUCATIONAL PACKAGES**

by

JOHN RICHMOND MATHEWSON., B.A., M.A.

**SCHOOL OF COMPUTING AND ADVANCED TECHNOLOGIES,
UNIVERSITY OF ABERTAY, DUNDEE**

**SUBMITTED FOR THE MASTERS DEGREE IN COMPUTING AND
INFORMATION TECHNOLOGY**

15 DECEMBER 2004

CONTENTS

| | | |
|----------|--|-----------|
| 1 | INTRODUCTION | 1 |
| 1.1 | THE CENTRAL QUESTIONS TO WHICH THIS THESIS ATTEMPTS TO PROVIDE ANSWERS | 4 |
| 1.2 | SUMMARY OF CHAPTER CONTENTS | 7 |
| 2 | LITERATURE AND SOFTWARE REVIEW: SOFTWARE LEARNING, OBJECT ORIENTED LANGUAGES, COMPUTER INTERFACES AND HOW THESE HAVE COME TOGETHER IN SOFTWARE AUTHORING PACKAGES | 11 |
| 2.1 | OVERVIEW | 12 |
| 2.2 | SOFTWARE LEARNING AND OBJECT ORIENTED LANGUAGES. | 13 |
| 2.3 | OBJECT BASED LANGUAGES | 16 |
| 2.4 | COMPUTER INTERFACES, THE CONTAINER METAPHOR AND EXPERT SYSTEMS | 17 |
| 2.4.1 | COMPUTER INTERFACES AND THE CONTAINER METAPHOR | 18 |
| 2.4.2 | EXPERT SYSTEMS | 19 |
| 2.4.3 | COMPUTER INTERFACES AND EXPERT SYSTEMS | 23 |
| 2.5 | ATTEMPTS AT PROVIDING END-USERS WITH A 'PROGRAMMING-FREE' INTERFACE: THE HYPERCARD MODEL | 24 |
| 2.6 | ATTEMPTS AT PROVIDING END-USERS WITH A 'PROGRAMMING-FREE' INTERFACE: OTHER ROUTES | 38 |
| 2.7 | WHAT HAS BEEN LEARNT | 43 |
| 2.8 | CONCLUSION | 44 |
| 3 | LITERATURE AND SOFTWARE REVIEW: CONCEPTS OF COMPUTER LITERACY, TYPES OF EDUCATIONAL SOFTWARE, INTERFACE CONSIDERATIONS, AGENTS, VIRTUAL | 45 |

| | | |
|----------|---|------------|
| | PERSONALITIES AND WIZARDS | |
| 3.1 | CONCEPTS OF COMPUTER LITERACY, TYPES OF EDUCATIONAL SOFTWARE AND INTERFACE CONSIDERATIONS | 45 |
| 3.2 | AGENTS, VIRTUAL PERSONALITIES AND 'WIZARDS' | 52 |
| 3.3 | WHAT HAS BEEN LEARNT | 59 |
| 3.4 | CONCLUSION | 60 |
| 4 | GENERAL METHODOLOGY | 61 |
| 4.1 | WORKSHOPS AND PROTOTYPING | 61 |
| 4.2 | QUESTIONNAIRE DESIGN | 74 |
| 5 | ANALYSIS OF QUESTIONNAIRE RESPONSE | 82 |
| 5.1 | SITUATIONAL QUESTIONS | 84 |
| 5.2 | INDIVIDUAL COMPUTER USAGE | 85 |
| 5.3 | EDUCATIONAL COMPUTER USAGE | 86 |
| 5.4 | PROGRAMMING EXPERIENCE | 88 |
| 5.5 | INTERFACE FAMILIARITY | 89 |
| 5.6 | VIRTUAL AGENTS AND YOU | 92 |
| 5.7 | CONCLUDING REMARKS | 93 |
| 6 | ANALYSIS OF RESPONSE TO WORKSHOP #1 | 95 |
| 6.1 | THE WORKSHOP | 95 |
| 6.2 | LESSONS LEARNT | 99 |
| 7 | ANALYSIS OF RESPONSE TO WORKSHOP #2 | 100 |
| 7.1 | THE WORKSHOP | 100 |
| 7.2 | LESSONS LEARNT | 104 |

| | | |
|-----------|---|------------|
| 8 | INTERFACE CONSIDERATIONS | 106 |
| 8.1 | MENUS | 106 |
| 8.2 | WINDOWS | 112 |
| 8.3 | ICONS | 116 |
| 8.4 | COLOUR | 117 |
| 8.5 | BEHAVIOUR | 118 |
| 8.6 | LANGUAGE | 119 |
| 8.7 | 'WIMP' OR 'OBJECT-POINT/CLICK' | 119 |
| 9 | DESCRIPTION OF NEW SOFTWARE CREATION INTERFACE | 120 |
| 9.1 | ASSUMPTIONS | 120 |
| 9.2 | THE PROGRAMMING ENVIRONMENT | 121 |
| 9.3 | SOFTWARE START UP | 121 |
| 9.4 | THE FEATURE TIER | 131 |
| 9.5 | CONCLUDING REMARKS | 132 |
| 9.6 | WHAT IS MISSING | 133 |
| 10 | CONCLUSION | 135 |
| 10.1 | CONCLUDING REMARKS | 135 |
| 10.2 | FUTURE WORK | 136 |
| 11 | BIBLIOGRAPHY | 138 |
| 11.1 | LITERATURE | 138 |
| 11.2 | SOFTWARE | 142 |

APPENDICES

| | | |
|-----------|--|-----------------|
| 12 | NEW SOFTWARE CREATION INTERFACE CODE SNIPPETS | 146 |
| 12.1 | START UP | 146 |
| 12.2 | SCREEN 2 | 147 |
| 12.3 | PROGRAM SIZER BUTTONS ON SCREEN 4 | 147 |
| 12.4 | PROGRAM BACKGROUND COLOUR BUTTONS ON SCREEN 5 | 150 |
| 12.5 | SELECTING AN EXIT BUTTON ON SCREEN 6 | 150 |
| 12.6 | CHOOSING AN EXIT BUTTON POSITION ON SCREEN 7 | 152 |
| 13 | WORKSHEETS FROM WORKSHOP #1 | 154 |
| 13.1 | AGENTS: WORKSHOP WORKSHEET 1 | 154 |
| 13.2 | AGENTS: WORKSHOP WORKSHEET 2 | 156 |
| 14 | WORKSHEET FROM WORKSHOP #2 | 157 |
| 15 | QUESTIONNAIRE | 160 |
| 16 | WORKSHOP CONSENT FORM | 174 |
| 17 | QUESTIONNAIRE CONSENT FORM | 175 |
| 18 | CD-ROM CONTAINING MOVIE FILES OF NEW SOFTWARE CREATION INTERFACE IN USE AND NEW SOFTWARE CREATION INTERFACE | ATTACHED |

1. INTRODUCTION

Software authoring package developers have always presumed that end-users of their packages:

- have an understanding of the principles underpinning computer programming,
- feel comfortable using computers,
- have both the time and inclination to invest effort in learning the in-house programming language of each authoring package.

While the vast majority of software authoring package manuals have claimed that there should be almost no time between installing the software on one's computer and being able to develop sophisticated, content-rich software this promise has not born fruit.

The literature and software review examines a number of these packages and the claims of their developers.

This thesis will claim that while many educators would like to develop bespoke computer software for content delivery and reinforcement, they are largely unable to without having to spend considerable time and effort learning programming languages or the necessary skills to work with one or more of the 'authoring' packages commercially available.

This thesis will also claim that many educators are unable to develop bespoke computer software for content delivery and reinforcement because while computer software “should be usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.” (Constantine and Lockwood 1999, p.47) it is not.

Part of the reason for this may be nature of the comparatively standardised WIMP (Windows-Icons-Menu-Pointer) graphic user interface (GUI) used across all personal computer operating systems. Other reasons may lie with the fact that even the commercially available authoring packages require both an understanding of the concepts underlying computer-languages and considerable time to be devoted to learning the proprietary languages associated with authoring packages.

The standardised GUIs now in use (Microsoft Windows, Macintosh OS, most forms of Linux, RISC OS) already place a layer between the user of a computer and the underlying machine-code (it should perhaps be remembered that VDU screens have not always been the standard user interfaces of computers).

This thesis proposes the development of either an additional or

different layer to replace the both current WIMP GUIs' and current authoring packages' dependence on a knowledge of programming in proprietary languages.

The model proposed in this thesis aims at fulfilling some of the benefits of successful IT-integration listed in the Learning Technology Dissemination Initiative's report (Stoner. Ed. 1996):

"Successful integration will lead to:

- Students learning more.
- Students learning better *how* to learn.
- Increased computer literacy among students (and staff).
- Increased computer confidence among students (and staff).
- Greater variety in the learning environment for both students and staff.
- Better pass rates.
- More interesting work."

(Stoner. Ed. 1996, p.20)

This thesis will also take issue with what constitutes computer literacy, whether 'computer literacy' is really a relevant or necessary outcome of using computers to deliver educational content, and why busy teachers should need to become 'computer

literate' to develop bespoke computer software for content delivery and reinforcement.

To support this thesis the author has developed a cross-platform (MicroSoft Windows, Macintosh OS, Linux) automated agent system that provides an alternative (non-WIMP) GUI to those currently available for developing bespoke computer software for content delivery and reinforcement. The point of this system is that 'computer literacy' is required of neither the person who composes the software for content delivery or the pupil/student who is the target of the end product.

This GUI has been used by Scottish primary teachers with no experience of computer programming or developing software for content delivery either with or without authoring packages. It has been modified to fall in line with their feedback and expectations.

Two focus group workshops were held at Greyfriars Primary School in St Andrews using the automated agent system within Microsoft Windows 98. Focus group participants were requested to develop a piece of software for content delivery using the system. Focus group participants also took part in a discussion relating to the value of virtual personalities and their potential role in assisting teachers in handling software preparation suites.

Underpinning this thesis is an examination of the history of the development of software systems that can learn. Particular attention is paid to Program By Example / Demonstration (PBE/D) systems.

1.1 THE CENTRAL QUESTIONS TO WHICH THIS THESIS ATTEMPTS TO PROVIDE ANSWERS.

Software authoring packages (as opposed to programming languages that are not embedded, such as FORTRAN and BASIC) have developed simultaneously to the *de facto* WIMP-GUI standard. It may be that the dependence on this WIMP-GUI model has hampered the development of a software authoring package that is genuinely

“usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.”

(Constantine and Lockwood 1999, p.47)

The literature and software review examines the role of the WIMP-GUI in how software authoring packages have been developed.

The following questions are felt to be central to both this examination and this project:

- Should the authoring interface and the standard interface be both

visible from the outset so that the user becomes comfortable with the presence of the standard interface right from the outset and therefore has no difficulty weaning him/herself away from the authoring interface and onto the standard interface?

- Should an authoring interface initially totally obscure the standard interface, and as the user becomes increasingly competent slowly become more 'transparent' so the user can slowly become a programmer who no longer requires the services of the authoring interface?
- Should an authoring interface totally replace a standard interface and automate such a large proportion of the programming process that the user never learns how to use the standard interface - merely learns how to manipulate the authoring interface to develop what he/she needs?

Does the last question really beg the question about what constitutes a standard interface?

- Should it be that an automated agent system is the standard interface both for software creation and media delivery?

As Microsoft Windows until comparatively recently was more a windowing environment and user interface floating on Microsoft

DOS than a free-standing operating system it was entirely possible for those who found it intrusive to access the underlying DOS system directly.

Macintosh OS X (despite Apple's claims to it being a free-standing operating system) is similar in that it is underpinned by BSD UNIX. Those who wish to can access the UNIX layer.

There seems no reason to object to the idea of a three-tier system whereby an automated agent system plays a similar role, overlying a WIMP-GUI that users can access should they wish.

On asking participants in Workshop #1 what system they had on their computers at home the universal answer was "Microsoft Office". This betrays a confusion between what constitutes an operating system and programs.

"This document tells you how to create applications that look right, behave properly, and fit into the GNOME user interface as a whole." (GNOME. 2004. p.13)

The LINUX community, having no commercial axe to grind, is, in its documentation, extremely clear about what each layer (operating system, windowing environment, user interface, program, document) is and does.

The confusion exhibited by workshop participants could be exploited in that it would be perfectly possible to develop a platform

that consisted of an operating system and an automated agent interface with no intervening windowing environment or WIMP-GUI.

1.2 SUMMARY OF CHAPTER CONTENTS

LITERATURE AND SOFTWARE REVIEW:

SOFTWARE LEARNING, OBJECT ORIENTED LANGUAGES, COMPUTER INTERFACES AND HOW THESE HAVE COME TOGETHER IN SOFTWARE AUTHORING PACKAGES

This is an extensive examination of literature relating to software learning, object oriented languages, and computer interfaces as far as these topics relate to authoring packages coupled with an examination of those packages from a semi-historical approach.

Attempts at providing a code-free interface to allow on-computer experts to develop software items for their own use are examined. The reasons for the gap between the goal of systems that provide users with a system that “should be usable, without help or instruction” and the packages available are also considered.

The parallel development of operating system user interfaces and individual program user interfaces is examined.

LITERATURE AND SOFTWARE REVIEW:

CONCEPTS OF COMPUTER LITERACY, TYPES OF EDUCATIONAL SOFTWARE, INTERFACE CONSIDERATIONS, AGENTS, VIRTUAL PERSONALITIES AND WIZARDS

The disparity between popular conceptions of 'computer literacy' and what constitutes genuine computer literacy is examined. This is especially relevant in the educational context of this project.

This leads into a discussion of the graphic user interface and direct manipulation. The strengths of the current WIMP-GUIs are looked at, as well as how the current dependence on them may have clouded people's judgement in terms of usability.

The nature of computer agents is explored and refined for the needs of a GUI for educators to develop bespoke computer software for content delivery and reinforcement.

GENERAL METHODOLOGY

Both Workshop design and theoretical parts of the design of the prototype software are explained and justified with reference to Apple Computer's interface design guidelines as well as other sources.

The methodology and section design of the questionnaire is explained and justified. The way the questionnaire attempts to filter out non-computer conversant respondents is explained.

ANALYSIS OF QUESTIONNAIRE RESPONSE

Answers to the questionnaire are examined and considered; illustrated, in some cases, by informative non-standard responses. The evident discrepancy between the few respondents who had programming experience and the many respondents who wished to be empowered to develop materials for their pupils is highlighted.

The chapter ends with a summary of conclusions that may be drawn from questionnaire response and how this may feed into any development of a virtual personality led software development environment.

ANALYSIS OF RESPONSE TO WORKSHOP #1

The response to Workshop #1 is analysed in conjunction with a description of some of the salient aspects of Prototype #1.

The chapter ends with a summary of changes to be made for Prototype #2 as a result of feedback from the workshop.

[Interface changes made from Prototype #1 to later versions should be understood to refer to both Prototypes #2.1 and 2.2 as they are functionally identical from the point of view of end-users.]

ANALYSIS OF RESPONSE TO WORKSHOP #2

The response to Workshop #2 is analysed in conjunction with a description of some aspects of Prototype #2 that differ from Prototype #1. Attention is paid to how the participants react to the changes that were introduced to the prototype as a direct result of their reactions to Prototype #1.

The chapter ends with a summary of changes to be made for a Prototype #3 as a result of feedback from the workshop.

INTERFACE CONSIDERATIONS

A case for abandoning the common WIMP interface in favour of an Object-Point/Click interface is constructed. The main concern is to ease additional cognitive load on would-be software developers in educational institutions. Examples of aspects of current WIMP-GUIs are contrasted with implementation in the prototype of an Object-Point/Click interface. This is supported by the observation that alternative interfaces do already exist.

DESCRIPTION OF NEW SOFTWARE CREATION INTERFACE

A description with a justification of the underlying rationale is given. What has to be taken as axiomatic is explained. These are coupled with an explanation of why the Software Creation Interface was developed using Runtime Revolution.

The chapter then proceeds with an account of the stages an end-

user is taken on one particular trajectory through Prototype #2.

The chapter finishes with an attempt to summarise the author's perceptions with the shortcomings of Prototype #2, and how these might be overcome in future developments of the software creation interface.

CONCLUSION

An attempt is made to draw together all the disparate threads and conclusions from all the preceding chapters into something that resembles a coherent whole.

2 LITERATURE AND SOFTWARE REVIEW: SOFTWARE LEARNING, OBJECT ORIENTED LANGUAGES, COMPUTER INTERFACES AND HOW THESE HAVE COME TOGETHER IN SOFTWARE

AUTHORING PACKAGES

“The computer should be accessible to everyone who chooses to use it.”

(Apple Computer, 1992. p. 14)

While computers were initially designed for a few specialist niches they have become almost universal portals for dissemination of information both over distances (via networks) and locally (e.g. for content delivery in learning institutions). They have also replaced typewriters. They are, already, usurping areas such as creative arts. Computers are being aggressively marketed (‘pushed’) as machines that will serve in all these roles as well as telephones, video-conferencing facilities and general providers of entertainment (television, radio, games, newspapers, etc.).

Just as, at present, a person who is unable to read is effectively both disabled and socially disadvantaged; it will not be long until a person who cannot perform basic data-gathering manipulations on a computer will be in the same position.

Historically alphabetic writing systems have tended towards simplification. This has meant that learning to read and write has become less of a cognitive struggle. There is no reason to see why this process (which has already started with the development of the WIMP-GUI) should not happen with computer interfaces.

2.1 OVERVIEW

Although the original intention had been to treat academic literature relating to software learning, expert systems, agents and virtual personalities in a separate section to authoring software packages and their support literature it became rapidly clear that as:

In the area of authoring software packages it is sometimes difficult to separate purely ‘academic’ literature from authoring software manuals; particularly as many manuals devote considerable space to theoretical considerations.

Most of the theoretical literature is difficult to comprehend *in vacuo* and needs be lined up with concrete examples of attempts to instantiate theoretical concepts.

Consequently this chapter examines the literature and existing authoring software packages in tandem. It also reaches some conclusions based on premises already stated in the Introduction and what emerges on examining existing authoring environments.

The threads followed in this review are:

- Software learning, and expert systems (insofar as they impinge on

the central ideas of this thesis).

- Usability issues, computer interfaces and the concept of ‘computer literacy’.
- The advent and nature of object oriented and object based programming.
- Agents and virtual personalities.

2.2 SOFTWARE LEARNING AND OBJECT ORIENTED LANGUAGES.

The early computers developed at places such as Bletchley Park (to decode wartime signals) were able to do precisely one thing; run any number through a complex algorithm. The idea of ‘software’ as distinct from ‘hardware’ would have been incomprehensible to the developers of those machines; new parameters would be set by performing mechanical adjustments to the machine in question. Even the concept of a machine that could, for instance, perform mechanical operations on itself would have been alien.

With the development of electromagnetic tape the idea of ‘software’ was born, and along with it the concept of a mechanically invariant machine being capable of being programmable by non-mechanical means, as well as the concept of programs being transferrable between machines.

However it was not until the 1960s that the idea that *a program might be modified while it was running* arose; whether by a

programmer or by itself:

“In 1962, Ivan Sutherland's *Sketchpad* became the exemplar to this day for what interactive computing should be like--including having the end-user be able to reshape the tool.”

(Cypher. 1991. [2])

The idea of case-based reasoning arose at much the same time; this embodies the idea that a program may modify its behaviour dependant upon the data it has to work with. Software learning goes beyond this insofar as a program modifies not just its behaviour upon data it is presently receiving but modifies itself so that its future behaviour is affected as well; one might say that the program rewrites itself, or, at least appears to:

“Teaching a lesson involves a series of interactions between a student and teacher. Many researchers have worked on computer tutoring systems, where the main question is how a computer tutor should interact with a human student to facilitate learning. In Programming-by-Example (PBE) systems (and similarly, Learning Apprentice systems) we study the opposite situation, where the computer is the student and human is the teacher.”

(Beach, Minton and Ticea. nd. p.3)

The first attempts at really effective trainable systems developed at the same time as the ‘icons-and-windows’ graphic user interface which has now come to seem trivial:

“Today, we have windowed interfaces everywhere, and even a

number of iconic object-construction kits. We have macro capture systems of every kind, and scripting languages. But we don't have "end-user programming". Nor do we have "programming by example".

(Cypher. 1991. [2])

With the advent of object-oriented programming and the widespread use of the metaphor of containers trainable / learning systems can learn in two possible ways; by modifying the actual software routines themselves, or by changing constants stored in objects (fields) that only affect routines *vis-à-vis* the constants used rather than the routines as such. The advantage of object-oriented languages is that constants may be stored without modifying software routines even when computers are not running. This allows a program to save time by bypassing considerable time while an end-user inputs data because it may be already stored in some type of container (either an object within the program itself or in some external file (e.g. a tab-delimited text file)).

“When a program, such as HyperCard, is said to support the Object Model, it means that the program’s designers have written the innards so that the items a user works with — including data — can be defined as distinct objects. Moreover, each object must have a name of some kind to distinguish it from all other objects.”

(Goodman, 1993. p. 983)

“Good implementations of the Object Model go further, specifying that any piece of data that is subject to manipulation via script can

also be identified within the cosmos.”

(Goodman, 1993. p. 983)

Objects themselves are then viewed as containers which may or may not contain data-objects that may or may not in turn be containers themselves.

Historically there have been two types of object oriented languages:

those that have been developed from scratch as such (more properly most of these should be described as ‘object based’ rather than ‘object oriented’), of which the first was HyperCard.

Those developed by building an object ‘layer’ over an existing non-object oriented language (e.g. Microsoft Visual Basic). One of the disadvantages of this type language is that its ‘previous life’ as a non-object oriented language sometimes interferes with the way objects work and interact with each other.

2.3 OBJECT BASED LANGUAGES

Objects in object based languages differ from objects in object oriented languages insofar as they are not capable of:

- containing methods and messages relative to themselves. This is termed *Encapsulation*.
- inheriting properties, subordinate (child) objects, scripts and methods of other objects. This is termed *Inheritance*.
- control access to their own scripts, methods and other characteristics by setting them as *shared*, *private* or *locked*. This is termed *Access*.
- capable of saving themselves between sessions. This is termed *Persistence*.

Object based languages contain predefined objects and do not allow programmers to develop user-defined types, either from code or by inheritance from predefined types.

As many object based languages mature this distinction is becoming increasingly blurred.

To further confuse the issue some schools of thought would lump together the types of languages described above as containing predefined classes (groups of objects).

The reason for doing this is to differentiate these from what are termed Prototype-based languages. These do not contain classes and function rather differently. An example of this type of language is

SELF where the programmer must define all objects *sui generis* for each program.

Arguably, one of the advantages of using a genuinely object based language is that the programmer needs little or no knowledge of a whole metalanguage to describe and define classes/objects and the types of inheritance relationships that exist between them; the objects being ready-made and there being no inheritance.

2.4 COMPUTER INTERFACES, THE CONTAINER METAPHOR AND EXPERT SYSTEMS

As stated in the introduction the development of software authoring packages has evolved concurrently with graphic user interfaces (GUIs). The theoretical underpinnings of expert systems were 'visible' before the advent of the GUI. However the GUI has effectively obscured this aspect of how computers work. It may be that the GUI has served to obscure the 'innards' of programming without giving end-users enough to apparently do without a knowledge of those innards. In simple terms we can state that the GUI has not gone far enough.

A motor car has its innards hidden from the end-user; allowing the end-user to modify the only parameters he/she needs to modify (speed, backwards/forwards). That is because a car is only expected to perform a limited number of actions.

A computer is, in many ways, like a custom car; end-users now

expect it to be capable of performing a multiplicity of actions in a multiplicity of combinations. However, to ‘tune’ one’s computer one needs to ‘get under the hood’, and that presumes that one is a trained engineer.

It is important to examine how developers have attempted to make software authoring packages infinitely customisable by people who are not ‘motor-mechanics’, and find out how and why these fall short of that idea. This starts by examining the development of the WIMP-GUI and expert systems.

2.4.1 COMPUTER INTERFACES AND THE CONTAINER METAPHOR

While attempts were made to develop icons-and-windows interfaces for such computers as the BBC Micro the first commercially viable version was that developed for the Apple (the BBC version eventually mutated into the GUI used on the Archimedes and other computers running RISC OS).

That the first effective object-oriented programming package (HyperCard) should have been developed for the Apple should come as no surprise as the icons-and-windows GUI is in itself a visual implementation of the container metaphor already well developed on many computer platforms as directories.

The advent of object-oriented programming has allowed the

development of software that while storing learnt information in containers can be reset to its initial (or default) state owing to its core code being unaffected. It also lead to a new definition of what constituted programming software; where once a programmer worked with a computer language he/she now worked with a programming environment that could itself be manipulated by one or more languages.

“In the early days of personal computing, in the mid-1970s . . . recognizing that software was an essential ingredient for their customers’ enjoyment of the hardware, the computer makers almost uniformly included a programming language with the computer. The idea — presumably — was that customers would “roll their own” software. That meant, of course, that learning to program the computer was virtually a prerequisite to making the machine do something other than play the handful of games that began to appear on software shelves.”

(Goodman, 1993. p. 374)

2.4.2 EXPERT SYSTEMS

What follows is an extremely trivial, simplistic explanation of the underpinnings of what constitutes expert systems. While the concentration here is mainly on the manipulation of numbers; highly developed expert systems that have been developed by computer programmers side-by-side with subject-specific experts are used to process knowledge garnered by specialists. The reason these highly complex knowledge gathering systems are not

discussed here is that they really fall outside the scope of what is being attempted here.

Early computer languages (used on mainframe computers initially) such as FORTRAN IV and BASIC were already capable of being used to create expert systems:

“The computer can also repeat instructions or choose which of several instructions it will obey. All programs are built up from a combination of the three program structures: *sequence*, *repetition* and *choice*.”

(Master *Compact*, 1986. p.C30)

The use of decision loops allowed for choice: (rule-based systems vary greatly in terms of syntax so examples have been confined to some written by the author in BBC BASIC in the belief that their structure is reasonably transparent)

```
10 L$= "LESS THAN 4"  
20 M$= "MORE THAN 3"  
30 INPUT A  
40 CLS  
60 IF A<4 PRINT L$ ELSE PRINT M$
```

The use of conditional loops allowed for repetition:

```
10 LET A=1  
20 IF A<5 THEN PRINT "XYZ"  
30 LET A=A+1  
40 GOTO 20
```

That the author(s) of the BBC Master Compact guide wrote about “program structures” show that the idea of modules capable of reuse was already present. It was only a matter of degree from the trivial examples above to develop expert systems involving forward chaining which can be used when programmers have access to initial facts (data) but do not know what any conclusion may be reached.

Rules (consisting of nested IF...THEN statements, or, in the case of BBC Basic choice statements) represent actions to be taken when specified conditions hold on items in the working memory (e.g. if I enter ‘5’ at the prompt in the first example above); also termed “condition-action” rules where a series of patterns in the conditional loops must match the item(s) in the working memory (i.e. data entered). This is also sometimes termed a “recognise-act” cycle. The output of the cycle can be used either to modify the contents of the working memory (and hence affect the further working of the cycle):

```
10 PRINT "INPUT START VALUE"  
20 INPUT A  
30 IF A>1000 THEN GOTO 70  
40 PRINT A  
50 LET A=A+A  
60 GOTO 30  
70 PRINT "-----"
```

The value in the working memory, “A” is updated (by doubling) for each cycle of the conditional loop until the escape-condition (the value of “A” exceeding 1000) is satisfied.

It is also, similarly, possible to perform backward chaining from some desired goal state to find likely data to produce that goal state.

If a software program contains code instructing it to save itself after some of its own constants have been changed then the program may be said to be capable of learning (in however a primitive fashion).

The step an object-oriented environment makes beyond this is the ability to store data in containers rather than modifying itself; and then “looking” in those containers and retrieving that data and acting on them at a later date. This allows for multiple sets of learnt data to be stored and acted upon.

Programming By Example (PBE) uses forward-chaining principles to “train” software to predict future user habits:

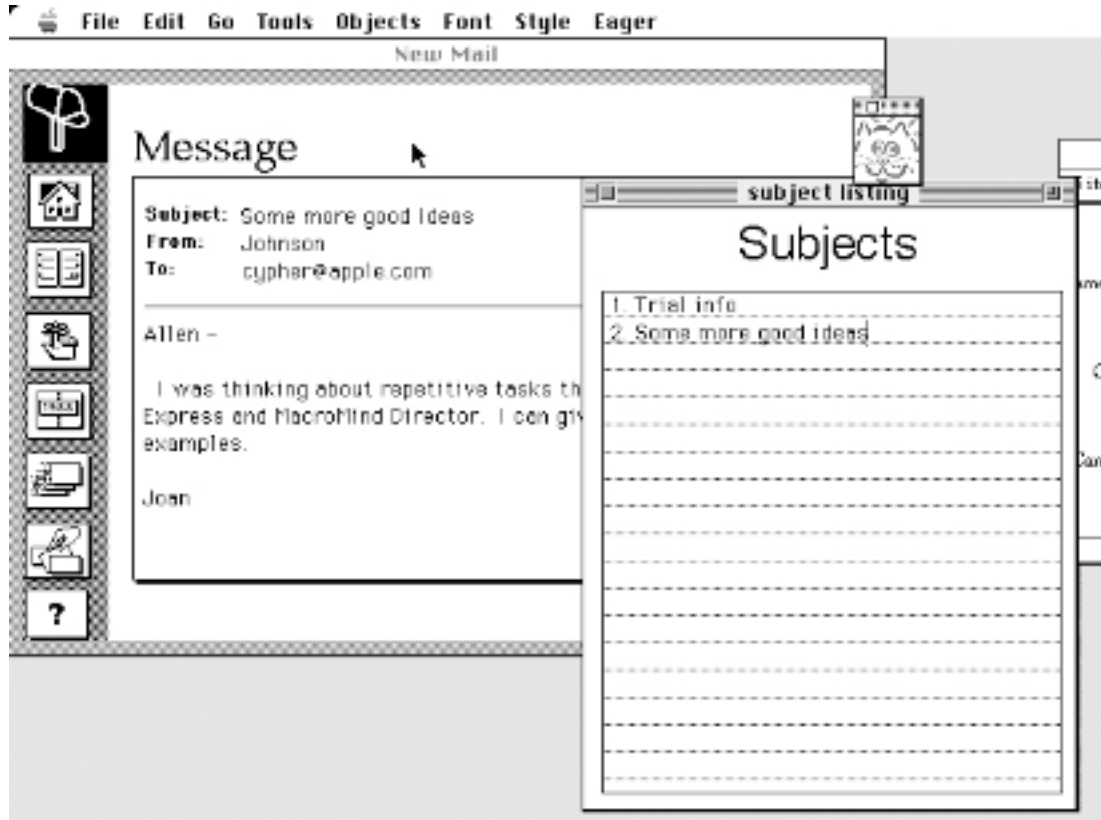
“Programming by example [or "programming by demonstration"*] is a technique for teaching the computer new behavior by demonstrating actions on concrete examples. The system records user actions and generalizes a program that can be used in new examples.”

(Lieberman. 2004)

“Programming by Demonstration is a technique that can potentially solve this problem. Since by definition most of the steps in a repetitive task are the same as in the previous repetition, it should be possible for a computer program to automate the task by

recording the steps and replaying them.”

(Cypher. 1991. [2])



Eager at work.

Note the presence
of a 'virtual
personality'



PBE software allows the end-user to specify loops by demonstration. PBE software is not, in itself, an end-user program as such, more a programming environment that will, itself, write a program to cater to an end-user's needs depending on what the end-user has "shown" the PBE software. At its crudest this is rather like a housekeeper, who, observing that his/her employer has asked for bacon and eggs for breakfast for the last 4 days concludes that

there is a high probability of him/her requesting bacon and eggs on day 5.

2.4.3 COMPUTER INTERFACES AND EXPERT SYSTEMS

Before the advent of the GUI PBE/D systems could only proceed by continually asking the end-user for data (as typing at a terminal was the only way of interacting with the computer); but with the advent of the GUI and pointing devices (i.e. a direct-manipulation interface) PBE/D systems could learn by tracking an end-user's mouse movements and clicks, even to the extent that the user might be completely unaware that his/her computer work was being "observed" at all. This is the foundation of subsequent predictive help systems such as that built into Microsoft's "Office" packages.

"Direct manipulation allows people to feel that they are directly controlling the objects represented by the computer. According to the principle of direct manipulation, an object on the screen remains visible while a user performs physical actions on the object. When the user performs operations on the object, the impact of those operations on the object is immediately visible."

(Apple Computer, 1992. p.5)

Expert systems are rule-based systems that produce decision trees

based on knowledge garnered from experts in highly specialised fields. However:

“Knowledge acquisition is difficult because the computer scientists who build expert systems (referred to as *knowledge engineers*) often lack the background needed to pose optimal questions, while experts in the application area often have difficulty appreciating how their knowledge is to be captured in the computer. It has long been suggested that if domain experts could somehow encode their knowledge directly without relying on programmers as intermediaries, the development of expert systems might be greatly expedited.”

(Hendler, J. ed, 1988. p.16)

While bespoke computer software packages for content delivery and reinforcement are not, in themselves, expert systems they often suffer from the problems mentioned above: they are usually an unhappy compromise created by either a talented computer programmer who has little or no domain knowledge, or a domain expert who is “fumbling around in the dark” in terms of programming expertise. What might solve this problem is an expert system that works by PBE/D principles to allow domain experts to produce bespoke computer software packages for content delivery and reinforcement with no specialist computer knowledge.

2.5 ATTEMPTS AT PROVIDING END-USERS WITH A

‘PROGRAMMING-FREE’ INTERFACE: THE HYPERCARD MODEL

There is a whole history of computer programming environments that have been developed over the last 20 years that have claimed to provide the “holy grail” of a software package that “should be usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.” (Constantine and Lockwood 1999, p.47).

What follows is an examination of several, with particular reference to those that claim to be capable of producing software without any specialist programming knowledge.

For reasons which should be self-evident software packages such as Microsoft PowerPoint that are only used to make slideshows for presentations are not considered here.

Examples are confined to object-oriented software development environments.

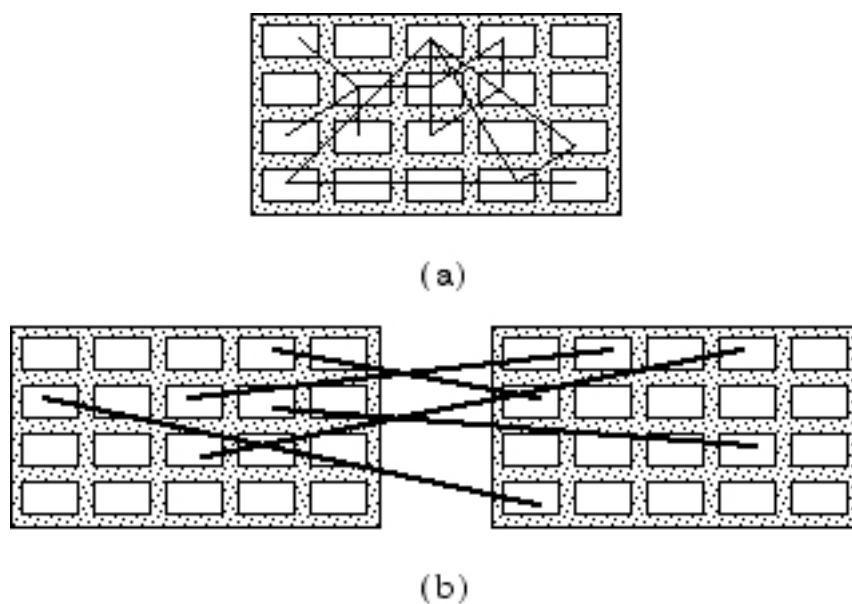
HYPERCARD

The HyperCard model is based on the image/metaphor of cards in a shoe-box filing system that are hyperlinked together (it was Bill Atkinson, the inventor of HyperCard, who coined the term “HyperText”) in stacks. It has proven both a powerful and long-lasting metaphor:

“Far more powerful than a simple table of contents is the creation

between pieces of information — links that may even stretch from stack to stack . . . you can think of a link as one of those colored strings on a bulletin board full of cards that ignores the straight (linear) organization of facts. Links may exist between diverse cards within a stack, or between cards in different stacks.”

(Goodman, 1993. p. 75)



recreation of diagram at (Goodman, 1993. p. 75)

Interestingly enough, HyperCard was not initially described as a Rapid Application Development tool (RAD):

“HyperCard is a new kind of software — a unique information environment for Macintosh computers.”

(Apple Computer, 1990. p.1)

However uses are listed which would show to the prescient computer cognoscenti the way HyperCard would go:

“People have created Hypercard stacks for a variety of purposes, such as

- “• self-paced demonstrations and training packages for business
- “• electronic, interactive directories at conventions and trade shows
- “• search and retrieval software for information on compact discs and videodiscs
- “• animated presentations with sound for use in education, business, and retail sales
- “• database management, with graphics and sound storage capabilities as well as text
- “• just for fun!”

(Apple Computer, 1990. p.28)

Right from the outset Bill Atkinson, the developer of HyperCard, was at pains to differentiate between what he termed the ‘application’ and the ‘language’ (what are now termed the ‘Rapid Application Development Suite’ (RAD) and the language): HyperCard and Hypertalk respectively.

However, that is not really the case:

“. . . it is difficult to discuss HyperTalk separately from HyperCard. They are intricately intertwined.”

(Goodman, 1993. p. 377)

What HyperCard and HyperTalk are are different facets of the same thing:

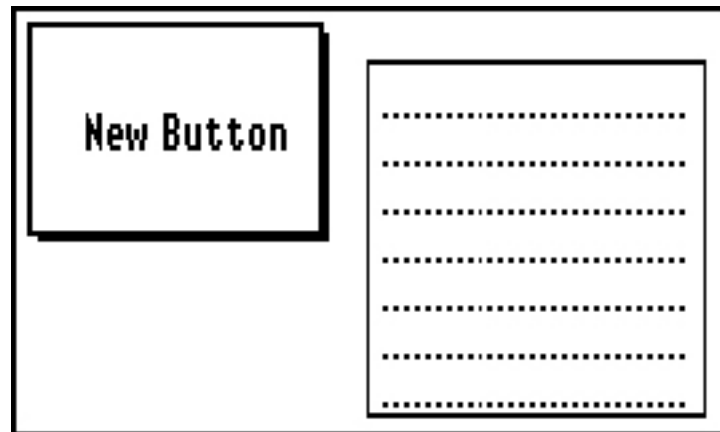
HyperCard being the visible interface on the Macintosh GUI and the visual representation of the predefined objects.

HyperTalk being the aspect of the underlying language that is used to define how the predefined objects would behave. That this is a subset of the complete code within the Hypercard package is that the objects are predefined and the end-user is able to drag them around the GUI without worrying about defining the objects' screen coordinates:

“It preprocesses all the gobbledygook out of sight.”

(Goodman, 1993. p. 377)

A HyperCard document containing objects.



Unfortunately HyperCard does not process “all of the gobbledygook out of sight.” To get objects to act (rather than just sit there, looking pretty) HyperTalk must be typed into the ‘script’ of each object.

Admittedly, to be fair it is claimed that:

“The language was painstakingly designed to be easy to learn and use. The goal was to put powerful authoring tools into the non-technical hands of those who need custom applications that no commercial developer would dream of. The HyperCard environment puts programming into the hands of specialist in areas other than computers”

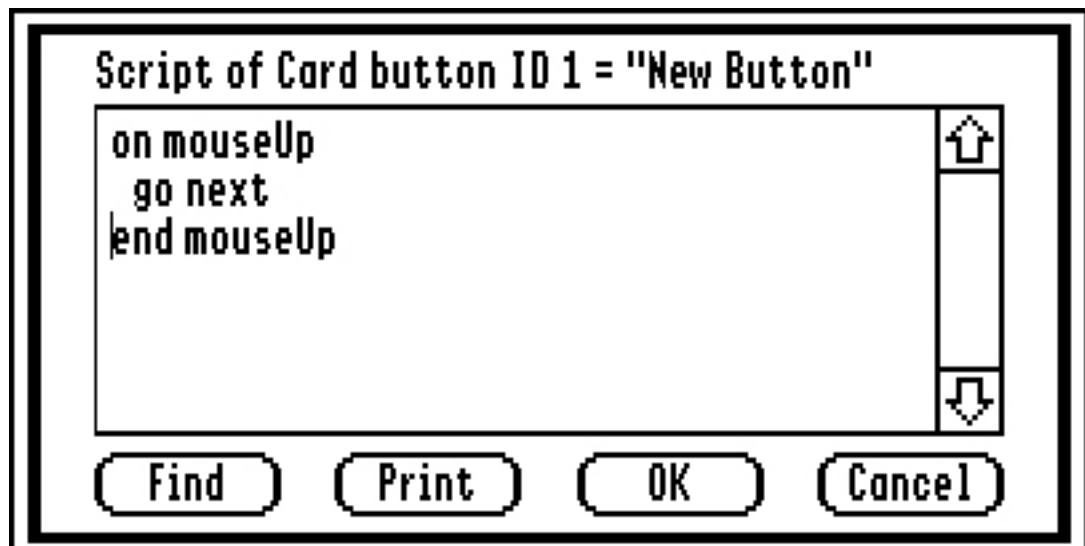
(Goodman, 1993. p. 377)

For instance if an end-user wishes to use a button object to navigate from one user screen to the next he/she must type:

```
on mouseUp  
    go next  
end mouseUp
```

into the script editor of the button object.

The HyperCard script editor (Apple II, HyperCard IIGS).



While the example given is essentially rather trivial it demonstrates that, despite Goodman's claims; there was a programming language to be learnt (HyperTalk), as well as all the strange conventions used in computer languages.

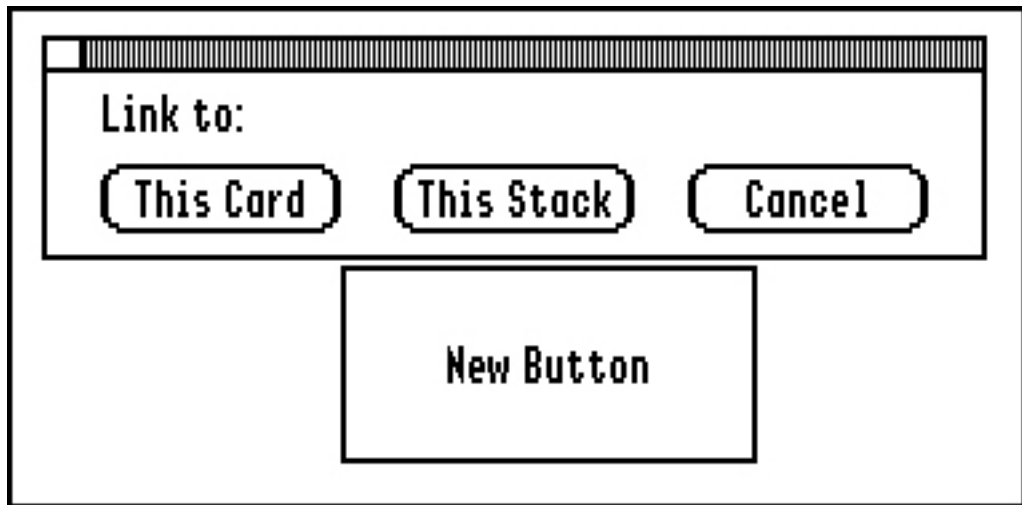
Programming computers does not simply consist of learning to 'speak' a reduced-vocabulary language (rather like Ogden and Richard's "Basic English"); like any spoken language a computer language has all sorts of cultural assumptions and associations embedded in it:

"candygrammar: n.

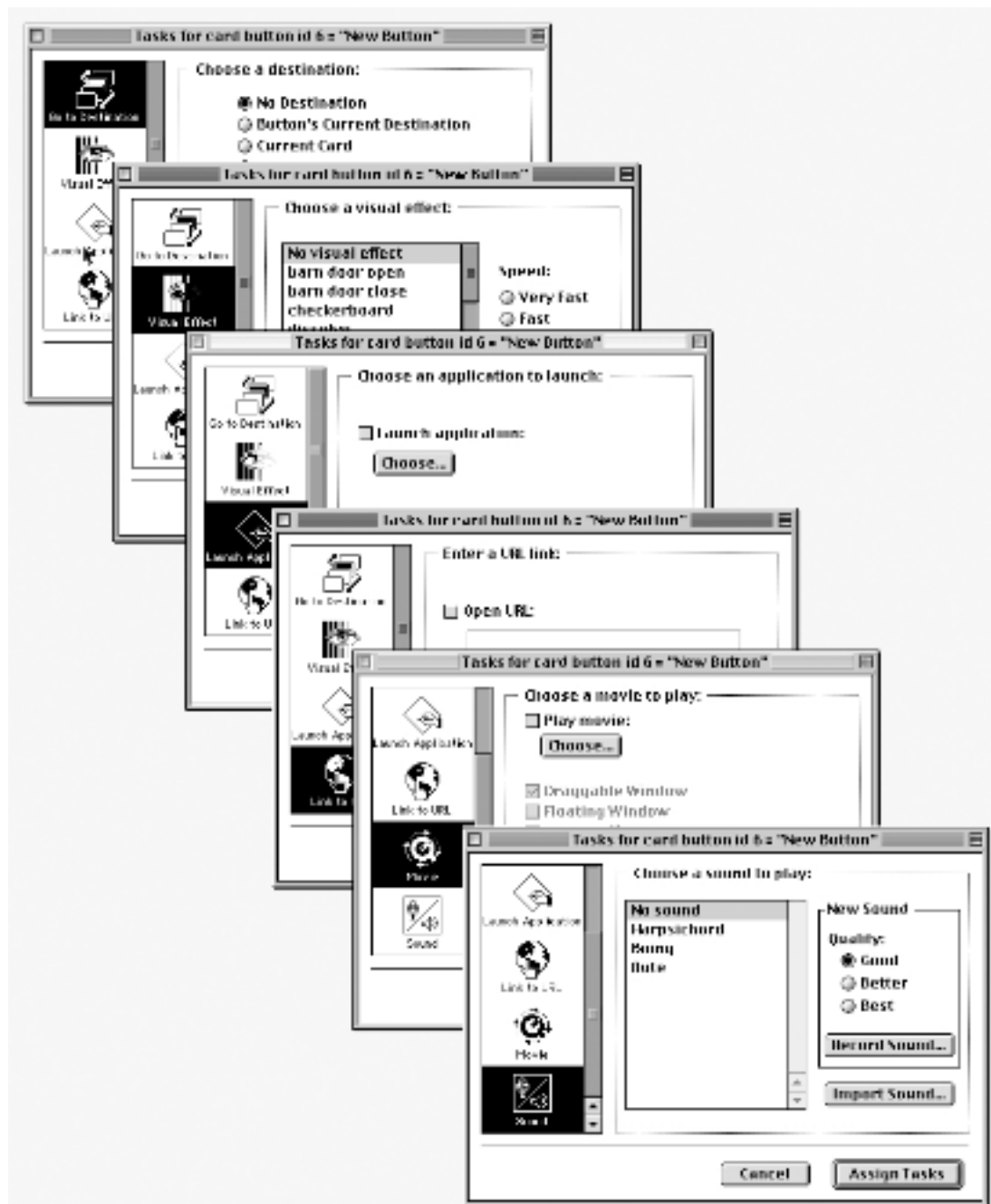
A programming-language grammar that is mostly *syntactic sugar*; the term is also a play on 'candygram'. *COBOL*, Apple's Hypertalk language, and a lot of the so-called '4GL' database languages share this property. The usual intent of such designs is that they be as English-like as possible, on the theory that they will then be easier for unskilled people to program. This intention comes to grief on the reality that syntax isn't what makes programming hard; it's the mental effort and organization required to specify an algorithm precisely that costs. Thus the invariable result is that 'candygrammar' languages are just as difficult to program in as terser ones . . ."

(Candygrammar. 2004)

To an extent HyperCard tried to get around this with the use of pre-programmed "tasks" for objects":



(Button Tasks in HyperCardIIGS for the Apple II)



(Object tasks in HyperCard 2.4.1 for the Macintosh)

But these were very limited and restricted the end-user to making either data-accessing programs or slideshows.

To do anything further than the rather basic tasks on offer within HyperCard the Script-Editor had to be invoked, and, despite claims to be programmable in natural language, HyperTalk features all the aspects of computer languages that so signally differentiate them from spoken language. HyperTalk statements generally consist of a varying number of commands and loops (subroutines) which must be terminated by control constructs (end if, end while, end do, etc.).

HyperCard was used by an extremely large number of educators (and others) to develop some extremely sophisticated software. Those developers had, however, to devote a considerable amount of time to learning both HyperTalk and the underlying principles of computer programming. Many people who had invested considerable time and effort into learning HyperCard therefore felt betrayed when Apple Computers stopped development of Hypercard in 1996 and formally discontinued it in 2004. Not least that, despite all claims to the contrary, efficient and effective programs could not be made in HyperCard without a knowledge of programming. In fact it is probably fair to say that Hypercard was many people's route into using computers as more than sophisticated typewriters.

The vast majority of educators who invested nearly 20 years of their lives into learning HyperCard-HyperTalk were not paid beyond their jobs as educators. Therefore very few were to prepared to invest the same amount of effort in learning how to control a new Rapid Application Development suite (RAD).

Many members of Apple's HyperCard development team left and

started rival (and incidentally much more commercial) RADs to HyperCard using what we had best characterise as ‘dialects’ of HyperTalk giving rise to the generic term ‘xTalk’ to refer to them as a class of programming language. Very few survived for long, but two that did were SuperCard and MetaCard. These came to maturity by very different roots as SuperCard attempted to steal HyperCard’s position as a RAD for the Macintosh platform while MetaCard was developed primarily as a MicroSoft Windows and Unix port of Hypercard. Many groups of developers such as the WildFire group (one of whose members, Dan Gelder, went onto develop his own, short-lived, HyperCard clone: ‘Serf’) and Harry Alloul (OpenCard) developed enhanced interfaces for HyperCard.

There have also been attempts to develop Hypercard clones for other platforms under the Open Source initiative. HyperSense was a version for OpenStep, NextStep and Rhapsody systems, but seems to have died with those operating systems despite the projected porting to other platforms.

SuperCard is, to all intents and purposes, a clone of HyperCard. The only real differences are that it has a much larger class of predefined objects (user-defined objects have not been allowed as of 2004) and a code-editor (SuperEdit) as well as a Runtime editor (SuperCard). If anything there is an increased emphasis on coding over that of HyperCard.

MetaCard was (effectively discontinued in 2003) a clone of HyperCard; but being developed for Microsoft Windows and Unix it

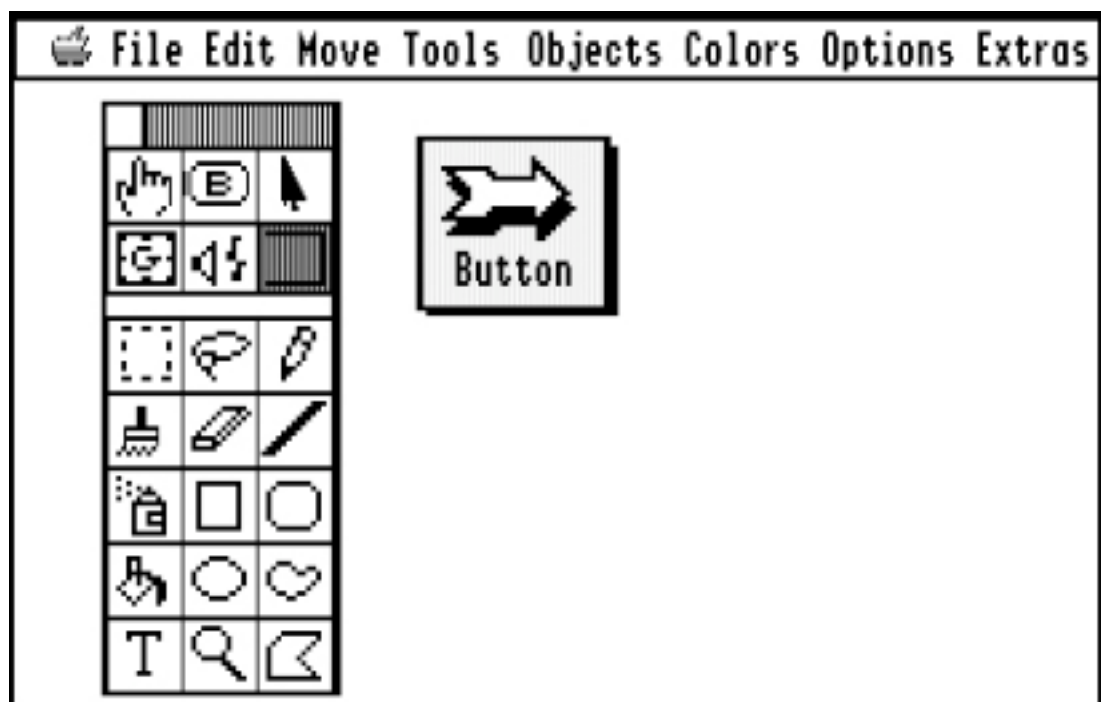
was able to ‘recolonise’ the Macintosh platform at the time when Apple allowed Hypercard to slide towards its death with the advantage that end-users could produce program packages that could run on the Macintosh, Microsoft Windows, Unix and Linux platforms.

Runtime Revolution is a RAD based on the MetaCard engine (which the company now owns the rights to) with a slightly different interface.

Runtime Revolution have recently released a cut-down version called “Dreamcard”. However, apart from the toolbar allowing users to drag-and-drop objects from it and a ‘candy coloured’ (i.e. modelled on the Microsoft Windows XP GUI) interface, it still relies heavily on the user’s computer knowledge; and beyond the simplest of activities requires a knowledge of Transcript (the Runtime Revolution dialect of xTalk).

MetaCard and Runtime Revolution allowed all referenced ‘stacks’ to be contained in one file with the idea of ‘substacks’; this got rid of the problem of a central program’s dependency on files which could be misplaced (‘orphaned’) in incorrect directories where the central program could not reference them.

HyperStudio is a RAD developed to resemble HyperCard almost exactly, except that it uses a dialect of Logo (“HyperLogo”) rather than an xTalk dialect.



HyperStudio 3.1 (Apple IIGS)

HyperLogo is far less intuitive than xTalk which makes HyperStudio's claim to be significantly easier to use than Hypercard rather difficult to understand:

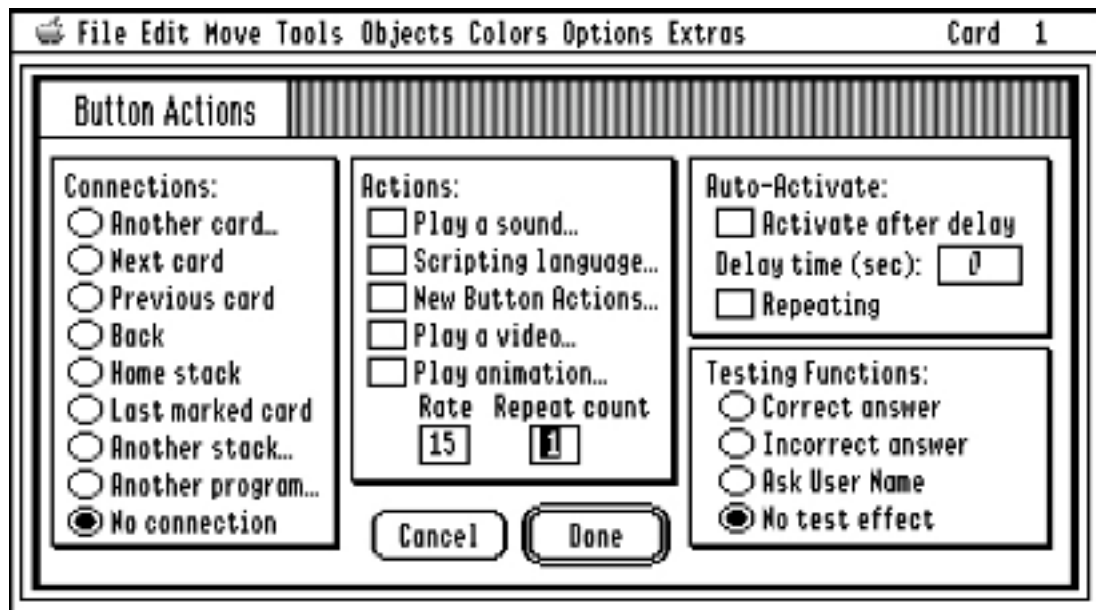
```
TO Initialize
; Set up the list of problems.

local "i
local "j

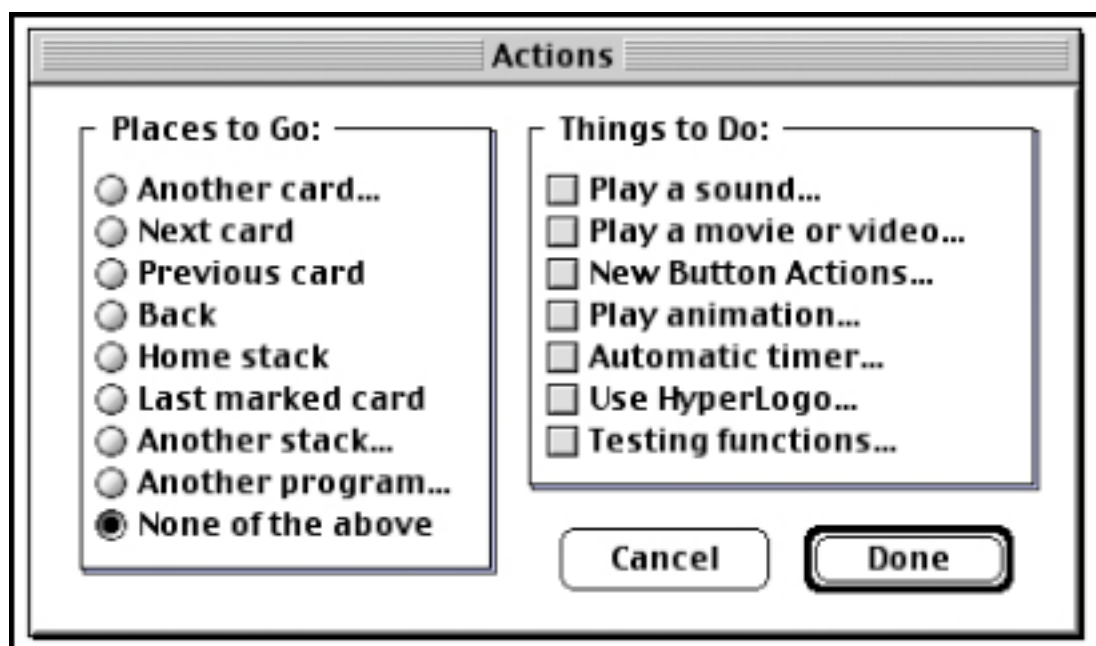
make "probs []
make "i 1
repeat 10 [
  make "j 1
  repeat 10 [
    make "probs fput (list :i :j) :probs
    make "j :j + 1
  ]
  make "i :i + 1
]
END
```

sample HyperLogo script

It did provide the end-user with significantly more predefined tasks / actions than Hypercard:



HyperStudio 3.1GS Button Tasks

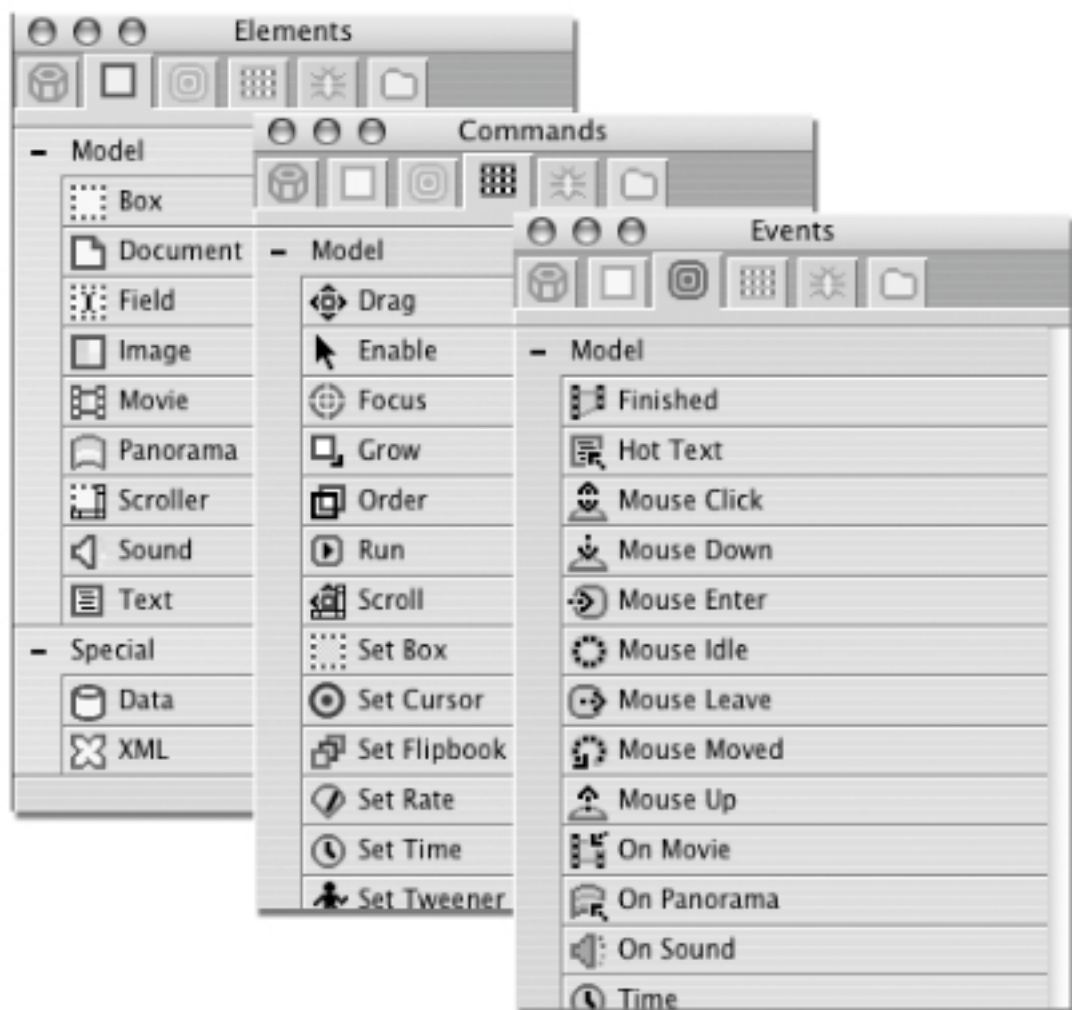


HyperStudio 3.2 (1999) Button Tasks

HyperNext is a new RAD based on the HyperCard model that appeared in 2003 using both some form of Apple Computer's own AppleScript and REALBasic as well as a native HyperNext language.

However HyperNext is at such a rudimentary stage of development (and the developer seems unable to write his/her documentation in anything approaching native-like English) that a coherent picture of the language is impossible to grasp.

Tribeworks' iShell differs slightly from the Hypercard model; offering a large number of predefined tasks for objects (which iShell terms 'Elements'):



It uses a non-xTalk language called 'Key':

```

object oHorizontal_Funny_ScrollBar
is
    cScrollBar
with
    release Editor:
        Name is "Funny";
        Identifier is
"oHorizontal_Funny_ScrollBar";
    end;
    URL is "./FHCSB.PIC";
    Mode is oTransparentMode;
    Red is 255; Green is 255; Blue is 255;
    Direction is Horizontal;
    HiliteOffset is 20;
    FirstArrowOffset is 0;
    FirstArrowSize is 20;
    BarOffset is 20;
    BarSize is 580;
    LastArrowOffset is 600;
    LastArrowSize is 20;
    ThumbOffset is 620;
    ThumbSize is 20;
end;

```

sample Key script

iBuild presents a reduced object set and only claims to be for developing slideshows; it uses Apple Computers' AppleScript.

What is clear is that none of the examples given above can get away from the need for the end-user to learn one computer language or another.

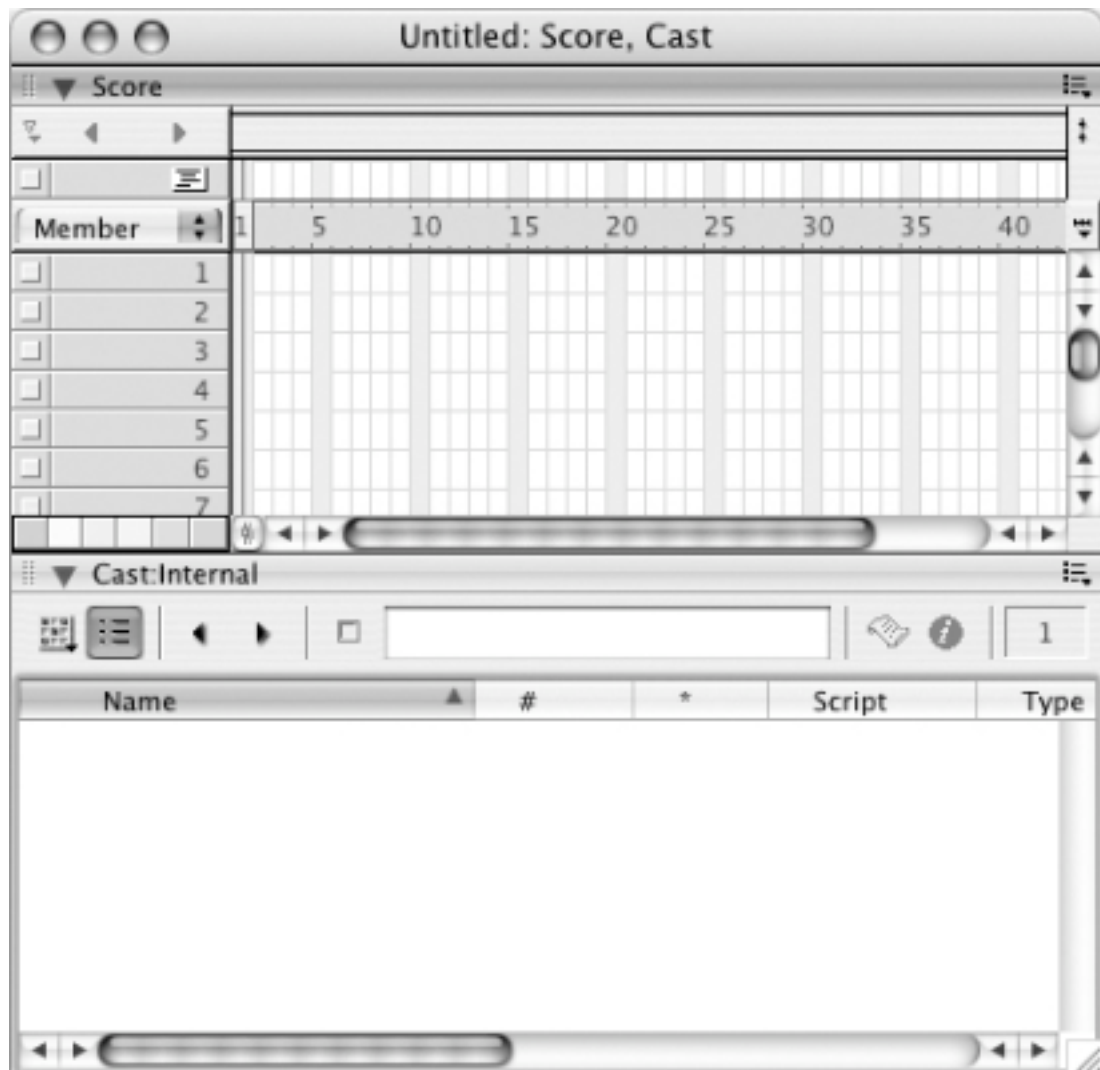
2.6 ATTEMPTS AT PROVIDING END-USERS WITH A 'PROGRAMMING-FREE' INTERFACE: OTHER ROUTES

Macromedia Director uses a different metaphor to the HyperCard model:

“The Director user interface is designed around a movie metaphor. Each project you create can be thought of as a movie, with a cast of characters, a score, a stage where the action takes place, and a director (you, the author). Each media element that appears in your movie (sound, video, images, text, buttons, and so on) can be thought of as a member of the movie's cast. In Director, the Cast window is where you view the list of media elements that appear in your movie.”

(Macromedia Director MX 2004 online help system, 2004.

“Understanding the Director metaphor”)



Macromedia Director MX 2004 Cast window

In addition to this Director uses a language ('Lingo'):

“This reference provides conceptual and how-to information about scripting in Macromedia Director MX 2004, and also provides reference descriptions and \ examples for the scripting application programming interfaces (APIs) that you use to write scripts.

“The scripting APIs are the means by which you access the

functionality of Director through script to add interactivity to a movie. By using these APIs, you can create interactive functionality that is identical to that provided by the prewritten behaviors that are shipped with Director, in addition to functionality that is more powerful and more varied than that provided by the prewritten behaviors.”

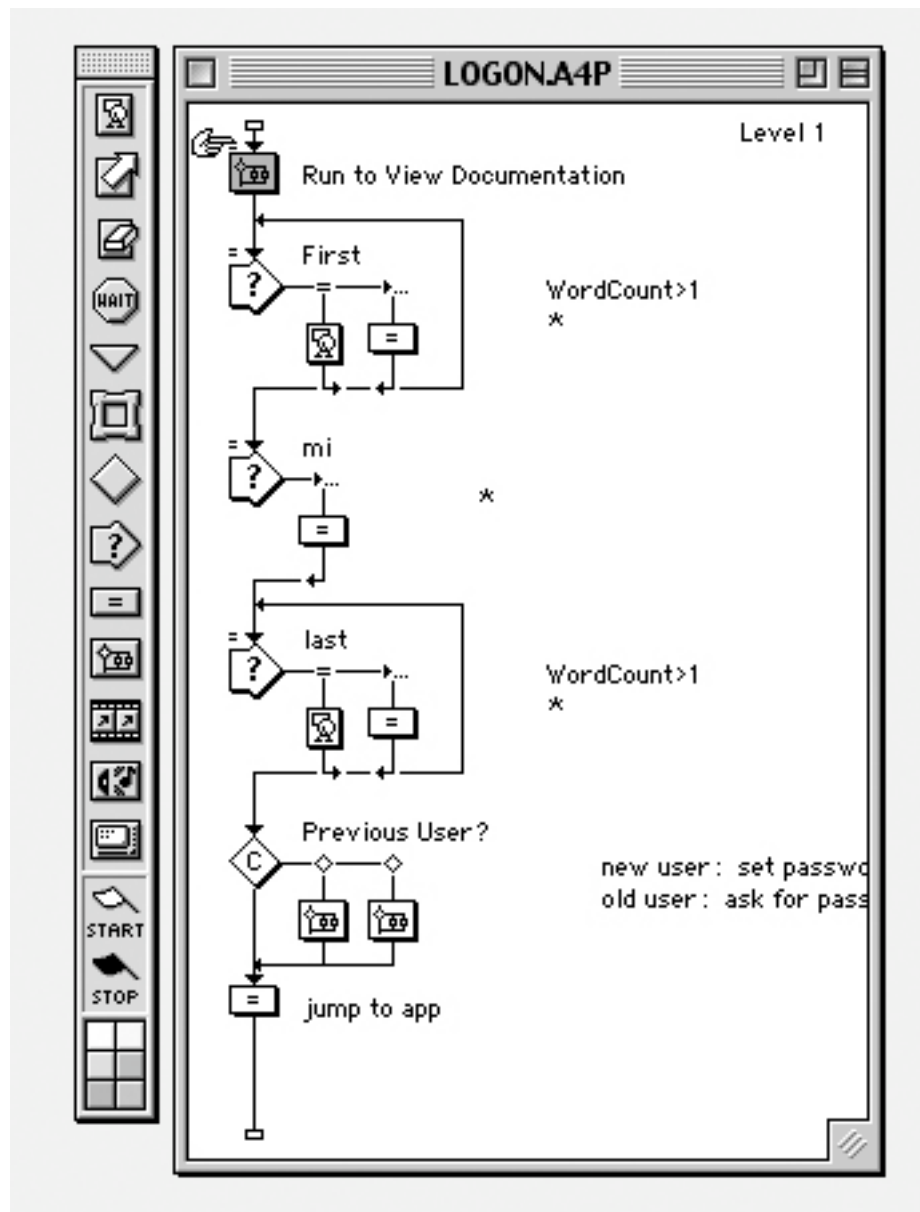
(Macromedia Director MX 2004 online help system, 2004.

“Director Scripting reference > Introduction”)

This is not “usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.”

(Constantine and Lockwood 1999, p.47).

Macromedia took a different route with their package ‘Authorware’, using a “flowline” metaphor:



End-users are expected to develop their flowline by dragging object icons from the ‘icon palette’ to the flowline.

Theoretically it is possible to make complete programs without any

scripting, but the effort involved in getting to grips with the flowline, navigation hyperlinks, 'maps', 'frameworks', 'glyphs' (all parts of the Authorware metalanguage) could, quite easily lead to cognitive overload.

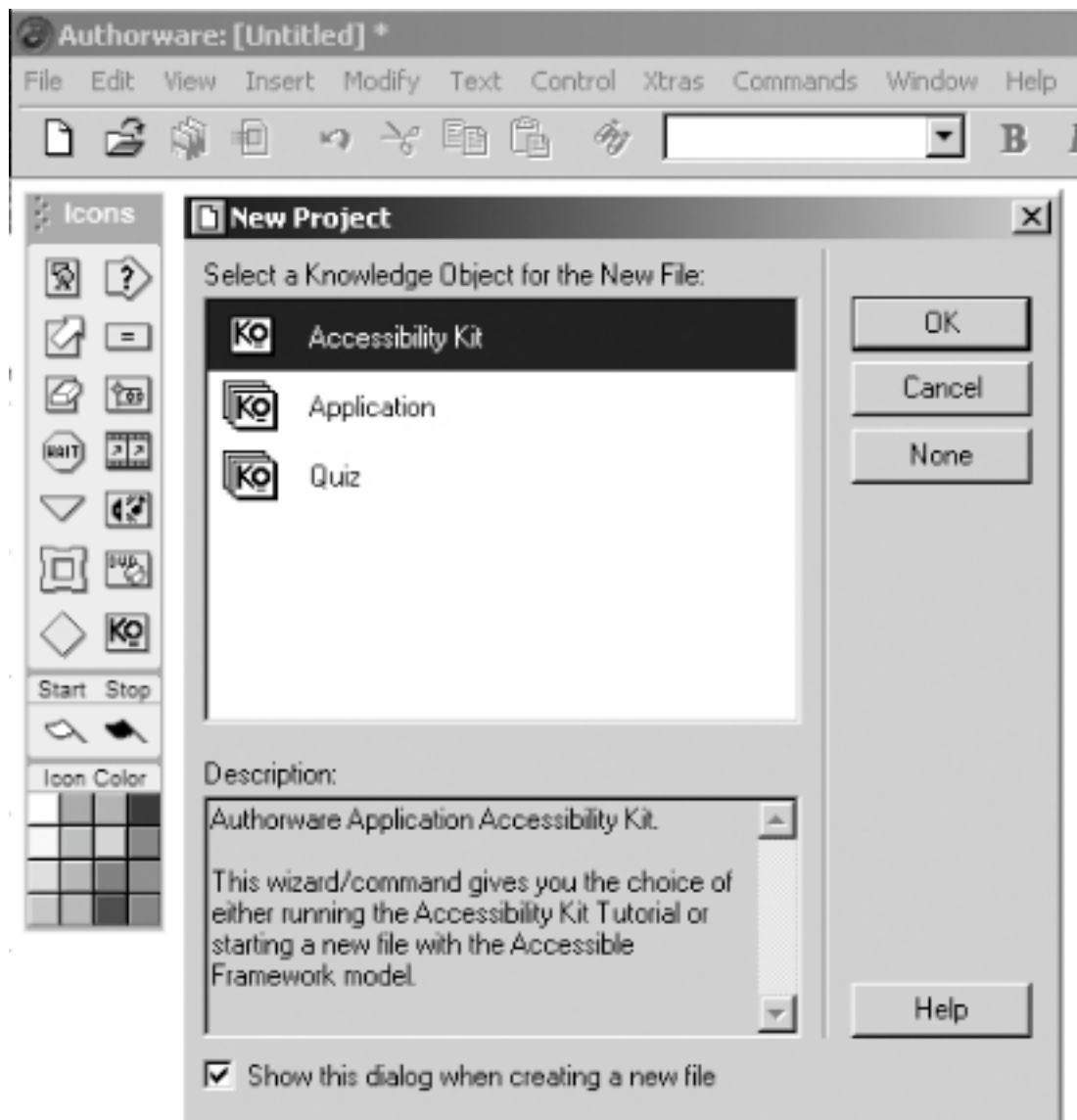
What is perhaps a telling fact is that Ruth Colvin Clark in her introductory chapter to "Taking The Plunge: Macromedia Authorware version 3" (Clark, 1995. pp. 3-28) uses examples of HyperCard stacks to demonstrate her point rather than Authorware programs!

Authorware has its in-house language called "Authorware Script Language", and in version 7 it allowed the use of JavaScript 1.5. This would seem to indicate that Authorware is moving away from the 'drag-and-drop' idea towards a more programmer-friendly environment.

Toolbook (a program that appears to change ownership annually) uses a series of templates and drag-and-drop objects with names like "Action Trigger" and "Action Timer" controlled by a language ("OpenScript"); its reference material makes no mention of 'metaphors', but it lies somewhere near to the Hypercard model.

The authoring packages discussed under the heading "Other Routes" are even less approachable by non-programmers. An attempt has been made in recent versions of both Toolbook and Authorware to lessen the burden with the introduction of wizards to allow various predefined templates to be put in place. These do not, however, in

any way ‘finish the job’:



Macromedia Authorware 7 Wizard

This is obscurantist in its use of package-specific terminology ('Bullshit baffles brains'); e.g. *Knowledge Object* (c.f. image above). Authorware also stores media required for software developed with it externally to the finished product. Therefore it extremely easy for associated media files to get mislaid, 'orphaned' by the user.

“Include all required resources inside your application bundle. Your application bundle should always have everything it needs in order to run.”

(Apple Computer, 2004. [1]. p.61)

It should be perfectly feasible for a programmer to use any of these RAD packages as a basis to develop a different interface to replace the both current GUIs and current authoring packages dependence on a knowledge of programming in proprietary languages.

This different interface must conform to the following rules:

- At no time should the end-user have access to or be aware of the “innards” of the RAD (i.e. the programming language).
- At no stage during use of the software should the end-user have to enter any programming code.
- At no time during the use of the software should the end-user have access to any features of the GUI of the operating system in which the software is running. For example, end-users running the software in Microsoft Windows should have no access to either the desktop or directories.
- Computer skills needed by the end-user to manipulate the interface should be reduced to a bare minimum (e.g. mouse-clicks and text typing only).

It does not seem too far-fetched to envisage (within the next 10 years) a voice-operated interface, doing away with the need for either a keyboard or a pointing device.

2.7 WHAT HAS BEEN LEARNT

The metaphor of a stack of hyperlinked cards is powerful and seems to have stood the test of time.

While promising that “the computer should be accessible to everyone who chooses to use it.” And that software packages would “be usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.” these high ideals have not really come to fruition.

Indeed, if one looks at the development of software authoring packages over the 1990-2004 period it is hard not to believe that those ideals have been lost sight of.

Certainly developers of currently available RAD packages seem to be concentrating on making them as feature-rich as possible, often at the expense of usability except by people who are dedicated computer programmers.

2.8 CONCLUSION

The development of the WIMP-GUI as a *de facto* standard, and the concurrent development of software authoring packages while

initially intended to make computer systems readily usable by non-computer experts has resulted in a stagnation of the goal of developing packages that are genuinely “usable, without help or instruction, by a user who has knowledge and experience in the application domain but no prior experience with the system.”
(Constantine and Lockwood 1999, p.47)

In fact it would be probably safe to say that many software authoring packages that started their lives with this goal as central to their development plans have tended through each development cycle / version to rely on an increased familiarity of the end-user with computer-specific concepts and programming language.

3 LITERATURE AND SOFTWARE REVIEW:

CONCEPTS OF COMPUTER LITERACY, TYPES OF EDUCATIONAL SOFTWARE, INTERFACE CONSIDERATIONS, AGENTS, VIRTUAL PERSONALITIES AND WIZARDS

3.1 CONCEPTS OF COMPUTER LITERACY, TYPES OF EDUCATIONAL SOFTWARE AND INTERFACE CONSIDERATIONS

Roger Rist and Sue Hewer list the following types of educational

software:

“Drill and practice

Tutorials

Information retrieval systems

Simulations

Microworlds

Cognitive tools for learning

Productivity tools

Communication tools”

(Stoner. Ed. 1996, p.3)

They then continue by stating:

“A further category refers more to learning about computers rather than learning with computers.”

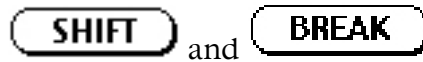
(Stoner. Ed. 1996, p.3)

This is an all-too important distinction that is often lost sight of for one signal reason: that, at present, it is extremely difficult for both pupils / students and educators (as well as experts in non-computer related subject areas) to use computers for and of the topics listed above without, of necessity, learning a considerable amount about computers.

Paradoxically this has largely come about with the advent of the Graphic User Interface.

With earlier personal / educational computers such as the Acorn

BBC Micro end-users would insert the floppy-disk containing the educational package into the floppy drive(s) and (in the case of the BBC) press the



keys simultaneously for the interface of the software package to load directly. The end-user did not need to know anything about the Acorn MOS operating system or BBC BASIC unless he/she chose to.

Needless to say this gave rise to a vast heterogeneity of interfaces.

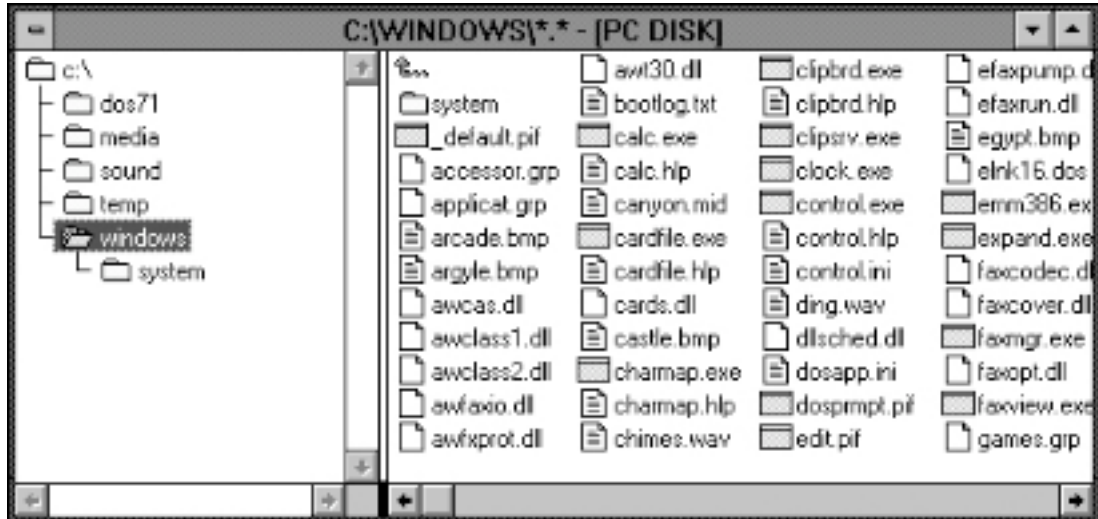
This is one of the reasons that companies such as Apple produced books outlining interface standards for programmers to use when developing software to function with their operating systems.

This has led, to a large extent, to an increasing homogeneity of both operating system GUIs and the programs that run inside them: often to the point that end-users are unable to distinguish programs from operating system interface.

The Graphic User Interface requires direct manipulation in that the user has to click on 'artefacts' within the GUI, and, possibly, 'tunnel' through directories to find icons representing software packages and documents.

An attempt to facilitate access to directories is the 'explorer' or 'file manager' model. This first came into widespread use with versions of Microsoft Windows before Microsoft cloned the Macintosh GUI

with Windows 95. It still presupposes end-users know where in the directory structure to look for what they want.

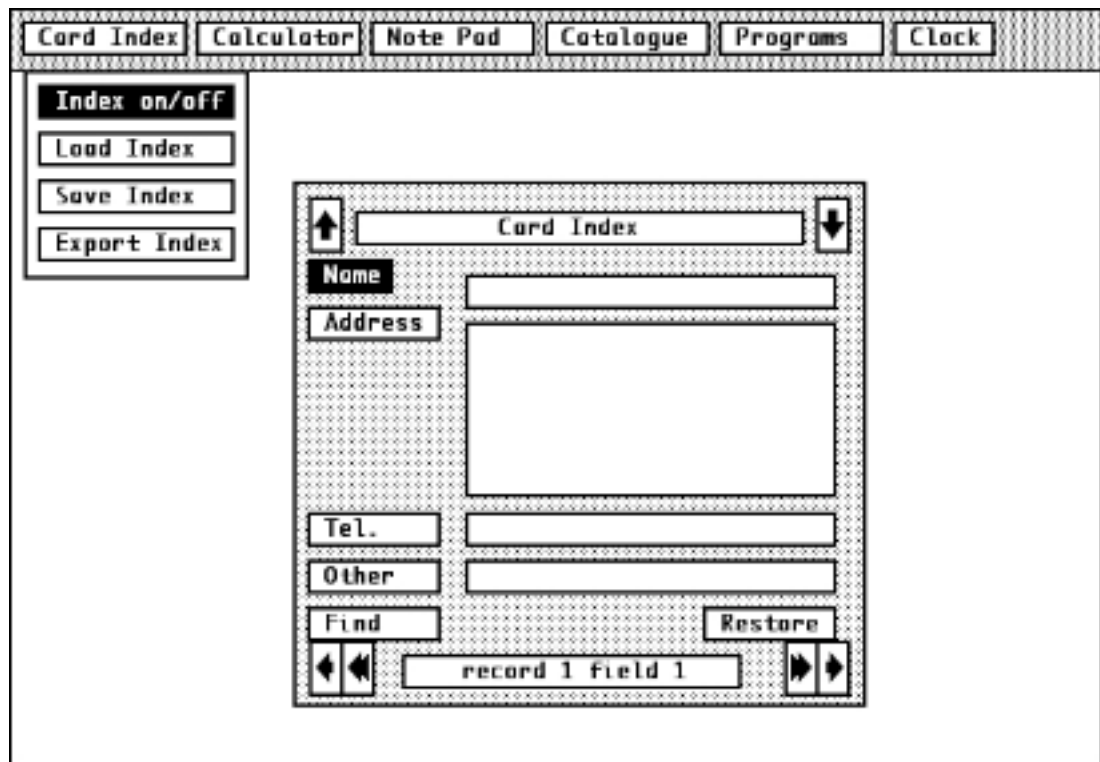


The Microsoft Windows 3.11 File Manager

The Acorn BBC Master Compact introduced a floppy-disk based GUI (the BBC did not have a hard-disk fitted as standard):

“In recent years increasing importance has been placed on making computers more consistent and therefore simpler to use. . . . Your Master Series computer is supplied with a set of programs which provide a simulated ‘desktop environment’ . . .”

(Master *Compact*, 1986. p.B1)



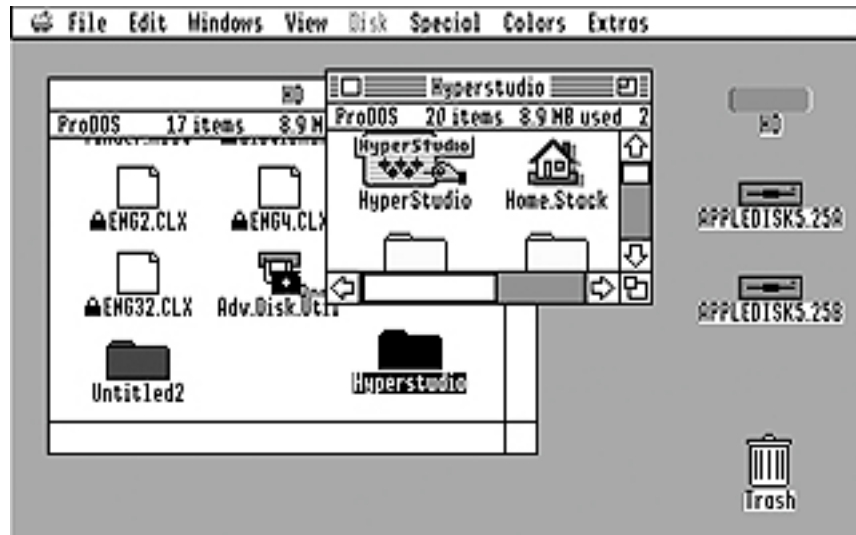
recreation of image at (Master *Compact*, 1986. p.B4)

It did simplify certain things considerably insofar as programs could be loaded by pointing and using the 'fire' button on a joystick or mouse at menu item along the top of the desktop and then, similarly, selecting from the drop-down menus.

Where this started to go wrong (it never got to that stage with the BBC) was when the number of programs exceeded the space for menu items and had to be replaced with icons, which, again due to spatial considerations, had to be hidden in nested windows.

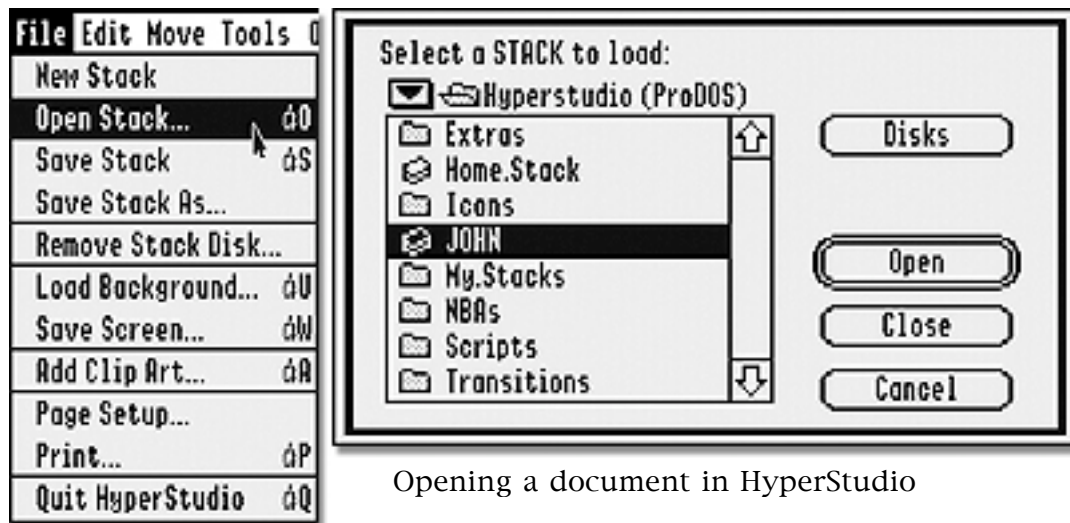
For instance; in an Apple IIgs if one wants to start using HyperStudio one has to double-click on the hard-disk icon, then again in the Hyperstudio folder, and then on the HyperStudio icon.

One also has to know to avoid clicking on any of the other mysterious and unexplained icons in either of the windows:



Locating HyperStudio 3.1 in an Apple IIgs
computer running Apple OS 6.0.1

Then to open the last HyperStudio document one was working on one has to navigate through directories again via the HyperStudio 'open' interface:



Opening a document in HyperStudio

The popular conception of what constitutes ‘computer literacy’ is to use a word-processing application, access and send e-mail messages, and access the internet (normally confined to variants of the Microsoft Windows family of operating systems).

If that is all that is required of educators / employers in terms of computer literacy there is no need to teach anything about computers in educational institutions as computers are at the stage where most people can manage these activities by themselves.

If by ‘computer literacy’ what is meant is an understanding of ‘Life-critical systems’:

“those that control air traffic, nuclear reactors, power utilities, staffed spacecraft, police or fire despatch, military operations, and medical instruments.”

(Shneiderman, 1998. p.16)

Then this is not what is generally thought of as taught in school.

If 'computer literacy' is understood to mean an ability to program computers then that is the subject area of specialist computer programming classes and degree programs.

"In the early days of computers, users had to maintain a profusion of device-dependent details in their human memories."

(Shneiderman, 1998. p.65)

What is tends not be observed is that since the advent of the GUI, users have had to maintain a profusion of platform-dependent (for 'platform' read 'operating system Graphic User Interface') details in their human memories; as well as certain device-dependent details.

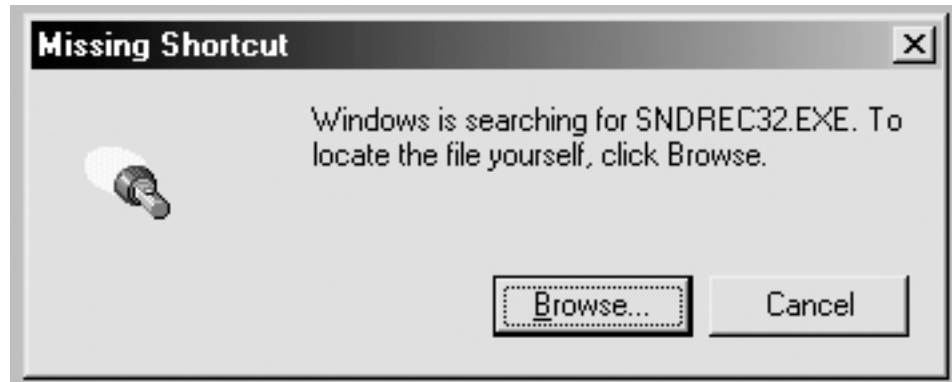
"The learning, use, and retention of this knowledge are hampered by two problems. First, these details vary across systems in an unpredictable manner. Second, acquiring syntactic knowledge is often a struggle because the arbitrariness of these minor design features greatly reduces the effectiveness of paired-associate learning."

(Shneiderman, 1998. p.66)

The golden rules of interface design (well stated and explained in Apple Computer's *Macintosh Human Interface Guidelines*. (Apple Computer, 1992.)) are that an interface should feature:

- Consistency. Controls that do similar things should look similar.

- Feedback. Users should be kept informed of what is happening in non-obscurantist language.

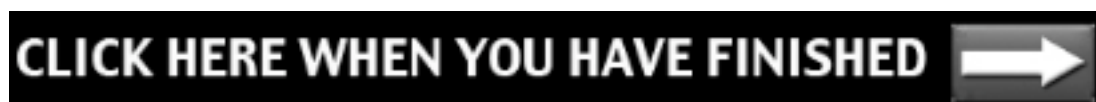


Obscure feedback signal from Microsoft Windows 98 SE
(as a result of clicking on an icon entitled “Sound Recorder”).



Feedback signal from prototype interface.

- User Control.



User control from prototype interface.

- The ability to ‘backtrack’ (Apple terms this “Forgiveness”)

— i.e. users should be able to change their minds and reverse actions taken.

- The ability to ‘backtrack’ (Apple terms this “Forgiveness”) — i.e. users should be able to change their minds and reverse actions taken.
- Aesthetic Integrity.



These observations seem to lead to three conclusions:

1. Computer literacy (however that term is to be understood) is not what should be taught by subject teachers (apart from those who teach computer science); nor should teachers be expected to be computer literate.
2. A platform-independent interface needs to be designed that requires almost no syntactic knowledge, and no computer literacy.
3. Any interface should be extensible to take into account new requirements as they arise.

3.2 AGENTS, VIRTUAL PERSONALITIES AND ‘WIZARDS’

An agent is “an entity that acts upon the environment it inhabits.”

(Wooldridge, 2000. p.1).

Unconscious agency consists of natural, non-conscious forces and

objects acting on other objects. An example of unconscious agency is where running water wears away stones.

Conscious agency Conscious Agency is where conscious objects (i.e. living things) act on other objects; either directly through physical acts acting on other conscious or unconscious objects, or, indirectly through the agency of language on other conscious objects.

An example of conscious agency acting on other objects is where I stir my cup of coffee with a spoon.

An example of a conscious agency acting on another conscious object through the agency of language is where I instruct my son to go to bed.

“In the practice of the use of language (2) one party calls out the words, the other acts on them. In instruction in the language the following process will occur; the learner *names* the objects; that is, he utters the word when the teacher points to the stone. -- And there will be this still simpler exercise: the pupil repeats the words after the teacher -- both of these processes resembling language.”

(Wittgenstein, 1968, 1995. Remark 7)

Wittgenstein’s pocket definition of what he terms ‘the language game’ is hard to fault when applied to human beings who are, generally, conscious beings capable of learning in the way Wittgenstein describes.

The only real problem with Wittgenstein’s definition is whether it

can be extended to computers, computer systems, and whether we can create conscious learning entities that inhabit computers rather like dualists would have the soul/spirit/mind inhabit the human body.

A computer may be said to 'contain' code. Programming languages may be used (via an interpreter) to instruct that computer to carry out various mathematical manipulations. in that code.

However, computers, unlike human beings cannot 'learn' linguistically in the sense that computers may not extend their vocabulary. Nor are computers able to learn new languages: new languages (and their compilers) are added to the computer either by inserting a ROM chip or copying code onto magnetic media.

A pet gerbil is a conscious agent insofar as it chews its way through anything put inside its cage. A gerbil is not, as far as can be ascertained, rational, as it will chew its way through a plastic water bottle thereby depriving itself of its water supply.

"An agent is said to be rational if it chooses to perform actions that are in its own best interests, given the beliefs it has about the world."
(Wooldridge, 2000. p.1).

Specifically, in the context of computers; an agent is a software package, part of a software package, or part of an operating system that appears to the end-user to behave like Wooldridge's description

‘inside’ a computer. Although, Wooldridge’s description doesn’t quite fit what computer agents do (despite his book being about computer agents) insofar as computers are made to serve human beings rather than themselves, and, following from that, agents that ‘inhabit’ computer systems should serve end-users. A modification of Wooldridge’s description that is more accurate is:

A computer agent *chooses* to perform software operations that serve what it *believes* to be the end-user’s best interests, given the information the end-user currently is presenting to it and its *knowledge* of that end-user’s past use of the computer.

The use of the word ‘belief’ here is inaccurate as computers cannot *believe* as humans do. All that a computer agent can use is ‘knowledge’; i.e. concrete data saved in data banks (punch cards, paper tape, hard-disks, and so on), and data being entered in real time.

From this point hence the term ‘agent’ is to be understood as confined to computer (i.e. software) agents.

An agent “should have the following properties:

- autonomy;
- proactiveness;
- reactivity; and
- social ability.” (Wooldridge, 2000. p.3).

In the context of a GUI for educators to develop bespoke computer

software for content delivery and reinforcement this needs some examination and, possible, refinement.

“An autonomous agent makes independent decisions — its decisions (and hence its actions) are under its own control, and are not driven by others.”

(Wooldridge, 2000. p.3).

While an educator sitting in front of his/her computer wants an ‘agent’ to develop a piece of software for them, they want to make the decisions with regard to form and content: i.e. the educator drives the decision making process rather than some agent within the software itself.

One of the common fears by non-computer specialists is that of loss of control.

“Proactiveness means being able to exhibit goal-directed behavior. If an agent has a particular goal or intention, then we expect the agent to try to achieve this intention.”

(Wooldridge, 2000. p.4).

An ‘agent’ working for educators should, ideally, exhibit goal-directed behaviour; but to achieve the intention of the end-user (educator) rather than the agent itself.

An agent that is goal-directed to develop, say, a recipe-book program, when the end-user wants a program to help students grasp the finer points of chemical titration is useless.

An agent that takes the information entered into a computer by the end-user (using, for instance, a series of screens taking the user through a decision tree) regarding the need for a program about chemical titration and then develops one (without the end-user having to do very much - especially in the area of computer programming) is exactly what is required.

“Being reactive means being responsive to changes in the environment.”

(Wooldridge, 2000. p.4).

Proactivity must be balanced by reactivity: an agent that continues to churn out programs about chemical titration because that is what the end-user required last month is no good. The agent must be able to react to new requests and, modify what it does as a result.

“To achieve our goals . . . we must *negotiate* and *cooperate* with these agents.”

(Wooldridge, 2000. p.4).

In our specific context the reverse is true; the agent must *negotiate* and *cooperate* with the end-user. And, in terms of negotiation only insofar as it can only present the end-user with possibilities determined by the possibilities built into its software.

‘Social ability’ is an odd term to use in this context unless the agent presents a ‘human face’ to the end-user so that the end-user feels that the agent is a friend rather than a collection of subroutines written in some obscure computer language.

“Metaphors and analogies are an important kind of learning used quite often in teaching. In teaching, the instructor chooses some concrete situation with which the student is familiar and presents new information in terms of how it relates to the old familiar information.”

(Eberts, 1994. p.208)

Agents are often represented to end-users as “virtual personalities”; i.e. human-like beings (or other entities) that somehow ‘inhabit’ the computer:



Agents

‘Wizards’ are effectively entirely *passive* agents that do not learn but merely help end-users navigate through predefined decision trees. They, normally, do not sport a virtual personality as a front-end.

In the field of authoring software both Toolbook Instructor II 8.5 and Macromedia Authorware 7 sport wizards of limited capabilities.

Outwith the field perhaps the most well-known wizard is that used by Microsoft’s Access database development package.

For the purposes of developing of either an additional or

different layer to replace the both current GUIs' and current authoring packages' dependence on a knowledge of programming in proprietary languages, it is felt that what is needed some type of wizard-agent hybrid with a virtual personality (c.f. the results of Workshop #1) with the following capabilities:

- The ability to 'learn'.

This can be implemented very simply as a capability for the agent to write to either some type of internal preference store, or to an external database.

- Exhibit goal-directed behaviour.

This must be based in part on what has previously 'learnt' and on what the end-user tells it in real time.

- Present the end-user with a set of structured questions to guide him/her through a decision tree to help the agent learn.

The main reason that a agent is needed is that no educators should have to learn anything about computers beyond the trivial.

"We do not (usually) build agents for no reason. We build them in order to carry out *tasks* for us. In order to get the agent to do the task, we must somehow communicate the desired task to the agent. This implies that the task to be carried out must be *specified* by us in some way . . . One way to specify the task would be simply to write a

program for the agent to execute.”

(Wooldridge, 2002. p.36)

Writing a program for an agent to execute means that the supposed agent is reduced to a compiler for a programming language (i.e. the routine that converts a set of instructions in a programming language into hexadecimal machine code for the processor to use). If this is what is understood to constitute an agent, the concept of ‘agent’ as discussed above simply evaporates.

What end-users need to be able to do is “*tell our agent what to do without telling it how to do it.*”

(Wooldridge, 2002. p.37) (Wooldridge’s italics)

To a large extent this is the point that has been entirely missed in all the existing authoring software packages. Almost all the existing packages have attempted to simply the *how*, but the *how* is still very definitely there.

What is needed for the purposes of this thesis is an agent that end-users can instruct in what to do, but never how to do it.

Further to this, end-users need to be able to instruct the agent what to do in an intuitive way that requires no knowledge of computer software (because, by its very nature as once as one strays into the area of software knowledge one is already involved with the *how* to a certain extent).

This necessitates a direct-manipulation interface.

Most of the literature on agents is concerned with implementing highly complex matters for programmers themselves; there seems a shortage of literature concerning how the agent concept may be developed to make computers so they are as easy to use as a pencil and paper.

3.3 WHAT HAS BEEN LEARNT

Agents are divided into 2 classes: those that work only highly complex computer routines for networking and other computer-programmer specific activities, and those that are ‘fitted’ with virtual personalities attached to office and internet software.

The former are not relevant to RAD packages; nor are the latter as they are normally embedded in packages that already presuppose quite a high level of computer sophistication.

3.4 CONCLUSION

While the types of agents that have been examined agents do not seem relevant to RAD packages, Wooldridge’s idea that it would be nice for end-users to tell an ‘agent’ *what* to do and let the ‘agent’ get on with the *how* is extremely attractive.

There is no real reason why a software creation interface should not sport what had best be termed a *fake agent*: that is to say a simulation which gives the end-user the impression a personality exists inside the computer with which (or whom) the end-user can

interact. This ‘agent’ can be a front-end for a decision-tree wizard.

4 GENERAL METHODOLOGY

While interface design and guidelines are dealt with extensively in the literature; both in academic textbooks such as Preece’s excellent book (Preece, Rogers and Sharp. 2002) and in various ‘Human Interface Guidelines’ publications (e.g. GNOME, 2004) the author has largely relied on Apple Computer’s *Macintosh Human Interface Guidelines* (Apple Computer. 1992.) as a ground plan for the underlying methodology. This is not to deprecate other publications;

merely that Apple's publication adopts a linear bare bones approach to matters that readily adapts itself to being used as a recipe book.

The author has not always adhered to Apple's terminology as he felt there are better, more accurate terms in certain cases.

The questionnaire and workshops were entirely separate, although participants in the workshops had previously completed the questionnaire (without being aware that they would subsequently be invited to attend workshops). However, workshop participants all stated that their having completed the questionnaires helped to sharpen their perceptions when taking part in the workshops.

4.1 WORKSHOPS AND PROTOTYPING

Using stages based on those outlined in Apple Computer's *Macintosh Human Interface Guidelines* (Apple Computer. 1992.) it is shown how workshops are a necessary part of the development of a software creation interface.

AUDIENCE

“Features Inspired by Market Pressures”

(Apple Computer. 1992. p.34)

The 'market' initially envisaged is the Primary and Secondary education sectors in the United Kingdom.

Market pressures / needs / wants / expectations were elicited by analysis of responses to the survey and workshop feedback.

FEATURE CASCADE

“When deciding whether or not to add features to your product, think about whether the benefits to users of additional capabilities outweigh the additional development efforts, growth in size, and reduction in running speed that the features would cost.”

(Apple Computer. 1992. p.35)

While developing a vertical prototype these considerations are not immediately relevant. However, if the software were to be developed further, towards some type of viability; this might have to be considered.

Runtime Revolution / Metacard (RR/MC) (the software used for the development of the prototypes) can either reference external resources (images, sound files, video files, etc.) or embed them. For the purposes of this project resources must be embedded to make it portable and to avoid resources being lost in differing directory systems on different platforms and individual computers.

The problem about any piece of software containing embedded

resources is that all these resources must be loaded into the RAM (Random Access Memory) of the computer when the software is opened.

A considerable amount of time during prototype development was spent on examining to what extent resource files could be compressed without an unacceptable loss of quality:



**8 bit PICT file
104 KB**



**2 bit PICT file
88 KB**

The fact that the project was to be developed to work on as many operating systems as the underlying RR/MC engine could provide for meant that there were file format constraints that had to be taken into account.

PICT files are Macintosh specific

JPEG files will not display on a Macintosh if the RR/MC software is run directly from a CD-ROM

BMP files are not handled well by Linux systems.

As this is to be a piece of software (a GUI) that is to be used to develop software for educators running speed should not be top priority for a number of sharply defined reasons:

- This is thought of as a virtual personality-led software that should be regarded by educators as a ‘friend’:

“These [synthetic characters] are commonly designed as 3D characters . . . Much effort goes into designing them to be lifelike, exhibiting realistic human movements, . . . The design of the characters’ appearance, their facial expressions, and how their lips move when talking are also considered important interface design concerns.”

(Preece, Rogers and Sharp. 2002. p.157)

- This ‘friend’ should proceed sufficiently slowly, step-by-step, as to allow educators think their way through the piece of software that they are negotiating with the ‘friend’; i.e. this should not be seen as confrontational software (‘this is what is good for you, you know-nothing teacher, so I am going to make it for you in this way, whether you want it or not’) but as a collaborative tool.
- The end-products prepared by the software should be seen to be created both in real time and directly in front of the educator so that the educator can see and appreciate that the software is doing exactly what they need. This should also both help the educator to understand just how their end-product will work and develop better ideas for future end-products.

- To achieve this goal, the software must needs be feature rich as it will have to guide educators through complex, multitiered decision trees, and contain all the necessary components to implement the decisions that educators make.

THE 80 PERCENT SOLUTION

“The 80 percent solution means that your design meets the needs of at least 80 percent of your users. If you try to design for the 20 percent of your target audience who are power users, your design will not be usable by the majority of your users.”

(Apple Computer. 1992. p.35)

As this software solution is being developed “for the rest of us” (i.e. non-programmers), the power users are automatically excluded. Those educators who already author software using an authoring package do not need this at all.

It is doubtful whether 20 percent of the ‘market’ are power users. There are, however, limitations on whose needs the design can meet (and, in turn, this also sets a limit on feature cascade). Many educators, not being computer specialists, are profoundly unaware of both what computers can and cannot be made to do. A certain proportion of educators’ expectations of what this type of software could do for them fall partly or completely outside the capabilities of computers.

PROGRESSIVE DISCLOSURE

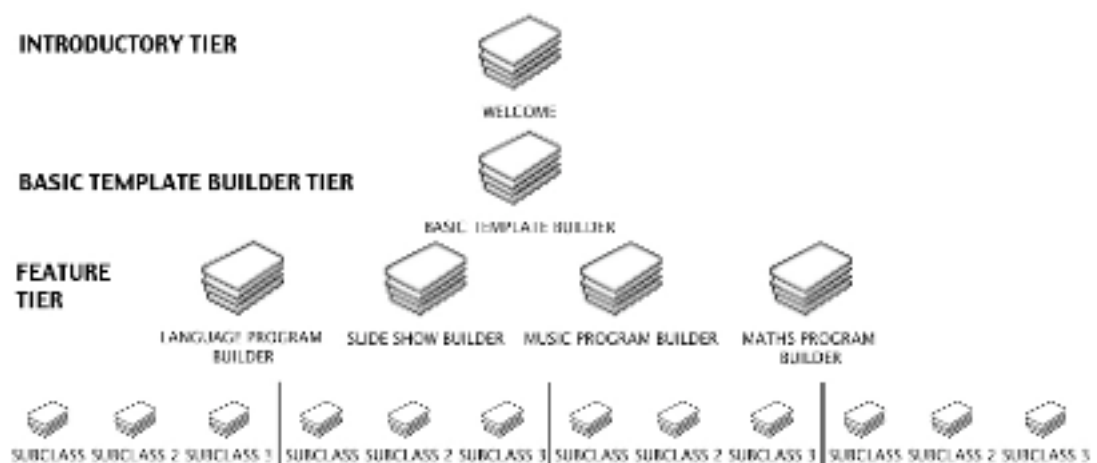
“Using Progressive Disclosure”

(Apple Computer. 1992. p.35)

“Your application should enable the user to concentrate on the task at hand. So, design your application to show only useful and relevant information and interface elements. Every extra piece of information or interface control competes with the truly relevant bits of information and distracts the user from important information. Hence, don't clutter your interface, and don't overload the user with buttons, menu options, icons, or irrelevant information. Instead, use progressive disclosure and other techniques to limit what the user sees at any given moment.”

(GNOME. 2004. p.17)

The prototype is developed on a multitiered structure consisting of embedded xTalk stacks (for reasons of clarity images of Hypercard stacks have been used here although the prototype was developed using Runtime Revolution and Metacard):



Multitiered structure of prototype.

Educators are not introduced / exposed to a new tier until they have completed the decision-tree presented to them in the preceding tier.

IMPLEMENTING PREFERENCES

“Preference settings are user-defined parameters that your software remembers from session to session.”

(Apple Computer. 1992. p.37)

Each tier of the finished software product should have a learning system (memory / preferences file) so that at subsequent re-use of the software educators can bypass a tier should they so wish.

In the prototype preference settings are implemented in the Basic Template Builder tier.

STAKEHOLDER INVOLVEMENT

“Involving Users in the Design Process

“The best way to make sure your product meets the needs of your target audience is by exposing your designs to the scrutiny of users.”

(Apple Computer. 1992. p.41)

“A prototype allows stakeholders to interact with an envisioned product, to gain some experience of using it in a realistic setting, and to explore imagined uses.”

(Preece, Rogers and Sharp. 2002. p.241)

This is exactly the point of conducting workshops with educators who were selected on the basis of two criteria only: that they had never learnt a computer language, and / or designed a bespoke piece of software for content reinforcement or delivery.

Preliminary soundings were also taken with an extensive questionnaire completed by 18 educators.

“You can do this during every phase of the design process to help reveal what works about your product as well as what needs improvement.”

(Apple Computer. 1992. p.41)

Owing to time constraints on both the author and the members of the focus-group taking part in workshops there were only 2 workshops: one at an early stage (prototype #1), and one at a late stage (prototype #2) of the development cycle.

PROTOTYPES

Three initial very basic prototypes (prototypes #0.1, #0.2 and #0.3) were built before submission of the thesis proposal to explore the possibilities of feature automation using xTalk.

All three of these basic prototypes were shown on an informal basis to teachers in their own homes to make sure the author was not completely wasting his time. Prototype #1 was used as the basis for

workshop #1 (June 2004).

Prototype #2 was used as the basis for workshop #2 (November 2004).

Prototype #2 differed significantly from prototype #1 in that:

- 1 Prototype #1 finished at the end of the basic Template Builder tier.

Prototype #2 implemented the Language Program Builder (one subclass) and the Music Program Builder tiers.

- 2 Prototype #1 had no memory (i.e. it could not save settings from a previous development session).

Prototype #2 implemented a memory system for the preceding session and a rapid development channel whereby the prototype could develop a basic template with no further prompting of those previous settings were selected.

- 3 Prototype #1 featured a 'female' virtual personality with an entirely artificial voice (the 'Victoria' voice from the Apple speech synthesizer present in Mac OS X).



‘Kala’

Prototype #2 featured a ‘male’ virtual personality using a voice consisting of pre-recorded segments from a real person.



‘Dan’

- 4 Prototype #1, while featuring a virtual personality supported the spoken prompts with no textual backup.

Prototype #2 featured textual reiteration of the spoken prompts of the virtual personality.



Textual support

- 5 Prototype #1 could not save the program developed by the interface (i.e. when the prototype was exited, the work was lost).

Prototype #2 saved the program developed at two stages in its development: at the end of the Basic template Builder tier, and at the end of the Feature tier.



Exit screen for Music Program Builder

Prototype #2 also was capable of saving itself so that the preferences created were not lost.

Points 3 and 4 were acted on on the basis of the representations of the focus group.

A third prototype ("Prototype #3") was not built after Workshop #2 owing to time constraints. However, a list of changes and suggestions for a third prototype was drawn up on the basis of Workshop #2.

USER OBSERVATION

"Once you have a prototype drawn or mocked up, you can begin to show it to people to get reactions to it"

(Apple Computer. 1992. p.42)

Prototype #1 was shown to the focus-group running on a number (1 per focus-group member) of IBM-compatible personal computers running Microsoft Windows 98, with 15 inch monitors set at 1024 x 768 pixel resolution in June 2004.

TEN COMMANDMENTS

“Ten Steps for Conducting a User Observation”

(Apple Computer. 1992. p.43 - 46)

These were adhered to rigidly:

“1. Introduce yourself and describe the purpose of the observation.”

The author introduced himself as an educator with 15 years experience in teaching and 28 years experience in computer programming. he outlined his belief that computers are underused in educational institutions and that their primary role should not be as word-processors or sources of entertainment but as tools which educators can use to deliver content to pupils.

The author explained that he was attempting to develop a way for teachers who had little or no experience with computers and no time and / or inclination to learn how to program computers to

‘knock up’ small programs for content delivery and reinforcement for pupils to use as all or part of a classroom session.

The author explained that he wanted the participants in the workshop to try to develop the target software themselves without help from him and that whilst they were doing that to concentrate on what they felt were the shortcomings of the development package.

“2. Tell the participant that it’s OK to quit at any time.”

The author told participants they could stop whenever they felt that they had had enough, and that they were free to leave the room whenever they felt like it.

“3. Talk about the equipment in the room.”

The author pointed out that the participants would be using the development package on the ‘usual PCs’ that they used with their pupils for word-processing and that they would only have to type a few words and click on screen objects.

“4. Explain how to think aloud.”

The author explained that he would be sitting among the participants and the way they could help him most was if they could tell him about any problems they were experiencing and any other ideas that came to mind as they worked their ways through the

development package.

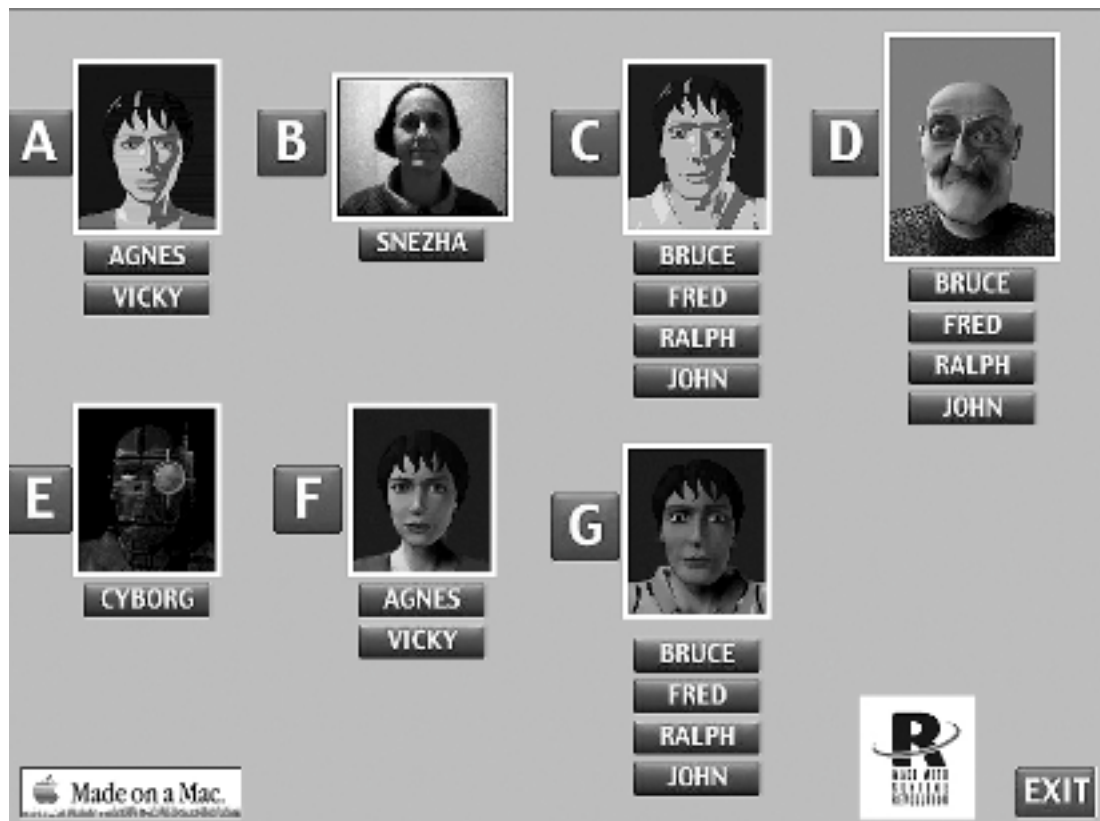
“5. Explain that you will not provide help.”

The author pointed out that the whole *raison d'être* of the development package was that it should be capable of being used by non-computer ‘savvy’ teachers without any help whatsoever, and, therefore, he would be offering no help.

“6. Describe in general terms what the participant will be doing.”

In Workshop #1 the author explained that participants would be:

- i. examining a number of images and listening to a number of voices of potential virtual personalities. The author gave a brief outline of what constituted a virtual personality from an end-user's point of view.



Agent chooser interface

(clicking on a named button runs
the animation above with a voice)

- ii. trying to develop a template for a program to be constructed on top of. He also directed their attention to the (brief) instruction and feedback sheet.

[see appendix containing workshop worksheets]

“7. Ask if there are any questions before you start; then begin the observation.”

“8. During the observation, remember several pointers:

“◇ Stay alert . . . A great deal of the information you can obtain is subtle.”

The author noticed several times participants made comments to each other (of which, may be, they thought the author was unaware) which were extremely informative.

“◇ Ask questions or prompt the participant.”

If a participant paused for a significant time the author asked them why.

The author never prompted any of the participants.

“9. Conclude the observation.”

The author thanked the participants for taking part, explained that he was trying to see whether the participants could construct the target software using the development package, and asked them for any further feedback.

“10. Use the results.”

All the results from Workshop #1 were taken into account when building Prototype #2.

4.2 QUESTIONNAIRE DESIGN

A questionnaire was designed with the object of eliciting attitudes of Primary and Secondary educators to the use of computers and associated software in schools. There was an emphasis placed on teacher attitudes to developing their own software for content delivery and reinforcement.

It was decided that the questionnaire should be self-administered to avoid the respondents feeling pressured either by the author, the school authorities or colleagues.

“The self-administered questionnaire is usually presented to the respondents by an interviewer or by someone in an official position, such as a teacher . . . The purpose of the enquiry is explained, and then the respondent is left alone to complete the questionnaire . . .”

(Oppenheim, 1992. p.103)

The survey was presented to teachers at Greyfriars Primary School (St. Andrews, Scotland) by the headmistress, and at Madras College (St. Andrews, Scotland) by the vice-principal.

Each copy of the questionnaire was accompanied by a consent form.

Questionnaires and consent forms were collected via 2 separate cardboard boxes in the relevant school offices. 2 boxes were used to dissociate consent forms from complete questionnaires to preserve anonymity.

The purpose of the questionnaire was explicitly stated at the top of the first page of the questionnaire:

“This is a survey designed to ascertain certain aspects of computer use in schools, how comfortable teachers feel using computers for making their own programs, and whether teachers feel they would enjoy and benefit from having access to an innovative type of computer programming interface to help them develop programs for their own to use with their pupils.”

(see Appendices)

Teachers were given 15 days in which to complete the questionnaire.

It was explicitly stated in the consent form that:

“The survey forms are completely anonymous and as such there is no risk of your privacy being compromised.”

and in the questionnaire

“This is an anonymous survey and all responses will be treated in complete confidence.”

and

“You are under no obligation to answer any questions in this survey.”

An attempt was made to avoid any personal questions that were not directly relevant to the object of the questionnaire.

Although some questions were personal (age, gender, how long one has been a teacher, “If you have children do you allow them to play computer games?”, etc.) it was felt that the anonymous nature of the questionnaire should both protect individual teachers and make they themselves feel secure in answering them.

“. . . our questionnaire will consist of a series of question *modules* or sequences, each concerned with a different variable.”

(Oppenheim, 1992. p.109)

The questionnaire was divided into the following sections in the following order:

SITUATIONAL QUESTIONS

These questions relate to the teacher’s level of teaching experience and their school.

INDIVIDUAL COMPUTER USAGE

This section tries to elicit whether the teacher has used a computer, and , if they have, their level of comfort using one. It asks whether the teacher has a computer at home and their attitude towards children using computers.

EDUCATIONAL COMPUTER USAGE

This is an in-depth section trying to elicit the teacher's level of computer use with pupils, and the type of computer use.

It also asks whether the teacher has ever developed his/her own computer program for pupils.

PROGRAMMING EXPERIENCE

This asks extremely specific questions relating to computer authoring packages that the teacher may have used.

How the teacher may have learnt to use an authoring package, as well as their self-assessment as to their level of competence.

INTERFACE FAMILIARITY

Asks questions about the type of generic tool palettes sported by authoring software packages.

Ask whether the teacher has seen / is familiar with any of a number of virtual personality type agents.

VIRTUAL AGENTS AND YOU

Asks the teacher's opinion about virtual personality type agents.

Each section starts with “*filter*” questions (Oppenheim); and each section acts as a filter for the subsequent sections.

This stops teachers wasting their time filling in sections that are not relevant to them, and will yield useless data if they are filled in by teachers who should be excluded from them.

“The *funnel approach*, preceded by various ‘filter’ questions . . . , is one well-known type of sequence which is often used . . . The funnel approach is so named because it starts off the module with a very broad question and then progressively narrows down the scope of the questions until in the end it comes to some very specific points.”

(Oppenheim, 1992. p.110)

Almost exclusively closed questions are used except where answers cannot be predicted (e.g. number of pupils in a class, opinion on computer use). This is to facilitate any types of analysis that may be performed on the completed questionnaires.

Foddy’s ‘TAP’ paradigm was followed throughout the construction:

“Topic

The topic should be properly defined so that each respondent clearly understands what is being talked about.

“Applicability

The applicability of the question to each respondent should be established: respondents should not be asked to give information

that they do not have.

“Perspective

The perspective that respondents should adopt, when answering the question, should be specific so that each respondent gives the same kind of answer.”

(Foddy, 1993. p.193)

The **Topic** was clearly defined at the top of the first page of the questionnaire.

The **Applicability** was established by the funnel approach: each section funnelling into the next section only those teachers who could give the information that was required:

SITUATIONAL QUESTIONS



INDIVIDUAL COMPUTER USAGE

0.5 Do you have a computer in your home ?

YES

NO

questions 0.5.1 - 9 about home
computer usage and attitudes



EDUCATIONAL COMPUTER USAGE

1. Have you ever used a computer program with pupils ?

NO

YES

questions 1.1 - 1.5 about school
computer usage and attitudes

1.3 Have you ever developed your own computer
program for use with your pupils?

1.4 Have you ever developed a computer program for
some other purpose than for use with your pupils?

YES

NO

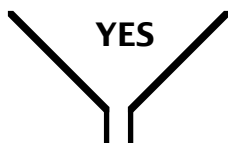


Complete **INTERFACE FAMILIARITY**

1.5 Would you like to learn to make small topic-
specific computer programs to use with your pupils ?

YES

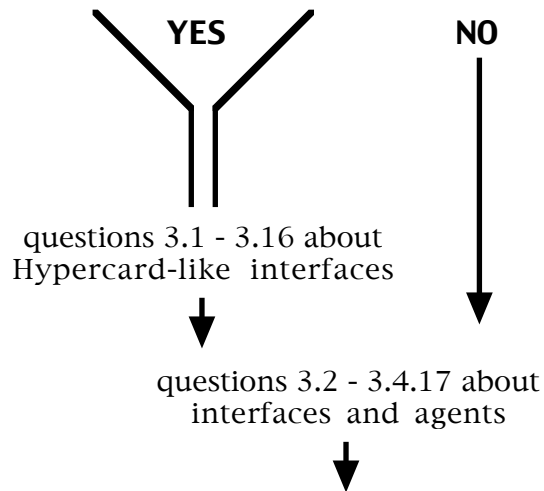
NO



Funnelling for Applicability [1]

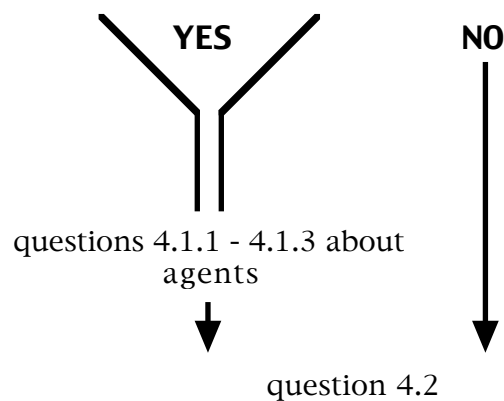
INTERFACE FAMILIARITY

If you have no experience with Hypercard please go onto question



VIRTUAL AGENTS AND YOU

Have you ever used a Virtual Agent of any of the types in Section 3 in connection with computer work you have done?



Funnelling for Applicability [2]

The **Perspective** was controlled by the dependence on predominantly closed questions, e.g:

4.1.2 If you used any of the first 3 types of Agent did you feel that:

- | | | |
|---------|-------------------------------|--------------------------|
| 4.1.2.1 | It helped me considerably | <input type="checkbox"/> |
| 4.1.2.2 | It was moderately helpful | <input type="checkbox"/> |
| 4.1.2.3 | It did not help me at all | <input type="checkbox"/> |
| 4.1.2.4 | It distracted me from my work | <input type="checkbox"/> |
| 4.1.2.5 | I hated it | <input type="checkbox"/> |

5 ANALYSIS OF QUESTIONNAIRE RESPONSE

[Numbers starting with a 'Q'; e.g. (Q. 1.4) refer to questions in the questionnaire.]

The questionnaire was completed by all the respondents who took part in Workshop #1 prior to their taking part in the workshop and being introduced to the Prototype #1 personality.

The Questionnaire was distributed to teachers at:

Greyfriars Primary School: a Catholic church affiliated state primary school in St Andrews, Scotland.

Madras College: a state high school in St Andrews, Scotland.

St Leonard's: a private sector school containing both a primary and a secondary department.

Six Greyfriars teachers filled in and returned questionnaires. Seven

questionnaires were distributed.

Twelve Madras teachers filled in and returned questionnaires. Fifty questionnaires were distributed.

The teacher at St Leonard's deputed to distribute questionnaires did not. When questioned about this she said that she "couldn't be bothered". Fifty questionnaires were delivered to the school.

It would appear that the Greyfriars and the Madras teachers who completed their questionnaires did so for very different reasons:

The headmistress of Greyfriars school explained to the author that the teachers had completed the form because the author's children went to the school.

The teachers at Madras who completed their questionnaires appear to have been entirely self-selecting; that they have spent some considerable time and thought working through the latter part of the funnelling process (which the Greyfriars teachers did not; admittedly because they were all blocked by the funnels) demonstrates that to a certain extent they seem to have been teachers with some above average stake in computer use.

"Could there have been a 'volunteer bias', where subjects allocate themselves to the experimental or the control group, perhaps because they are more, or less, interested in the topic?"

(Oppenheim, 1992. p.30)

Undoubtedly so; but it is not clear whether this should be a cause for concern:

- The questionnaire in itself was designed to filter / funnel out disinterested parties — arguably all the teachers at Madras College were doing was helping that process along.
- The whole ‘point’ of the project is to design an interface for software design for teachers who are, either by choice or by work description, expected to use computers for at least certain aspects of the educational process. The author believes that there are an increasing number of teachers who have to deliver content via computer but who have neither the requisite knowledge to *program* (as that word is currently understood) computers, nor the time or inclination to learn how to do so.
- Another way of pointing out the advantage of self-selection by respondents is that, unlike in Oppenheim’s question, there was no control group who were, say, given a ‘spoiler’ or dummy questionnaire.

As explained the questionnaire was separated into sections that by involving a funnelling technique separated out teachers with and without various levels of computer experience.

The analysis that follows deals with responses section by section.

5.1 SITUATIONAL QUESTIONS

Teachers seemed to fall into 2 types determined by age;
new teachers and what we could term 'career teachers':

| | | | | | | | | |
|---|-----|------|-------|-------|-------|-------|-------|-------|
| 4 | | | | | | | | |
| 3 | | | | | | | | |
| 2 | | | | | | | | |
| 1 | | | | | | | | |
| | 1-5 | 6-10 | 11-15 | 16-20 | 21-25 | 26-30 | 31-35 | 36-40 |

Greyfriars respondents by numbers of years served.

| | | | | | | | | |
|---|-----|------|-------|-------|-------|-------|-------|-------|
| 4 | | | | | | | | |
| 3 | | | | | | | | |
| 2 | | | | | | | | |
| 1 | | | | | | | | |
| | 1-5 | 6-10 | 11-15 | 16-20 | 21-25 | 26-30 | 31-35 | 36-40 |

Madras respondents by numbers of years served.

| | | | | | | | | |
|---|-----|------|-------|-------|-------|-------|-------|-------|
| 5 | | | | | | | | |
| 4 | | | | | | | | |
| 3 | | | | | | | | |
| 2 | | | | | | | | |
| 1 | | | | | | | | |
| | 1-5 | 6-10 | 11-15 | 16-20 | 21-25 | 26-30 | 31-35 | 36-40 |

All respondents by numbers of years served.

All the Greyfriars respondents were female (all the Greyfriars teachers are female).

4 of the Madras respondents were female; 8 respondents were male.

5.2 INDIVIDUAL COMPUTER USAGE

This section was designed to elicit the level of domestic computer usage among respondents.

All but one of the respondents claimed to have had experience with computers. The platform that the majority had had experience of was Microsoft Windows.

The one respondent who claimed no knowledge of computers nevertheless stated that she believed that some computer games have educational value (Q. 1.6) and that some concepts can be taught

through the medium of [computer] games (Q. 1.6.1). in the **EDUCATIONAL COMPUTER USAGE** section.

Respondents who answered **YES** to questions Q.0.5 to Q.0.5:

[The first number represents those who answered **YES**, and the second number represents the total who answered that question.]

| | | |
|---------|---|-------|
| Q.0.5 | Do you have a computer in your home? | 16/18 |
| Q.0.5.1 | Do you use your computer to connect to the internet? | 14/16 |
| Q.0.5.2 | If you have children do you allow them access to the internet? | 6/6 |
| Q.0.5.3 | Do you play computer games? | 4/16 |
| Q.0.5.4 | If you have children do you allow them to play computer games? | 6/6 |
| Q.0.5.5 | Do you use your home computer for preparing educational materials for school? | 13/16 |
| Q.0.5.6 | Do you use your computer for listening to music? | 5/16 |
| Q.0.5.7 | Do you use your computer for watching films (such as VCDs or DVDs ? | 5/16 |
| Q.0.5.8 | Do you have a printer at home? | 16/16 |
| Q.0.5.9 | Do you have a scanner at home ? | 9/16 |

As very few of the respondents use their home computers for

listening to music or watching films we can conclude that most of them view a computer as a tool rather than the ‘centre of their digital lifestyle’.

5.3 EDUCATIONAL COMPUTER USAGE

Most respondents appear to use their home computers to prepare educational materials. It is only by their response to questions later in the questionnaire that we understand the nature of the educational materials they generally prepare:

| | | |
|-----------|---|----|
| Q.1. | Have you ever used a computer program with pupils? | |
| Q.1.1 | What type of program was it ? | |
| Q.1.1.1 | A word-processing program (e.g. Microsoft Word)? | 15 |
| Q.1.1.2 | A presentation development package (e.g. Microsoft Power Point)? | 9 |
| Q.1.1.3 | An educational CD-ROM? | 13 |
| Q.1.1.3.1 | A CD-ROM-based encyclopaedia? | 8 |
| Q.1.1.3.2 | A tightly focussed educational package? | 10 |
| Q.1.1.4 | A topic-specific program developed by a member of staff at your school ? | 1 |

i.e. they, predominantly, prepare Power Point presentations (computerised slide shows) and handouts for printing.

Fife Council (the education authority for both the schools surveyed) has been particularly generous in supplying their schools with off-

the-shelf CD-ROMs.

The single teacher who had developed her own topic-specific program was, unsterotypically, a Drama teacher.

What is equally informative is that two of the respondents were teachers of Computing; yet they had not developed programs for any purpose to use with their pupils.

There was quite a large positive response to:

Q.1.5 Would you like to learn to make small topic-specific computer programs to use with your pupils ? 10/13

Which might lead one to conclude that there is a gap between respondent's capabilities and desires; or, at least, between their capabilities as they themselves perceive them and desires.

Possibly one of the most informative things about this questionnaire is the contradiction between a comment scribbled at the top of a copy completed by a teacher of Computing:

"Totally unapplicable to just about everyone in this establishment"

referring to;

"This is a survey designed to ascertain certain aspects of computer use in schools, how comfortable teachers feel using computers for making their own programs, and whether teachers feel they would

enjoy and benefit from having access to an innovative type of computer programming interface to help them develop programs for their own to use with their pupils.”

[underlining used to duplicate effect of respondent’s orange highlighting pen]

and how many teachers responded positively to Q.1.5 .

5.4 PROGRAMMING EXPERIENCE

[here programming was used in a narrow sense to refer to building programs with authoring packages. However, several teachers misunderstood this and took it to refer to programming in a broader sense encompassing more ‘traditional’ types of programming: this proved informative.]

None of the Primary teachers had had any programming experience. As a result they were filtered out of this section entirely.

Three of the High School (Secondary) teachers had had programming experience.

All three had computing experience with some of the authoring packages listed in the **PROGRAMMING EXPERIENCE** section:

| | | |
|---------|-------------|-----|
| Q.2.1.1 | HYPERCARD | 2/3 |
| Q.2.1.2 | HYPERSTUDIO | 3/3 |

| | | |
|----------|-----------------------|-----|
| Q.2.1.3 | MACROMEDIA AUTHORWARE | 1/3 |
| Q.2.1.4 | MACROMEDIA DIRECTOR | 2/3 |
| Q.2.1.5 | SUPERCARD | 1/3 |
| Q.2.1.6 | TOOLBOOK | 1/3 |
| Q.2.1.21 | VISUAL BASIC | 3/3 |

[Those authoring packages which had been used by no one have been excluded here]

One of the respondents (a Computing teacher) had experience in more traditional languages: BASIC, PASCAL, C, CShell, LISP and PROLOG.

5.5 INTERFACE FAMILIARITY

Despite this they were unable to identify with any accuracy iconic items in the toolbars displayed in Q.3.1.6 and Q.3.2.1.

Certainly, these respondents, despite having experience with authoring packages did not have an intuitive grasp of a connection perceived by the developers of HyperCard and Runtime Revolution between the icons on their respective toolbar and the objects and actions they are supposed to represent.

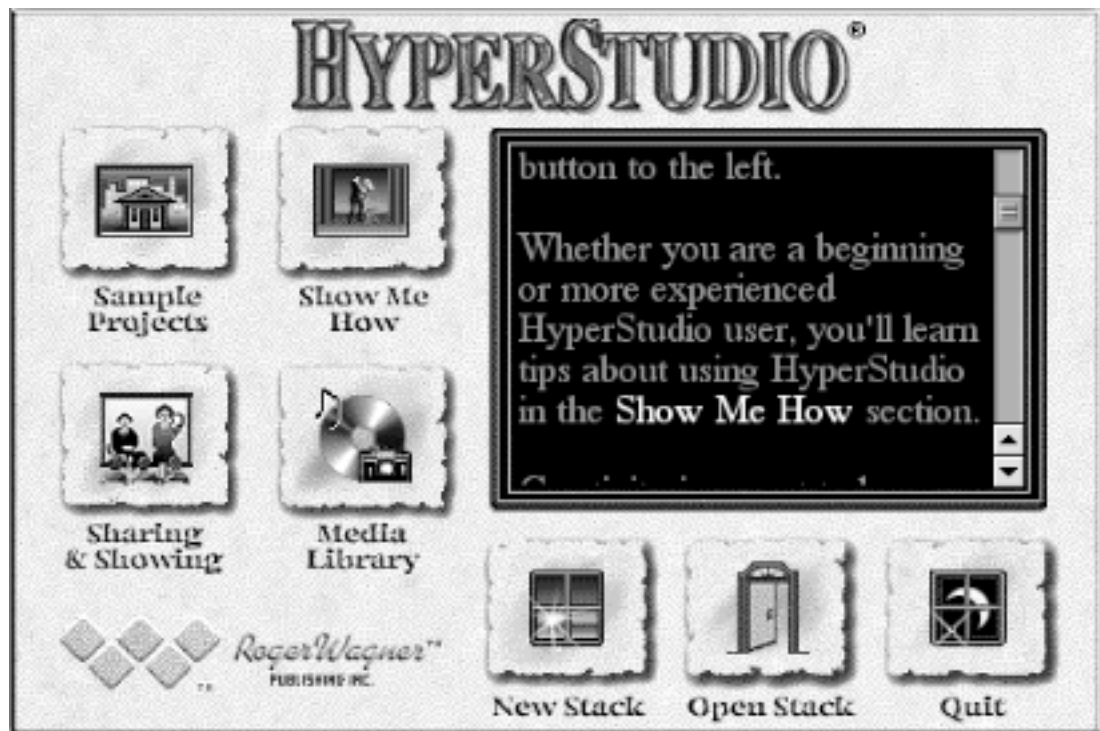


HyperCard toolbar

This might be used to argue that the interfaces of these authoring packages are not as intuitive as their protagonists might like us to believe. Two of the High School respondents who had no programming experience and one of the Primary level respondents tried questions Q.3.1.6 and Q.3.2.1. with similar levels of success as those who claimed that they had.

Those respondents who answered Q.3.3 all favoured the colourful, pictorial interface Q.3.3.1 (the HyperStudio startup screen) to Q.3.3.2 (the HyperCard toolbar).

The HyperStudio startup screen (already adopted in HyperStudioIIs for the Apple II.) is an initial move away from a WIMP interface towards an Object-Point-only interface:

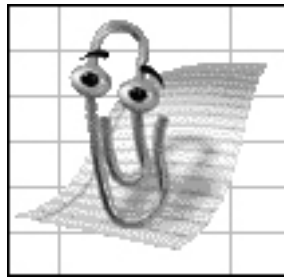


HyperStudio 3.2 startup screen

Q.3.4. presented respondents with a series of pictures of agent-like characters including 14 ‘personalities’ associated with software in some respect or another, and one ‘spoiler’: “Max Headroom”, a television virtual personality from the early 1980’s (conceived of as computer-generated).

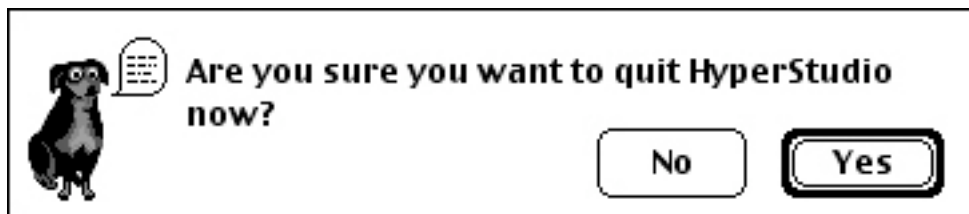
Not all of the 14 ‘personalities’ are, in fact, part of agents, as such. However, those that are not agents are, in some way, attached to dialogue boxes as part of a software package.

An example of a personality attached to a Help engine is the animated paper-clip used by Microsoft Windows versions of Microsoft Office:



Microsoft Office Virtual Personality.

An example of an image used to, in some way, enforce or highlight dialogue boxes is 'Addy' used in HyperStudio 3.2:



HyperStudio 3.2 Dog

It is not entirely clear what the object of this dog is. It is nothing more than a static image; it is not animated, nor does it talk. It only appears in a very few dialogue boxes.

The author feels that the developers of HyperStudio were, at one time or another, playing with the idea of moving away from a WIMP-and-code interface to one that was to have been entirely Object-Point-only, and the series of screens in the 'Home' stack and the pictures of 'Addy' are, so to speak, fossils of these ideas which

never came to fruition.

All of the respondents who answered Q.3.4. recognised the Microsoft Office paper clip (Q.3.4.3) — hardly surprising, MicroSoft Windows being the *de facto* standard in Fife Council schools (and throughout the UK).

2 respondents managed to identify images attached to internet ‘bots’ (Q.3.4.9 and Q.3.4.14).

None of the 3 respondents who claimed to have had experience of HyperStudio recognised ‘Addy’ (Q.3.4.1): perhaps because it is largely irrelevant in terms of the HyperStudio interface.

5.6 VIRTUAL AGENTS AND YOU

[With the wisdom of hindsight the author would not have chosen this slightly silly title for this section.]

This section was intended to give the author an idea of the level of consciousness about virtual personalities amongst the respondents:

Q.4.1 Have you ever used a Virtual Agent of any of the types in Section 3 in connection with computer work you have done? 3/14

Q.4.1.1 Was this with:

- | | | |
|---------|-------------------------------|---|
| 4.1.1.1 | An Application Helper Agent | 2 |
| 4.1.1.2 | An Internet Web Search Helper | 2 |

| | | |
|--|---|---|
| 4.1.1.3 | A Virtual Secretary | 1 |
| 4.1.1.4 | One of the other types | |
| Q.4.1.2 If you used any of the first 3 types of Agent did you feel that: | | |
| 4.1.2.1 | It helped me considerably | 1 |
| 4.1.2.2 | It was moderately helpful | 1 |
| 4.1.2.3 | It did not help me at all | |
| 4.1.2.4 | It distracted me from my work | 1 |
| 4.1.2.5 | I hated it | |
| Q.4.1.3 Do you prefer: | | |
| 4.1.3.1 | Agents that communicate via text messages | 2 |
| 4.1.3.2 | Agents that communicate via some type of speech synthesis | 1 |
| 4.1.3.3 | A plain-vanilla interface with a toolbar like the one shown in question 3.3.2 | 2 |

Q.4.2 Do you think you would like to work on a computer with some type of Virtual Agent to help you ? 9/13

As this is largely limited to responses given by 3 respondents (with the exception of Q.4.2) regarding the Microsoft Office paper clip it should be regarded as virtually valueless.

5.7 CONCLUDING REMARKS

As this questionnaire was filled in by a limited number of teachers, and very few of them completed all the sections, none of these findings should be taken as in any way as definitive.

What can be seen is that the majority of respondents have home

computers and are quite capable of connecting to the internet (receiving and sending e-mail messages), word-processing and using Microsoft Power Point.

While these respondents probably consider themselves computer literate; they are not anywhere close to 'computer literate' as they are unable to program computers.

Most of the respondents have used those skills as well as utilising pre-packaged non-modifiable software items (CD-ROMs) with pupils.

Many have used pre-packaged modifiable educational software with pupils.

Virtually none of the respondents have ever developed their own software items for content delivery or reinforcement, yet most of them would like to learn how to do that.

Most respondents do not find tool bars/palettes easy to understand.

This may be why a majority feel that some type of agent / virtual personality led programming environment might be a good thing.

6 ANALYSIS OF RESPONSE TO WORKSHOP #1

6.1 THE WORKSHOP

The participants were 4 female primary-level teachers at Greyfriars primary school in St Andrews, Scotland.

St Andrews has 4 Primary schools; not one of them has a male teacher. This is symptomatic of primary-level education throughout Scotland.

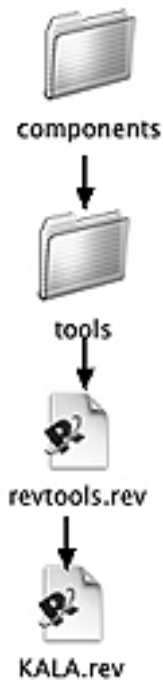
The participants were self-selecting. All the participants had all completed questionnaires.

Whether the fact that the participants had completed questionnaires predisposed them towards some of the

concepts introduced during Workshop #1 is unknown; it would, however, seem reasonable to assume that that had ‘prepped’ them to a certain extent.

Workshop #1 took place in a room dedicated for computer use. Each participant had access to a computer running Microsoft Windows 98 SE. Iconic links to both the Agent Chooser Interface and Prototype #1 were available on the operating system desktop.

Prototype #1 was embedded in the Runtime Revolution components directory as a substack of the *revtools.rev* stack:



Prototype #1: single stack 'KALA'
embedded within RR file structure

The worksheets were distributed to the participants, ground rules for the workshop were explained.

All the participants said that they had never even thought that it would be possible to develop software for content delivery and reinforcement without many years of experience of computer programming. They also observed that, while the pre-packaged educational CD-ROMs available at the school were, by and large, extremely good, they suffered from one problem: all the CD-ROMs were so full of features and content that it was impossible to keep children focussed on a single topic.

It was also pointed out that their pupils tended to treat time spent

working with the computers as recreational time. The author observed that that might be a consequence of the revolution in education that had its roots in the 1960s where primary education in the UK moved from a rote-learning process to an interactive 'learning-is-play' activity and not the fault of either the designers of educational CD-ROMs or other software available to pupils on the school's computers.

All the participants explained that most of the educational CD-ROMs available at the school fostered the view of education as a recreational activity. They all expressed some dissatisfaction with their jobs insofar as the idea of education as recreation sometimes interfered with their need to inculcate educational basics (e.g. spelling) in children.

The 3 most experienced participants (the least experienced participant was a student teacher serving a probationary term; she subsequently left the school and so did not take part in Workshop #2) said that it was extremely difficult to design worksheets for pupils which would strike a balance between reinforcing content and being 'boring'. These participants did say that they believed that anything presented on a computer screen would hold pupils' attention better than paper handouts owing to a culture of almost religious belief in the capabilities of computers amongst pupils.

Participants then examined the Agent Chooser Interface.

They all said that they hated the computer generated voices, and

that they would serve more as an annoyance than a help during programming work. They all preferred the natural voices (recorded samples of human speech), and would favour a natural male voice.

They all said that while they wanted a natural voice they did not want a realistic face for an agent (e.g. Agent Chooser Interface personality B). All wanted personality G. One participant explained that she felt a realistic male face could be felt to be threatening by female end-users; especially as computer expertise is stereotyped as a male sinecure.

Whether male participants would have favoured a female voice, and whether that should have had a natural / non-natural face is, obviously, unknown. However, owing to the fact that there are almost no male teachers in primary education, if this project was to develop into a full-blown interface for primary teachers the question seems redundant.

Participants then moved on to Prototype #1.

All participants expressed an extreme sense of shock:

- At an animated virtual personality that ‘spoke’ to them in terms they understood.
- That it was so easy to rapidly develop (or “knock together” in the words of one participant) a program template using Prototype #1.

- That it was as impossible with the standard Runtime Revolution interface as they had expected of both interfaces.

One participant asked “If it is so easy to make software for our kids why has nobody told us before?”

Three of the four participants said that they liked the way the agent helped them focus on the questions that the front-end asked them.

One participant said she did not like the agent, but felt that this might be a factor of it being a “bitchy woman”, and that if it were a male agent with a natural voice her attitude might be different.

All the participants said that they would like textual reinforcement of the messages spoken by the agent.

All had a problem navigating through the computer’s directories to find the ‘sunburst’ image to place on the title page of the template program (it was placed in the ‘My Pictures’ subdirectory of the ‘My Documents’ folder on the Microsoft Windows 98 desktop). This is problematic on 2 counts:

1. Non-computer experts should not have to navigate through directories (they should not have to know they exist).
2. Different operating systems have different directory structures. RR/MC will open import boxes in the default folder of whichever operating system it is functioning on.

They expressed great interest in taking part in Workshop #2 and wondered what extra features would be included.

6.2 LESSONS LEARNT

- Remove the female agent (A) with a synthesized voice ('VICKY') and replace it with a male agent (G) and a natural voice ('JOHN').
- Accompany spoken messages with textual reinforcement.
- Put images for Workshop #2 in the default folder of each computer operating system.

7 ANALYSIS OF RESPONSE TO WORKSHOP #2

7.1 THE WORKSHOP

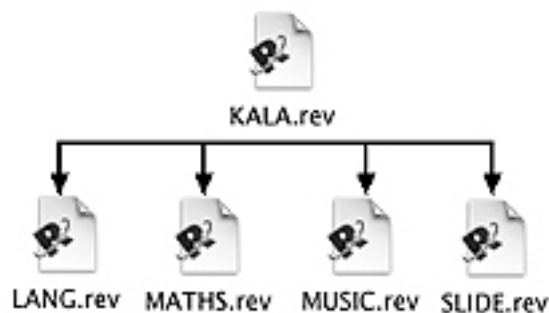
The participants were 3 of the 4 primary-level teachers who had taken part in Workshop #1: the other participant in Workshop #1 no longer working at the school.

Workshop #2 took place in the same room dedicated for computer use used for Workshop #1. Each participant had access to a computer

running Microsoft Windows 2000. An iconic link to Prototype #2.2 was available in a directory called “Mathewson” on the C drives of these computers.

A double mouse-click on the iconic link for Prototype #2.2 automatically loaded Runtime revolution with the Software creation interface acting as a ‘wrapper’ on top of it.

Prototype #2.2 was a free-standing xTalk stack (called *KALA.rev*) with a series of embedded substacks representing the components of the feature tier:



Prototype #2.2: feature tier stacks embedded within *KALA.rev*

For the purposes of Prototypes #2.1 and 2.2 the 2 subclasses developed (one subordinate to *LANG.rev* and one to *MUSIC.rev*) were embedded in their respective feature tier stacks.

As subclasses, in most cases, contain a large number of resources, where they would not share them with other subclasses they should be hived off into substacks.

Many subclasses will share resources with others within the same

feature tier group. An example of this is that most of the subclasses planned for the Music Program Builder will share embedded sound (AIFF) files for all the notes found on a piano keyboard.

The worksheet was distributed to the participants, ground rules for the workshop were explained.

All participants read through the worksheet and ‘dived in’ to the task set.

All of the participants expressed pleasure and satisfaction at the smooth and trouble free fashion the interface worked.

- Participants experienced some difficulties locating the folder “Mathewson” containing images for import through the standard import dialogue box. This merely serves to underscore the point made elsewhere in this thesis that tunnelling through directories is neither easy nor intuitive.
- Responses to the point made in the worksheet relating to changing the personality from a female with a computer-generated voice to a male with a naturalistic voice were:
 1. “Voice clear, easy to understand, icon is OK.”
 2. “Easy to relate to - less threatening. Language sounds real, not too computerised. Easy to understand.”

3. "I liked the voice and the icon. The voice was pleasant and not irritating."
- Responses to the addition of written support for what was spoken by the personality were:
 1. "Good, instructions are backed up on screen in-case they are missed aurally."
 2. "I find this reassuring - I do prefer to hear and read at the same time."
 3. "The written message helps reinforce the auditory message."
- The section that asked for ways in which the software creation interface could be improved yielded 3 suggestions for further improvement that were generally agreed on by all the participants. Each of these points is examined here in some detail.
 1. A language recognition program should have some facility for students to hear (as well as read) the target words/structures.

The author showed the participants how to use the Music feature tier to construct a note recognition program. The closed set of the notes on a piano keyboard were contrasted with the open-ended set of both words within a language and of languages.

To pre-supply audio files of words (for all vocabulary in all languages ?) would obviously be impracticable. The only workable

solution to this would be to provide a facility for teachers to import sound files prepared either:

outwith the software creation interface, or

via a module of the interface accessible from any parts of the feature tier that might need it.

The former possibility would involve the teacher with computer specialists recording into a computer.

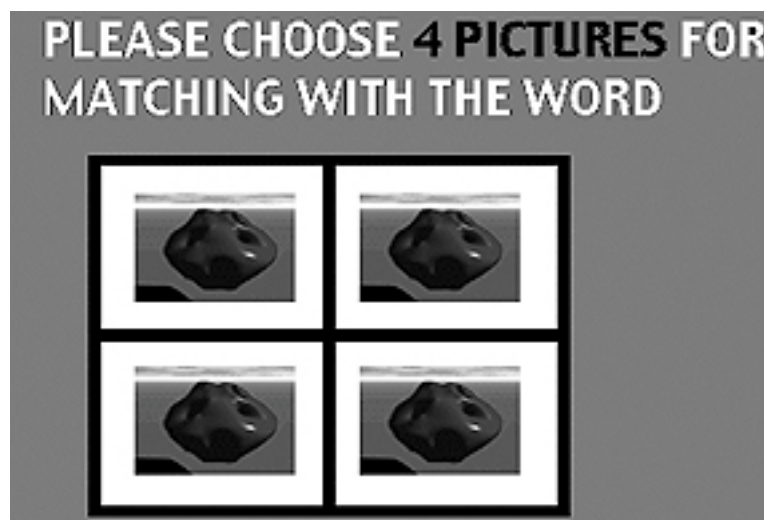
The latter possibility would mean that the requirement to use the WIMP-GUI would be avoided and end-users could record sounds as they needed them rather than rely on a bank of prerecorded files. Both Runtime Revolution and Metacard allow for direct recording and saving in a number of embedded formats via the audio-in facility of a computer. This presupposes Apple Quicktime is preinstalled in the platform on which the software creation interface is being used.

2. Manipulation of font attributes (font, size, colour) by end-users.

This point was precipitated by the fact that the font attributes in text boxes within Prototype #2.1 had been set as 18 point. The font attributes were set on the author's Macintosh computer. When Prototype #2.1 was transferred to the school computers running Microsoft Windows 2000 the font attributes defaulted to 12 point and, therefore, words in text boxes became hard to see.

This is only part of the problem as font attributes in the text boxes belonging to the Software creation interface itself appeared as 12 point as well as those set up in the template program by Prototype #2.2.

3. Difficulty understanding how to use the picture placement buttons in the word-matching section of the language program tier:



One of the participants wrote that the interface required an online

help system only at this point.

Invoking a help system would probably necessitate cluttering up the computer screen with another dialogue box which would, in all probability, have to be non-modal (i.e. movable via drag-and-drop) to avoid possible overlap problems in small screens.

To keep the software creation interface as an Object-Point/Click interface and avoid reverting to some of the problems associated with a WIMP-GUI it would probably be better to redesign the button system.

7.2 LESSONS LEARNT

- Try to find a way in which end-users can import images without having to see or tunnel through the directory structure of the underlying GUI.
- A system whereby end-users can record sounds (via a microphone?) directly into software items they are building.
- See whether it is possible to enforce font attributes as constant cross-platform.

This would, at most, only be possible between Macintosh and Microsoft Windows operating systems: Linux and Unix do not employ true-type fonts.

8 INTERFACE CONSIDERATIONS

PART TWO of *Macintosh Human Interface Guidelines* (Apple

Computer. 1992.) ‘The Interface Elements’ was examined in depth; as were the GNOME and KDE documents for windowing systems on Linux (GNOME. 2004. and KDE. 2004.). To a very large extent all three of these documents concurred. The Apple document was felt to be written using a less jargonised vocabulary.

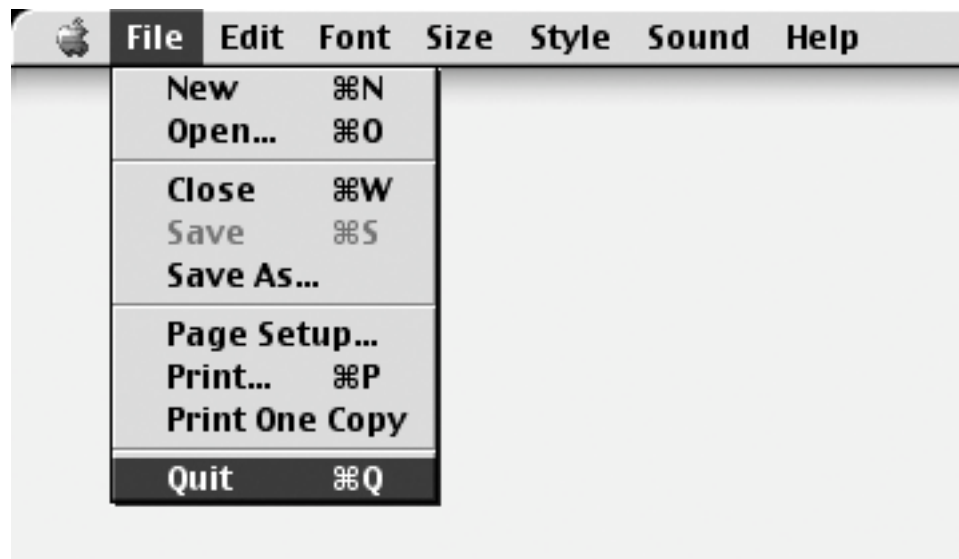
The underlying philosophy of this examination was essentially minimalist in its conception as it was felt that while many of the interface elements of the standard GUI (it, perhaps should be pointed out at this point that the Microsoft GUI (post Windows 3.1) is heavily dependent on the Macintosh interface) were originally intended to facilitate users access many features of their computers’ capabilities relatively easily this was not necessarily relevant for this project because:

- The whole *raison d’être* of the project is to build an interface that restricts users from having direct access to most features of their computers’ capabilities.
- This project is to design a prototype development interface where end-users need to know nothing about either computers or their programming languages.

8.1 MENUS

“In the Macintosh interface, people use a specific syntax for completing actions. First they select an object, then they choose a command to act on that object.”

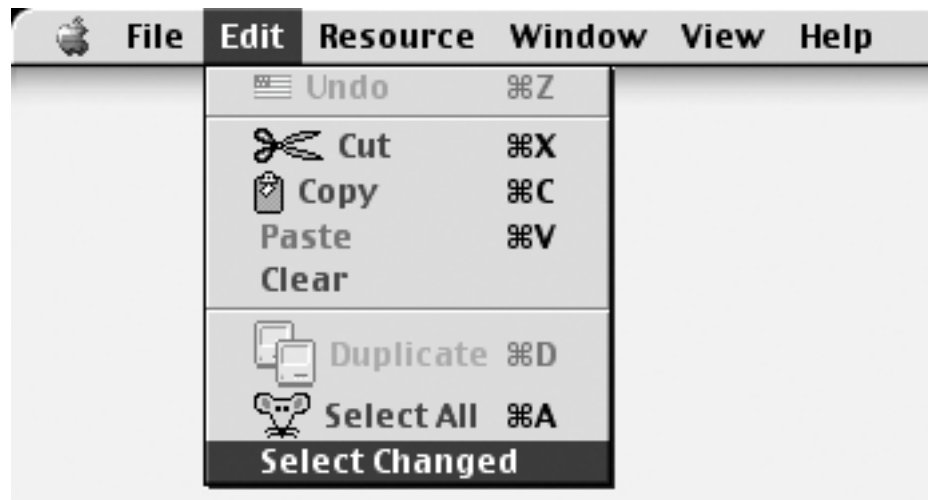
(Apple Computer, 1992. p.50)



Menu driven: Macintosh interface




Menus of this type require a knowledge of computer metalanguage (e.g. 'Copy', 'Paste', 'Save As') that we cannot assume all end-users will know. Nor should they be expected to learn a metalanguage to engage with the development package.

One of the ways that has been used to attempt to ease matters is with the use of iconic menus:



Example of drop-down menu with iconic support.

This, however, still requires end-users to learn the significance of the icons. The theory underlying the use of icons is that end-users recognise semantic connections between the icons and the menu-items.

In the example give above both the  and the  icons are heavily culture-bound, and the  appears to have no obvious relationship with the 'Select All' menu item whatsoever.

All this seems to do is add an extra cognitive load on the end-user.

“Managing cognitive load — the amount of information people can process — is essential to effective teaching or training. Indeed, bombarding learners with too much information at once, called cognitive overload, is one of the chief obstacles to learning.”

(Clark, 1995. p.8)

If educators developing software for content delivery and reinforcement have to take this factor into account there is no very good reason why it should not be taken into account when developing the interface with which those educators are going to develop that software.

In a decision tree driven interface there is not need for these type of menus. Generally, end-users are going to be presented with a restricted number of choices at any one point in the decision tree:



Example of choice screen (Prototype #2)

In the example above end-users are expected to choose the type of EXIT button they want for the program they are developing; the screen presents them with a series of images of all the styles and colours of exit buttons that the interface has on offer. On selecting one choice with a mouse-click the choice is implemented and the user is presented with the next screen in the decision tree.

“Menus are based on the principle of see-and-point. people don’t have to remember command names because they can see all the

options at any time and choose any available option.”

(Apple Computer, 1992. p.52)

It doesn't take too much reflection to realise that this claim is nonsensical for 2 reasons:



Apple menu bar

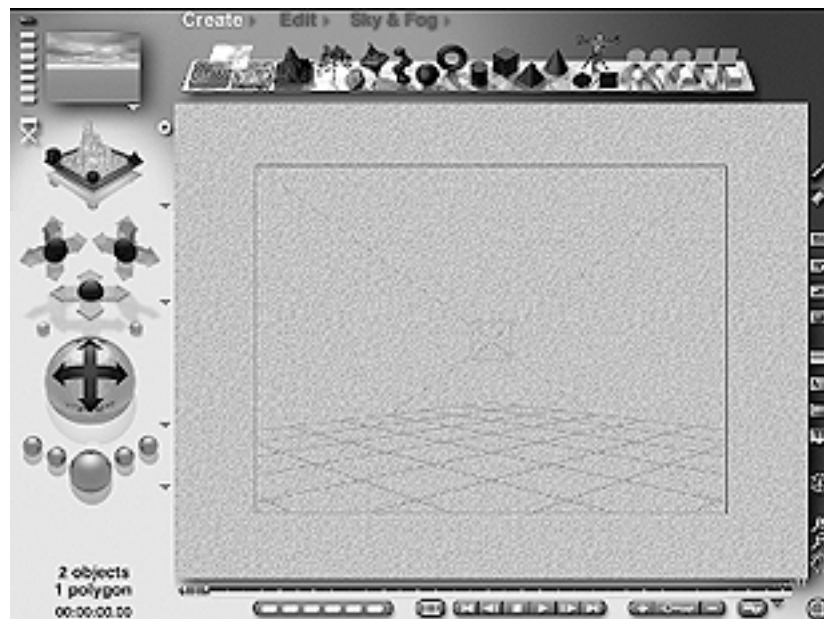
- i. It is not possible to see all the options at one time as options are nested in drop-down menus under each header on the menu bar.
- ii. 'Command names' may not have to be remembered, but menu headings do: if the end-user does not know what 'Paste' stands for he/she will not know whether or not to select it.

“The menu bar should always contain the standard menus . . . the File menu, the Edit menu, the Help menu . . .”

(Apple Computer, 1992. p.52)

This is just a subjective judgement call that presupposes the menu-driven GUI is superior to other types of interface.

Other types of interface have been developed:

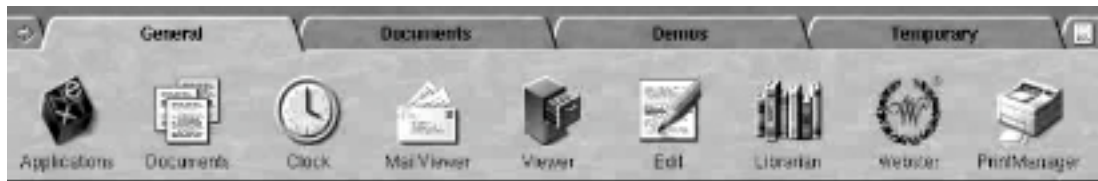


Interface of Bryce (Bryce. 2001)

Most of these, however, also offer a plethora of complex choices, and involve considerable time and effort to become competent in their use. Many of the iconic objects in the Bryce interface offer access to drop-down menus, so are not as easy as appears at first look:



Bryce drop-down menu



NeXTSTEP tabbed 'dock'

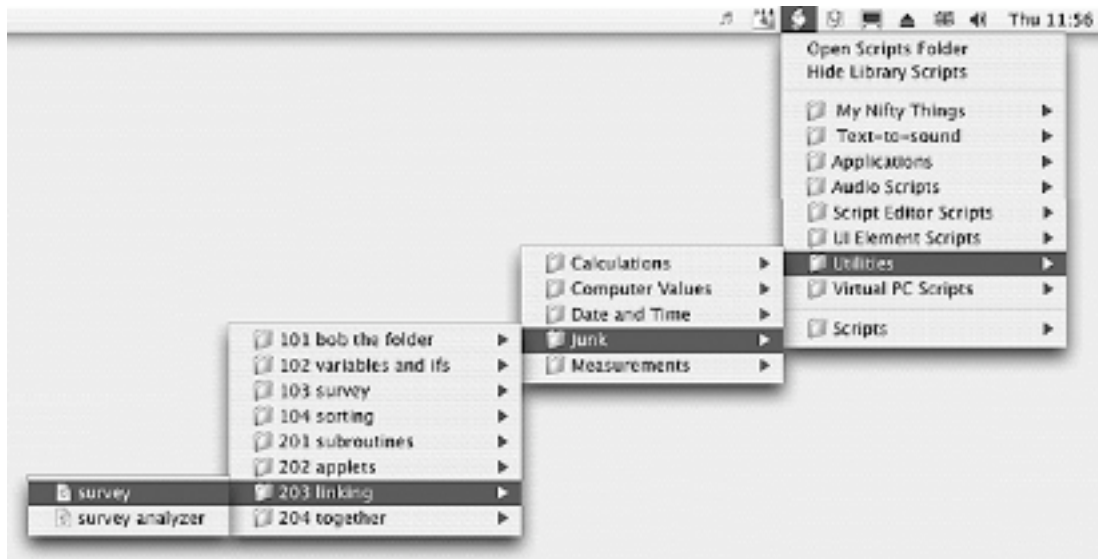
The 'dock' is however, not a system-wide solution insofar as it opens menu-driven program windows.

“Ever since the creation of the WIMP (Windows, Icons, Menus, Pointing Device) concept, menus have played a key role in virtually every graphical user interface for computer systems. These simple structures are easy to use from a user’s perspective, while for a developer they’re a handy method of stashing a lot of features away in an easily accessible interface.”

(Bacon, 2004. p.90)

Arguably, the only reason why menus are ‘easy’ is that we (i.e. regular computer users) have become so inured to them that it is rather difficult to conceive of alternatives.

The very problem with menus is that they contain lots of features stashed away in a generally confusing manner:



“stashing a lot of features away”: Mac OS X

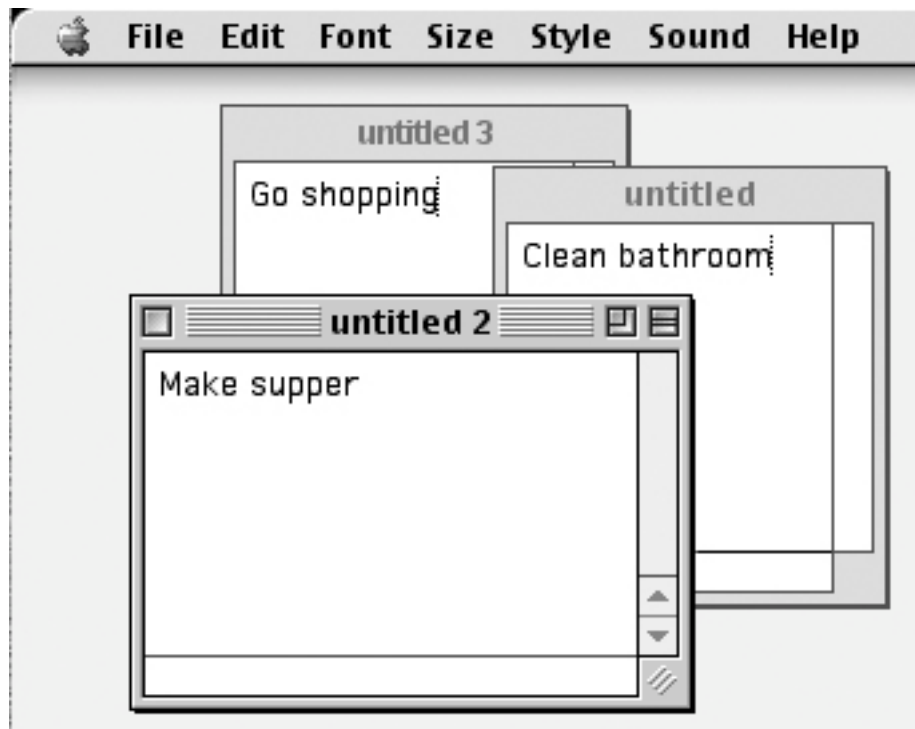
Therefore ‘Menus’ as they have been traditionally understood for the computer GUI have been discarded for this project.

“Don’t offer so much functionality that it confuses the user or harms functionality.” (KDE. 2004 . p.2)

While this project should offer “a lot of features” they can be offered by taking the user through a decision tree *feature* by *feature*, giving time for the user to think about what he/she is choosing as well as space for written explanations of each feature.

8.2 WINDOWS

“Windows provide a way for people to view and interact with their data.” (Apple Computer, 1992. p.132)



Multiple Windows in Mac OS 9

In this project end-users do not interact with their data.

End-users interact with the virtual personality via a non-menu interface. There should be no reason for end-users to access more than one aspect of the software creation process at any one time.

Standard GUI windows can be moved around the screen and resized:

“An SDI-based [Single Document Interface] application doesn't have a mother window that contains all the windows of the application . . they can be moved about the desktop environment as the user wishes. If the application needs to open and close additional windows, they are all opened as children of the root window. They

may logically be children of other windows, in the sense that a dialog window belongs to some document window, but there is no control on the way the dialog window is use or positioned.”

(KDE. 2004. p.5)

None of these capabilities should be needed as each screen / frame presented by the virtual personality to the end-user should not need to be:

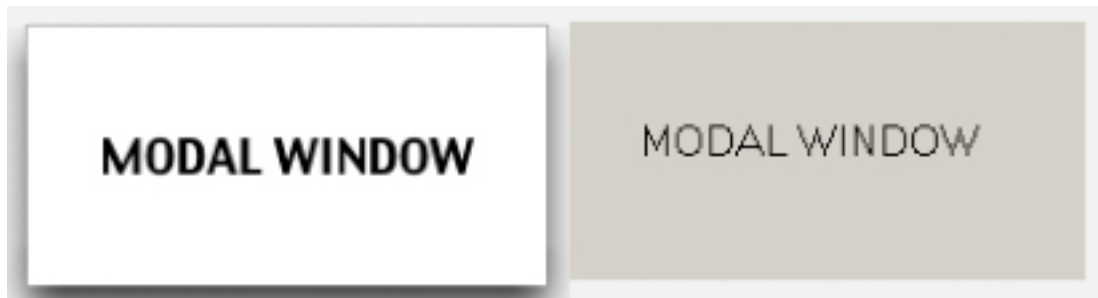
- Moveable: the screen needs to display both the decision tree screen of the virtual personality and the piece of software under creation — therefore the need to be in fixed positions, so that users do not confuse the two.
- Resizable: all the information that the end-user needs to see at any one time will all be displayed directly.
- Daughter windows: all the information that the end-user needs will be displayed within one modal ‘window’.

“Dialog Boxes” (Apple Computer, 1992. pp. 175 - 202).

“Dialog boxes are windows that provide a standard framework in which the computer can present alternatives from which the user can choose. The purpose of dialog boxes is to elicit responses from the user, typically several responses at one time.”

(Apple Computer, 1992. p.176)

The virtual personality will present the user with a series of modal dialogue boxes (rather than windows or modeless dialogue boxes) to force the user to make a choice.



Runtime Revolution Modal Windows

(Mac OS X and Microsoft Windows 98 respectively)

The concept of ‘windows’ as such will not be necessary as there will be no data outwith the modal screen; nor, for that matter, anything to ‘overlap’.

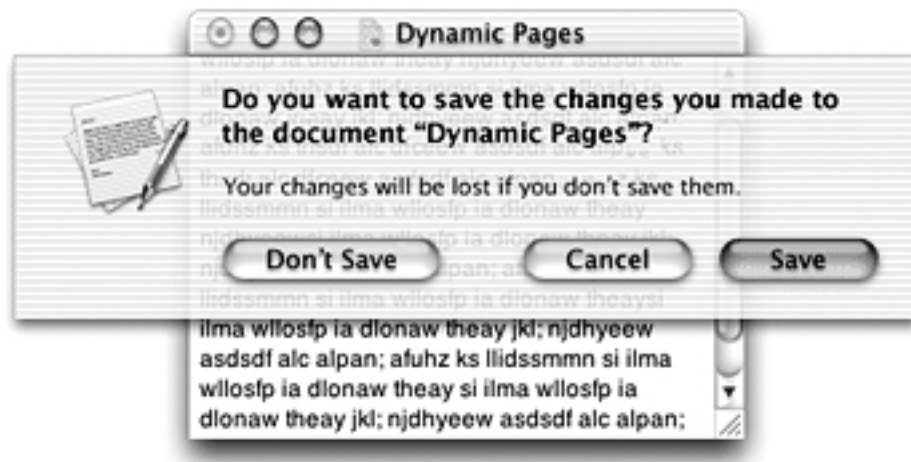
It is interesting to note that with the advent of Mac OS X Apple recommends replacing modal dialogue boxes with ‘sheets’:

“A **sheet** is a modal dialog attached to a particular document or window, ensuring that the user never loses track of which window the dialog applies to.”

(Apple Computer, 2004. [2] p.134)

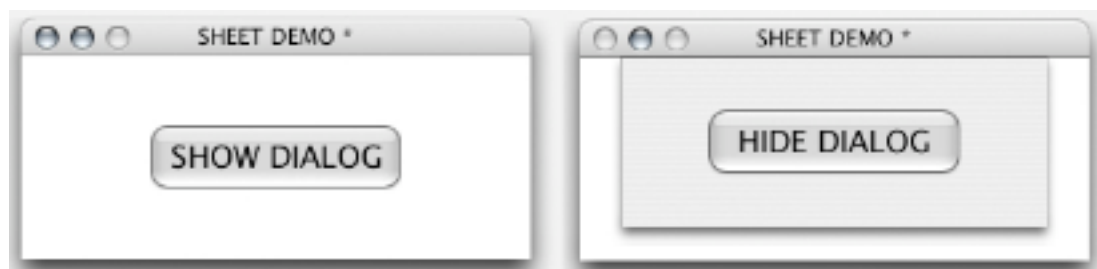
“A sheet is **document modal**—that is while it is open the user is prevented from doing anything else in the window, or document, until the dialog is dismissed.”

(Apple Computer, 2003. “About Sheets”)



Mac OS X Sheet Dialogue Window
(Apple Computer, 2003. “About Sheets”)

Sheet windows have been implemented in recent versions of Runtime Revolution / MetaCard:



Runtime Revolution Sheet dialogue window (Mac OS X)

Sheets are, as yet, confined to the Mac OS X platform.

However, the idea of a ‘sheet’ is something that emerges from a main window: this is not the point of the dialogue boxes to be presented by the virtual personality — they are to be free-standing.

The underlying metaphor of this project is that of a virtual

personality-led dialogue between the virtual personality and the end-user.

As the interface should be as straight forward as possible, most of the modal dialogue boxes will elicit no more than 2 responses at one time, and where 2 responses are elicited they will be closely conceptually related (e.g. 1. Resize a picture, and 2. Choose a coloured border for that picture).

“A modal dialog box cannot be moved or resized.”

(Apple Computer, 1992. p.190)

Alert boxes as such are not necessary as warning and ‘wait’ signals can be presented to the user via manipulation of the ‘visible’ characteristic of image-objects under xTalk.

“In a dialog box, the user can navigate through the interface elements that accept keyboard input, such as text boxes and scrolling lists, in several ways.” (Apple Computer, 1992. p.198)

Navigation will only be possible via mouse-clicks (the right button on a 2 or 3 button mouse). The keyboard should only be used for text entry. This will avoid confusion and the need to memorise multiple key combinations.

There will be no requirement for a SAVE alert box as any software will be automatically be saved at the root level of the user’s computer’s main drive at several intervals during the software

creation process.

8.3 ICONS

This project will attempt to avoid any form of icons beyond a few representations of objects that will be placed by the virtual personality in the software under development:



Start buttons (coloured)

or affect how objects placed in the software will appear:



Title page image sizers

The end-product will, necessarily, be saved on the user's disk drive with the platform-specific icon determined by whether the project is being used as a front end to Metacard or Runtime Revolution.

While the project floats on top of a standard GUI that is unavoidable:



Various Metacard and Runtime Revolution icons

8.4 COLOUR

“When designing interfaces to provide color in your application, avoid using the engineer or hardware model of color. Ideally, you

want to translate what you know into what your users expect. Although its essential for you to understand how the computer produces color so that you can deal with the implications of it, your users operate under a very different model.”

(Apple Computer, 1992. p.261)

With very little alteration this could be the ‘creed’ of the author of this thesis:

End-users will be operating under a model rooted in the world of primary and secondary education, not that of the computer programmer: it is, therefore the author’s intent to develop an interface that acts as a mediator between the end-user and the computer and does *all* the translation (rather than the currently available authoring packages that do a small proportion of it).

“A color specifier is . . . three comma-separated integers between zero and 255, specifying the level of red, green, and blue; or an HTML-style color consisting of a hash mark (#) followed by three hexadecimal numbers . . .”

(DeVoto, J., et al. 2002. [1]. volume 1 p.119)

End-users should not have to learn colour specification numbers, nor wrestle with base 16. They should be able to click on a ‘colour chooser’ button:



clicking on the purple button is preferable to having to type in the following (xTalk) code:

set the backgroundColor of this card to "#AA33BC"

8.5 BEHAVIOUR

“A pointing device makes possible the direct manipulation that is an important aspect of the interface.” (Apple Computer, 1992. p.268)

When this was written direct manipulation was not widespread in interfaces outside the Macintosh interface.

It has subsequently become almost universal.

The only thing in the new interface that should need the use of a keyboard is text entry, and, text entry should only consist of educational content (rather than programming language). All other operations should be ‘reduced’ to single mouse clicks. Therefore all possible operations must be encoded in the interface itself.

8.6 LANGUAGE

“it’s very tempting to use the words that you’re familiar with when you’re developing documentation, training materials, or elements

on the screen. However, it's best to use terms that your *users* are familiar with . . . Don't use technical jargon or computer science terminology.” (Apple Computer, 1992. p.307)

The only faintly technical vocabulary items that have been used in Prototypes #2.1 and 2.2 are 'Navigation', 'Button' and 'Image'. Ideally, even these should be avoided.

As the interface is being built on the premise that end-users know virtually nothing about computers all “technical jargon or computer science terminology” is to be avoided at all costs.

8.7 'WIMP' OR 'OBJECT-POINT/CLICK' ?

The commonly accepted Windows-Icons-Menus-Pointer (WIMP) interface, while, initially developed to facilitate non-specialist computer users has, in itself, developed into something that in many ways is rather cumbersome; and certainly requires quite a bit of 'computer literacy' to use.

An interface that only consists of objects to be selected to navigate users through a series of interlinked decision trees could be developed to greatly simplify computer usage for non-specialists such as educators who wish to develop bespoke software for content delivery and reinforcement.

9 DESCRIPTION OF NEW SOFTWARE CREATION INTERFACE

The Software Creation Interface is specifically designed to block access to the standard 'Desktop' environment by the Macintosh Classic, Macintosh OS X, Microsoft Windows interfaces and the various windowing environments (KDE, GNOME, etc.) provided for LINUX and UNIX.

It denies users access to the file-system (directories and files) of the computer.

It is also designed to prevent end-users having any access to what Goodman terms the 'innards' of the programming environment: i.e. the end-user has no way of accessing any of the underlying coding in the programming environment.

In the Macintosh operating systems (Macintosh Classic and OS X) it is perfectly feasible to adjust settings such that a computer could boot (i.e. start up) directly into the Software Creation Interface. The author presumes this is also possible in other operating systems.

9.1 ASSUMPTIONS

The reason for this is that the target audience of the Software Creation Interface is presumed to have little more knowledge /

experience of computers than an ability to type and point-and-click with a mouse / trackball / trackpad.

It is also to be assumed that as the target audience consists of teachers at Primary and Secondary schools they have neither the time nor inclination to learn the complex syntax and algorithms needed to 'program' computers in the traditional sense of the term.

9.2 THE PROGRAMMING ENVIRONMENT

The author developed the New Software Creation Interface using Runtime Revolution 2.0.1 (RR) and MetaCard 2.5 (MC).

The reasons the author chose these packages were:

- They are completely cross-platform.
- The author has 11 years of programming experience using xTalk, the underlying programming language of Runtime Revolution and MetaCard: he has programmed professionally, using xTalk, for 4 universities, and has also developed a complete educational CD-ROM for Music Theory and Training for a commercial company.
- Because the New Software Creation Interface hides its own 'innards' it should not matter to end-users which package underlies it - therefore the obvious choice was the one with which the author has the most experience.

9.3 SOFTWARE START UP

When the Software Creation Interface starts up it hides the following:

- Macintosh Classic: The Menu Bar and the Desktop.
- Macintosh OS X: The Menu Bar, Dock and Desktop.
- Microsoft Windows: The Task Bar and Desktop.
- GNOME Desktop (LINUX): Dock and Desktop.

The user is presented with a black screen that blocks access to the standard GUI.

For examples of code underpinning what end-user see readers are referred to Appendix 12. [Description is confined to Prototype #2.1]

The first screen of the Software Creation Interface appears at the bottom of the VDU (monitor) where the Virtual Personality (in Prototype #2 this is a male figure called 'Dan') introduces itself via an short animation:



Users are give a choice between starting to build a new software package or quitting.



New Software Creation Interface start screen at bottom of monitor

If the user clicks on the red button “Not just now thanks” the interface defaults to a screen which allows the user either to exit the interface or change their mind. If the user clicks on the green button “Great!” the user is shown the next screen:

SCREEN 2



Screen 2

If the user mouse-clicks on the image of the personality the openCard script runs again so that the user sees the animation and hears the question again.

If the user clicks on the blue button ‘Yes, Please!’ a template is built on the basis of previous setting saved in a series of fields on a hidden card (card “PREFSCARD”).

The user is then shown the card that allows them to move to the Feature Tier:



Feature Tier Chooser card

If there are no stored settings a warning image is shown and on its acceptance reverts to Screen 2.

If the user clicks on the yellow button 'No, Thanks' they are then shown:

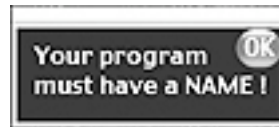
SCREEN 3



Screen 3

There is a standard openCard script statement to provide the opening animation and voice. The openCard script is mirrored in the personality image.

If the user clicks on the green button without having typed a title for their program they are shown a warning message in a modal dialogue window:



On the user mouse-clicking on the 'OK' button the dialogue is dismissed and the cursor defaults to the title entry box in Screen 3.

If the user types a title and then clicks on the green button Screen 4 is displayed:

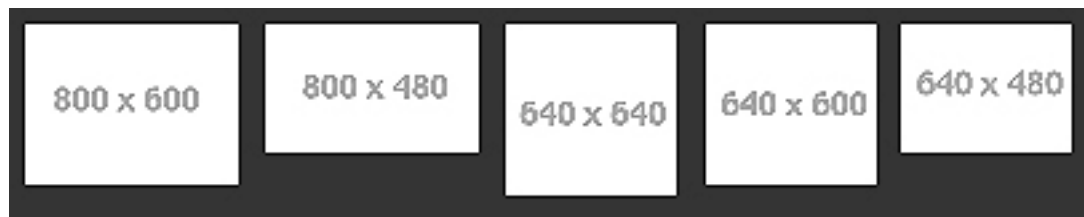
SCREEN 4



Screen 4

There is a standard openCard script statement to provide the opening animation and voice. The openCard script is mirrored in the personality image.

The user is prompted to select a size for his/her new program:



Program Size Buttons

The buttons are scaled relative to one another to give the user an idea of their comparative sizes.

SCREEN 5



Screen 5 with template program in place

As Screen 5 is displayed there is a standard openCard script statement to provide the opening animation and voice. The openCard script is mirrored in the personality image.

The user by seeing what is being made according to his/her choices should feel that he/she is involved in the creation process.

The user is prompted to chose a basic colour for his/her program. This is done by a bank of square buttons that are, in each case, the exact colour to be chosen.

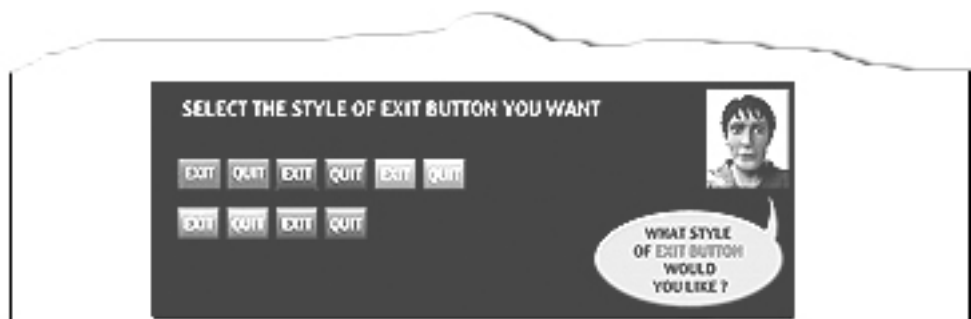
Screen 5 features a 'next' button to allow the end-user to change his/her mind about the general colour of the template program:



'Next' button

This button opens Screen 6.

SCREEN 6



Screen 6

As Screen 6 is displayed there is a standard openCard script statement to provide the opening animation and voice. The openCard script is mirrored in the personality image.

The user is prompted to choose an 'Exit' button for his/her program.

SCREEN 7



Screen 7

As Screen 7 is displayed there is a standard openCard script statement to provide the opening animation and voice. The openCard script is mirrored in the personality image.

The user is prompted to choose a position for his/her Exit button.

The user is also given the option to change his/her chosen Exit button: if this button is chosen the current exit button is removed and the user returned to Screen 6.

The Exit button is moved to the position chosen.

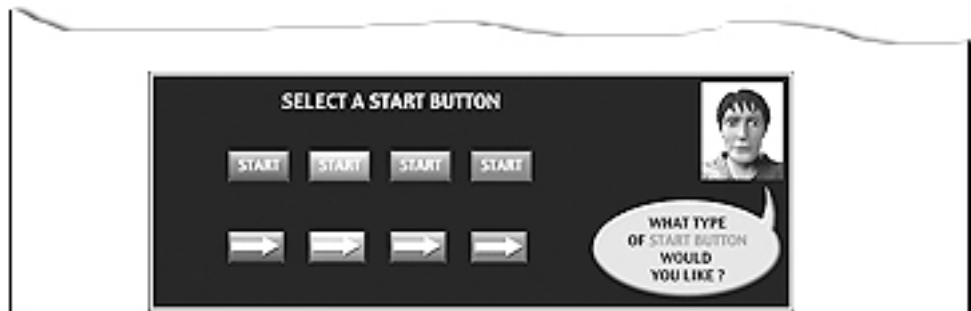
Screen 7 features a 'next' button to allow the end-user to experiment with Exit button positions.

The 'next' button shows the user Screen 8.

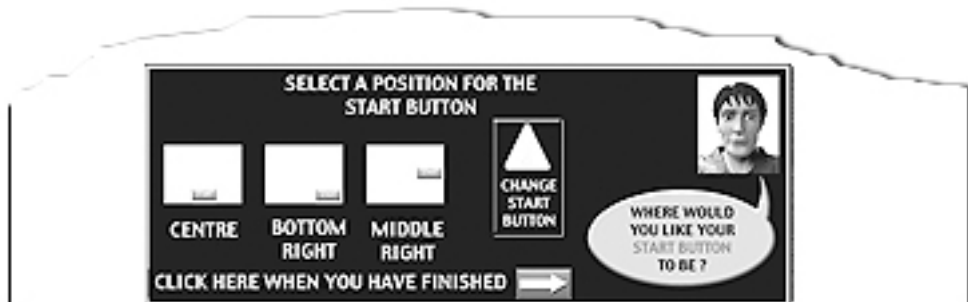
SCREENS 8 AND 9

Screens 8 and 9 both have standard openCard script statements to provide opening animations and voices. The openCard scripts are mirrored in the personality image.

Screens 8 and 9 allow users to choose and place a 'Start' button on exactly the same principle as Screens 6 and 7 are used to place an 'Exit' button. Similarly to the hidden 'Exit' buttons on Screen 6, Screen 8 contains hidden 'Start' buttons:



Screen 8

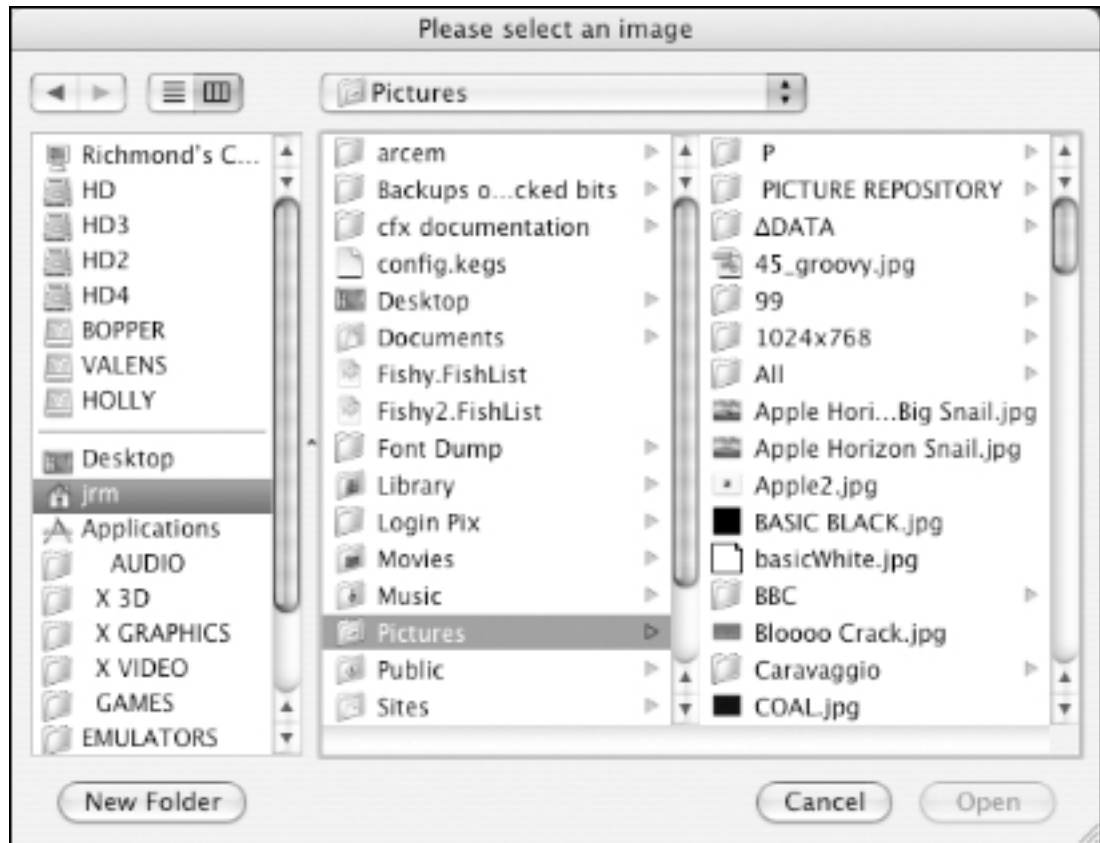


Screen 9

SCREEN 10

Screen 10 offers the user the opportunity to import and place an image on the title screen of the template program.

Unfortunately, this involves using a standard directory navigation box from the underlying GUI:



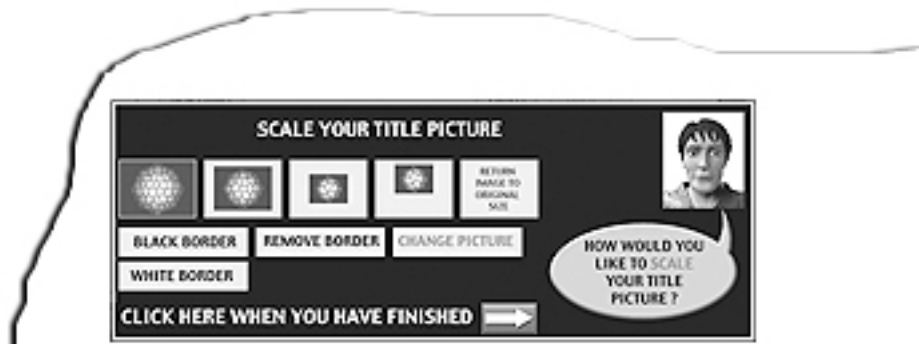
Navigation box (Macintosh OS X 10.3.6)

On an image being selected the user is shown Screen 11 (in the event of the user opting not to have a title picture he/she is taken to Screen 12 directly).

SCREEN 11

As Screen 11 is displayed there is a standard openCard script statement to provide the opening animation and voice. The openCard

script is mirrored in the personality image.



Screen 11

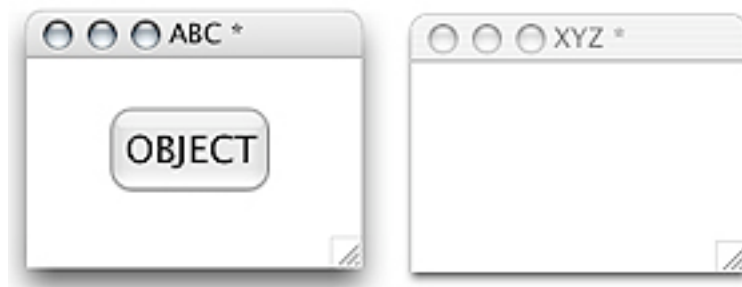
Screen 11 allows the end-user to choose a scale for the image, whether to have a black or white border (or no border) or to change the picture for a different one.

The Screens that copy and modify objects from the Software Creation Interface to the template program involve aspects of the RR/MC dialect(s) of xTalk; 'Transcript' and 'Metatalk' respectively (as of November 2004 they are identical) that have been developed after the demise of HyperCard and, if implemented at all in other xTalk environments (e.g. SuperCard) are implemented differently.

RR and MC work on the 'stack' metaphor common to the xTalk family; however, they allow stacks to have 'daughter' stacks embedded within them, and allow the user to have multiple stacks open at one time.

HyperCard could have more than one stack open at one time; but objects in one stack could not affect the other.

To manipulate objects in one stack via a script in an object in another stack involves very specific syntax ensuring the path from the script in the first stack to the object in the second is explicit. For example, let us consider 2 stacks “ABC” and “XYZ”:



If our objective is to copy the button “OBJECT” from the first card (screen) from ABC to XYZ when the user clicks on the button we will use the following script:

```
on mouseDown
  copy button "OBJECT" of card "FRONT" of stack "ABC"      §1
  to card "TITLE"  stack "XYZ"                              §2
end mouseDown
```

§1 and §2 are one line of code that indicate exactly where the object to be copied resides and exactly where it has to be copied to.

SCREEN 12 TO 14

Screen 12 prompts the user for the number of screens / pages (apart from the title page) required for the template program.

Screen 13 prompts the user for a page numbering style and copies hidden images across to the template program.

Screen 14 prompts the user for a navigation button style and copies hidden images across to the template program.

Screen 15 prompts the user for the start of the Feature Tier.

Up to Screen 14 the user has been compiling their basic template.

This is where the user is enabled to develop a program to suit their needs.

SCREEN 15



Screen 15: Choosing the type of program to develop

9.4 THE FEATURE TIER

On selecting a program type on Screen 15 the template program is saved in the home directory of the user's computer.

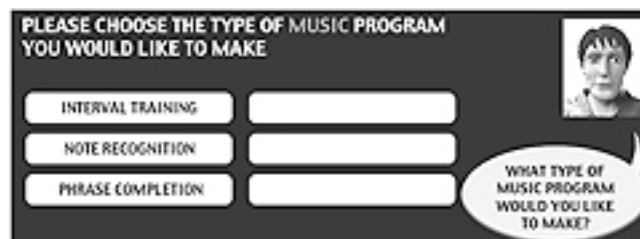
The Feature Tier consists of 4 embedded stacks:



The Music Program Generator: Welcome Screen

In Prototype #2 only a single feature in the Music Program and the Language Program options has been implemented.

The underlying principles of decision trees and copying and manipulation of pre-packaged objects continues throughout the feature tier:



The Music Program Generator: Screen 2

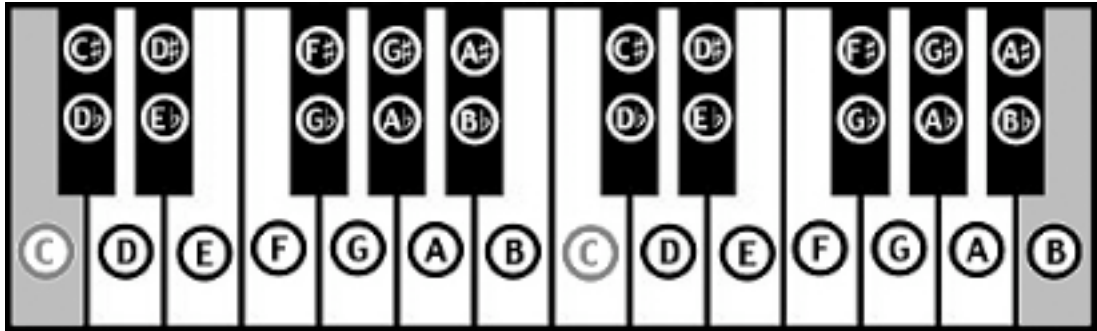


The Music Program Generator: Screen 3

In the case of the Music Program generator it is required to copy embedded audio interface format (AIFF) sounds across to the

template program.

Some objects consist of groups of objects that are quite complex:



Keyboard Object Group consisting of 32 button and image objects that each contain individualised scripts.

9.5 CONCLUDING REMARKS

This Software Creation Interface is based on an ‘Object-Point/Click’ only principle avoiding the need for end users to learn:

- Menu navigation
- Obscure keyboard key combinations
- Directory navigation
- Tool bar icons
- Any form of programming language

9.6 WHAT IS MISSING

Ideally this prototype should have been able to save the finished

products as 'stand-alones'. As it stands Prototype #2 only saves products as documents dependent on either RR/MC or an xTalk player to run.

The feature tier stacks do not have preference-saving facilities implemented.

Prototype #2 is only able to save one set of preferences (those of the last user). Ideally the Software Creation Interface should be implemented as a multi-user system where users can call up their preferences via a password.

To implement multiple preferences without resorting to storing preferences in external files (e.g. tab-delimited text files) might prove complex as it would involve the Software Creation Interface having to modify itself significantly more than simply sending settings to a series of text fields on a preferences card.

The reason why the author did not arrange for the Software Creation Interface to export preferences to an external file is that it would be entirely possible for that file to be lost or corrupted.

The Software Creation Interface would have to generate a new card within itself named after each user duplicating the field structure of a template preferences card.

Where scripts within the Software Creation Interface refer to values stored in a single preferences card at present they would

have to keep track of the specific user and refer to that user's specific preference card.

One of the disadvantages of building up a series of preference cards in the Software Creation Interface is that when RR/MC loads it would become increasingly memory hungry.

10 CONCLUSION

10.1 CONCLUDING REMARKS

Educators are hampered in fulfilling their desires to develop bespoke software for content delivery and reinforcement by the requirement that they have to have computer-programming skills as a prerequisite.

Currently available ‘wizard’ interfaces attached to software authoring packages go a small way to solving that problem. However there is a tendency for these interfaces to use ‘computer-speak’ terms to explain themselves; this acts as another barrier to non-specialist empowerment.

The current dependency on the WIMP-GUI means that all available software authoring packages adhere to an in-house style to integrate the package interfaces into the general methodology of WIMP-GUIs. Participants in the two workshops said that they found

menu-driven interfaces extremely confusing.

It should not be necessary for specialists in non-computer based disciplines to be computer literate. It should, however be possible for those specialists to rapidly develop software items relating to conveying information about their specialist disciplines without that computer literacy.

This thesis presents one viable route to a solution to this problem by abandoning the WIMP-GUI for a highly simplified interface that requires little or no effort to learn how to use.

This was supported when the participants in the workshops were empowered to rapidly develop simple software items without any ‘prepping’ or computer programming knowledge.

A decision-tree structure for a software creation interface removes the need for end-users to tunnel through a maze of choices embedded in often complex and nested drop-down menus.

Functionality is not sacrificed, but on the principle of progressive disclosure it guarantees it, but not at the price of cognitive overload and end-user fatigue.

The inclusion of a virtual personality enhances the interface’s immediacy to end-users and helps to build their self-confidence. The information presented by the personality in an audible form is double-encoded with textual support.

10.2 FUTURE WORK

The final prototype developed alongside this thesis (Prototype #2.2) should in no way be seen as a finished work. Short-term modifications are discussed at the end of chapters 7 and 9. Ideally, the author feels that the principle demonstrated with the prototypes should be extended to replace current windowing -interface environments used on personal computers. If this is to happen then the new GUI would have to become more than just a software creation tool and be capable of presenting to end-users all the facilities presently afforded by current WIMP-GUIs, without compromising the simplicity of use offered to the end-user by the current prototype.

A school computer could have a start screen featuring only 2 buttons:



The 'TEACHERS' button would launch a Software Creation Interface. The 'PUPILS' button would launch into a screen containing iconic links to the various pieces of software prepared by the teachers (as well as 'bread-and-butter' software such as a simple word-processor). No end-user would ever have to cope with the WIMP-GUI.

While it is possible to create a complete Object-Point/Click GUI using xTalk, by the nature of xTalk it would probably always have to overlay a WIMP-GUI. This is extremely inefficient as the computer

has to ‘work’ very hard to provide windowing services and so forth for a GUI that should be never seen.

It should be possible to provide a similar GUI that can either remove the need for a WIMP-GUI underpinning the GUI proposed; or, once started would shut down the WIMP-GUI and free memory resources that would otherwise be performing redundant activities. This would probably have to be developed on a platform-specific basis (PC, Macintosh, RISC, etc.).

There is no real reason why xTalk should be adhered to beyond its relative simplicity for developers to modify to provide bespoke software creation interfaces to suit differing client needs. As Metacard has been brought-out by Runtime Revolution and the other xTalk and xTalk-like interfaces are largely in commercial hands this could stand in the way of a high educational uptake, owing to pricing.

Which ever route was taken a considerable amount of man-hours would be involved; with, towards final roll-out, an ever closer consultation with educators and pupils/students (end-users).

11 BIBLIOGRAPHY

11.1 LITERATURE

Apple Computer. 2003. *Apple Developer Center Documentation*. [Download]
Available from Apple Computers on the World Wide Web at :
<http://www.apple.com>.

Apple Computer. 2004. [1] *Apple Software Design Guidelines*. [pdf] Available
from Apple Computers on the World Wide Web at :
<http://www.apple.com>.

Apple Computer. 1990. *HyperCard Basics*. Cupertino: Apple Computer, Inc.

Apple Computer. 1992. *Macintosh Human Interface Guidelines*. Reading,
Mass.: Addison-Wesley.

Apple Computer. 2004. [2] *Macintosh Human Interface Guidelines*. [pdf]
Available from Apple Computers on the World Wide Web at :
<http://www.apple.com>.

Bacon, J. 2004. KDE Development. In: *Linux Format* [LXF60]. Bath: Future
Publishing. 2004, pp.90-93.

Beach, J., Minton, S. and Ticea, S. nd. *Trainability: Developing a
responsive learning system*. [pdf] Available from the World Wide
Web at : [http://www.isi.edu/info_agents/workshops
/ijcai03/papers/Ticea-ijcai_03_paper1.pdf](http://www.isi.edu/info_agents/workshops/ijcai03/papers/Ticea-ijcai_03_paper1.pdf).

Brenner, W., Zarnekow, R. and Wittig, H. 1998. *Intelligent Software Agents*.

Berlin: Springer.

Caglayan, A. and Harrison, C. 1997. *Agent Sourcebook*. New York: John Wiley.

Candygrammar. 2004. *candygrammar*. [html] Available from the World Wide Web at : <http://catb.org/~esr/jargon/html/C/candygrammar.html>

Clark, R. 1995. *Taking The Plunge*. San Francisco; Macromedia, Inc.

Constantine, L. and Lockwood, L. 1999. *Software for use*. Reading, Mass.: Addison-Wesley.

Curtis, B. 1987. *Human Factors in Software Development*. Washington: IEEE Computer Society Press / North-Holland.

Cypher, A. 1991. [1] *EAGER: PROGRAMMING REPETITIVE TASKS BY EXAMPLE*. [html]. Available from the World Wide Web at <http://www.acypher.com/Publications/CHI91/EagerCHI.html>

Cypher, A. 1991. [2] *Eager: Programming Repetitive Tasks by Demonstration*. [html]. Available from the World Wide Web at : <http://www.acypher.com/wwid/>

Cypher, A. 2004. [3] *Eager*. [html]. Available from the World Wide Web at : <http://www.acypher.com/Eager/>

DeVoto, J., et al. 2002. [1] *Revolution 1.1.1. User Guide*. Edinburgh: Runtime

Revolution.

DeVoto, J., et al. 2002. [2] *Revolution 1.1.1. Transcript Dictionary*.

Edinburgh: Runtime Revolution.

DeVoto, J., et al. 2004. *Revolution 2.2 Online Help System*.

Eberts, R. 1994. *User Interface Design*. Englewood Cliffs: Prentice-Hall.

Evans, R. 1990. "Expert Systems and Hypercard." *Byte*. January. pp.317-324.

Foddy, W. 1993. *Constructing Questions for Interviews and Questionnaires*. Cambridge: Cambridge University Press.

Gaskin, R. 2001. *Fourth World Scripting Style Guide*. San Francisco: Fourth World Media Corporation.

GNOME, 2004. *GNOME Human Interface Guidelines 2.0*. [pdf] Available on the World Wide Web at : <http://developer.gnome.org/projects/gup/hig/>

Goodman, D. 1993. *The Complete Hypercard 2.2 Handbook*. New York: Random House.

Hendler, J. ed. *EXPERT SYSTEMS: THE USER INTERFACE*. 1988. Norwood, NJ: Alex Publishing Corporation.

KDE, 2004. *KDE 3 Styleguide*. [pdf] Available on the World Wide Web at :

<http://developer.kde.org/documentation/standards/kde/style/basics/index.html>

Kommers, P., Grabinger, S. and Dunlap, C. eds. 1996. *Hypermedia Learning Environments*. Mahwah, New Jersey: Erlbaum.

Lieberman, H. 2004. *Programming by Example Homepage*. [html]
Available on the World Wide Web at : <http://lieber.www.media.mit.edu/people/lieber/PBE/>

Mandel, T. 1997. *Elements of User Interface Design*. New York: John Wiley & Sons, Inc.

Master Compact WELCOME GUIDE. 1986. Cambridge: Acorn Computers Limited.

Mayhew, D. 1999. *The Usability Engineering Lifecycle*. San Francisco: Morgan Kaufmann.

Mullet, K. and Darrell, S. 1995. *Designing Visual Interfaces*. Englewood Cliffs: Prentice Hall.

Oppenheim, M. 1992. *Questionnaire Design, Interviewing and Attitude Measurement*. London: Pinter Publishers.

Preece, J., Rogers, Y. and Sharp, H. 2002. *Interaction Design*. New York: John Wiley.

Redmond-Pyle, D. and Moore, A. 1995. *Graphical User Interface Design and Evaluation*. London: Prentice Hall.

Shneiderman, B. 1998. *Designing the User Interface*. Reading, Mass.: Addison-Wesley.

Stoner, G. ed. 1996. *IMPLEMENTING LEARNING TECHNOLOGY*.
Edinburgh: Learning Technology Dissemination Initiative.

Waring, A. 2004, 'Working Towards A Better Environment',
MacFormat, 142, pp. 87 - 91. Bath: Future Publishing Ltd.

Wayner, P. *Agents Unleashed*. 1995. Boston: AP Professional.

Wittgenstein, L. (Trans. Anscombe.).1968, 1995. *Philosophical Investigations*. Blackwells, Oxford.

Wooldridge, M. *Reasoning About Rational Agents*. 2000. Cambridge, Mass.: The MIT Press.

Wooldridge, M. *Multiagent Systems*. 2002. Chichester: John Wiley & Sons Ltd.

11.2 SOFTWARE

Acrux Software. 2001. *iBuild Lite version 1.1.2*. [Download]. Macintosh OS X.
Acrux Software, Inc.

Alloul. 1996. *OpenStack 1.1.1*. [Download]. Macintosh Classic. Harry Alloul.

Apple Computer. 1991. *HyperCard GS v1.1*. [Disk]. Apple IIgs. Apple Computer.

Apple Computer. 1991. *Mac OS 6.0.1 GS*. [Disk]. Apple IIgs. Apple Computer.

Apple Computer. 1998. *HyperCard 2.4.1*. [Disk]. Macintosh Classic. Apple Computer.

Apple Computer. 2001. *Mac OS 9.2.2*. [Disk]. Macintosh Classic. Apple Computer.

Apple Computer. 2004. *Mac OS Classic 9.4.1*. [Disk]. Macintosh OS X. Apple Computer.

Apple Computer. 2004. *Mac OS X v10.3.5. (Panther)*. [Disk]. Macintosh OS X. Apple Computer.

Bet'nct. J. et al. 2001. *WildFire*. [Download]. Macintosh Classic. WildFire Group.

Click2Learn. 2000. *ToolBook II Instructor v8*. [CD-ROM]. Microsoft Windows PC. Click2Learn.com, inc.

Bryce. 2001. *Bryce 5*. [CD-ROM]. Macintosh OS X. Corel Corporation.

Connectix. 2003. *Virtual PC 6*. [CD-ROM]. Macintosh OS X. Connectix Corporation.

F.E. Systems Emulation Technologies. 2000. *Bernie][The Rescue 2.6*.

[Download]. Macintosh Classic. F.E.Systems Horstmann & Gudat.

Gelder, D. 1999. *Serf. 1.0 Beta*. [Download]. Macintosh Classic. Dan Gelder.

IncWell. 1998. *SuperCard 3.6*. [Download]. Macintosh Classic. IncWell DMG, Ltd.

MetaCard Corporation. 2002. *MetaCard 2.5*. [Download]. Macintosh OS X. MetaCard Corporation.

Macromedia. 2004. *Macromedia Director MX 2004*. [CD-ROM]. Macintosh OS X. Macromedia Inc.

Macromedia. 1998. *Authorware 4.0.3*. [Disk]. Macintosh Classic. Macromedia Inc.

Macromedia. 2004. *Authorware 7.0.1. Trial* [Download]. Microsoft Windows PC. Macromedia Inc.

Mathewson. J. 2004. *Attack on the MetaCard Interface*. [Download]. Mathewson. J.

Mathewson. J. 2003. *revtools 6+*. [Download]. Mathewson. J.

Microsoft. 1998. *Windows 98 SE*. [Disk]. Microsoft Windows PC. Microsoft, Inc.

Microsoft. 2004. *WindowsXP_DesignGuidelines.exe*. [Download]. Microsoft, Inc.

NeXTSTEP. nd. *NS Beta Promo Vid*. [mp4]. Available on the World Wide Web at : <http://next.z80.org/next/videos/>

Red Hat. 2004. *Fedora Core 1 Linux*. [Download]. Red Hat, Inc.

Runtime Revolution. 2002. *Revolution 1.1.1*. [Download]. Macintosh Classic. Runtime Revolution Ltd.

Runtime Revolution. 2002. *Revolution 1.1.1*. [Download]. Macintosh OS X. Runtime Revolution Ltd.

Runtime Revolution. 2002. *Revolution 1.1.1*. [Download]. Microsoft Windows PC. Runtime Revolution Ltd.

Runtime Revolution. 2002. *Revolution 1.1.1*. [Download]. Linux PC. Runtime Revolution Ltd.

Runtime Revolution. 2003. *Revolution 2.0.1*. [Download]. Macintosh Classic. Runtime Revolution Ltd.

Runtime Revolution. 2003. *Revolution 2.0.1*. [Download]. Macintosh OS X. Runtime Revolution Ltd.

Runtime Revolution. 2003. *Revolution 2.0.1*. [Download]. Microsoft Windows PC. Runtime Revolution Ltd.

Runtime Revolution. 2003. *Revolution 2.0.1*. [Download]. Linux PC. Runtime Revolution Ltd.

Runtime Revolution. 2004. *Revolution 2.5*. [Download]. Macintosh Classic. Runtime Revolution Ltd.

Runtime Revolution. 2004. *Revolution 2.5*. [Download]. Macintosh OS X. Runtime Revolution Ltd.

Runtime Revolution. 2004. *Revolution 2.5*. [Download]. Microsoft Windows PC. Runtime Revolution Ltd.

Runtime Revolution. 2004. *Revolution 2.5*. [Download]. Linux PC. Runtime Revolution Ltd.

Runtime Revolution. 2004. *Dreamcard 2.5*. [Download]. Macintosh OS X. Runtime Revolution Ltd.

Solutions Etcetera. 2002. *SuperCard 4 Trial*. [Download]. Macintosh OS X. Solutions Etcetera.

TigaByte Software. 2004. *HyperNext 1.31. Developer*. [Download]. Macintosh OS X. TigaByte Software.

Thoughtful Software. 1999. *HyperSense*. [Download]. Thoughtful Software.

Tribeworks. 2001. *iShell 2.5.1*. [Download]. Macintosh Classic. Tribeworks Inc.

Tribeworks. 2003. *iShell 3.0.4*. [Download]. Macintosh OS X. Tribeworks Inc.

Wagner. R. 1989. *Hyperstudio v3.1*. [Disk]. Apple IIgs. Roger Wagner Publishing Inc.

Wagner. R. 2000. *Hyperstudio 3.2*. [Disk]. Macintosh. Roger Wagner Publishing Inc.

X. *clipperx v1.5.2*. [Download]. Platform Independent. Mister X.

X. *colorx v1.5.2*. [Download]. Platform Independent. Mister X.

X. *control_browser_x v1.5.2*. [Download]. Platform Independent. Mister X.

X. *xs_message_box v1.5.2*. [Download]. Platform Independent. Mister X.

X. *Xs Script Editor v1.5.2*. [Download]. Platform Independent. Mister X.

12 NEW SOFTWARE CREATION INTERFACE CODE

SNIPPETS

This appendix consists of xTalk samples from Prototype #2. They support the *what* of chapter 9 with some of the *how*.

This is not intended as an exhaustive code listing, but as some samples illustrative of how certain aspects of the software creation interface's capabilities are effected.

12.1 START UP

[The description here is of Prototype #2.1 that was built into the *revtools* component of Runtime Revolution 2.0.1: it was a comparatively simple operation to sever this link and turn it into a free-standing front-end for either Revolution or Metacard in Prototype #2.2. The reason for keeping Prototype #2.1 embedded was that it speeded up the development process considerably.]

As the interface starts up it sends the following messages to the RR/MC engine:

```
on openStack
    set the vis of stack "revmenubar" to false
    set the vis of stack "revtools" to false      §1
    hide menubar                                §2
    set the backdrop to black                    §3
    put item 3 of the screenRect into WW
    put item 4 of the screenRect into HH          §4
    put (WW / 2) into WMID
    move stack "KALA" to WMID, HH                §5
end openStack
```

§1 RR-specific code to hide the standard items of the RR interface

§2 hides GUI artefacts (Task Bar, Menu Bar,etc.)

§3 hides Desktop and blocks access to files

§4 determines monitor size

§5 places UI at bottom-centre of monitor

12.2 SCREEN 2

As Screen 2 opens the image of the virtual personality animates and asks "Would you like to use your previous settings?":

```

on openCard
    set the lockscreen to true           §1
    play audioClip "PREVSETS"           §2
    play videoClip "2Tm" at 735, 75     §3
    set the lockscreen to false         §4
end openCard

```

§1 locks the screen so that the user is unable to perform mouse-clicks while the animation is playing.

§2 plays the embedded audio interface format (AIFF) file 'PREVSETS'.

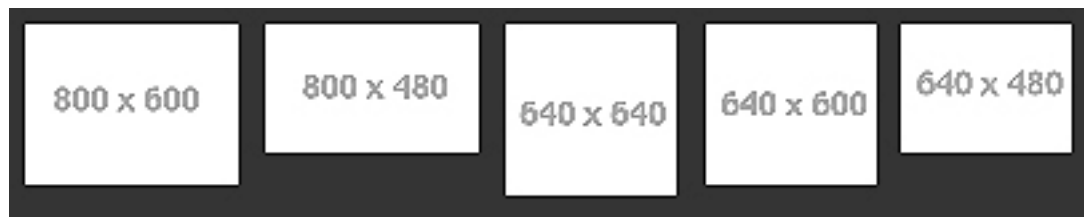
To economise on resources the interface contains separate sound files and personality animations; otherwise there would have to be separate animations with sound for each item spoken by the personality.

§3 plays the embedded QuickTime Movie '2Tm' at the same location as the still image of the virtual personality.

The interface contains a number of animations of different lengths, facial expressions and gestures of the virtual personality that can be paired with sound files.

§4 releases the screen.

12.3 PROGRAM SIZER BUTTONS ON SCREEN 4



Program Size Buttons

When the user mouse-clicks a size button the following script is called (the example used is for the '640 x 480' button):

```
on mouseDown
    beep
    put 640 into fld "fp1" of card "PREFSCARD"
    put 480 into fld "fp2" of card "PREFSCARD"
    do fld "fSTKMAKER" of card "THIRD"
    go next
end mouseDown
```

§1
§2
§3
§4
§5

§1 gives a 'beep' as a form of feedback to the user.

§2 stores the chosen width for the new program in the field where it can be accessed by the script in field 'fSTAKSIZE' (called from field 'fSTKMAKER') and, also, is stored as a preference for future template builds.

§3 stores the chosen height for the new program in the field where it can be accessed by the script in field 'fSTAKSIZE' (called from field 'fSTKMAKER') and, also, is stored as a preference for future template builds.

§4 calls the script in field "fSTKMAKER".

§5 displays Screen 5.

The do fld “fSTKMAKER” statement calls the script in the field “fSTKMAKER”:

```
put fld “fp12” of card “PREFSCARD” into STAKKNAME      §1
create stack                                             §2
set the name of it to STAKKNAME
set the label of stack STAKKNAME to STAKKNAME          §3
do fld “fSTAKSIZE” of card “THIRD”                     §4
```

§1 takes the title of the new program from the storage field where it was stored when the user typed it into the text box on Screen 3 and puts it into a variable called STAKKNAME.

§2 makes a new program template.

§3 names the template with the title stored in STAKKNAME.

§4 calls the script in field “fSTAKSIZE”. The do fld “fSTAKSIZE” statement calls the script in the field “fSTAKSIZE”:

```
put fld “fp1” of card “PREFSCARD” into STAKKWIDTH      §1
put fld “fp2” of card “PREFSCARD” into STAKKHITE       §2
set the width of stack STAKKNAME to STAKKWIDTH         §3
set the height of stack STAKKNAME to STAKKHITE         §4
set the top of stack STAKKNAME to 50                   §5
put STAKKNAME into fld “fSTNOM” of card “BACKCOLOR”    §6
set the resizable of stack STAKKNAME to false          §7
```

§1 gets the chosen width for the template program from its storage field.

§2 gets the chosen height for the template program from its storage field.

§3 sets the width of the new program template to the value taken from the storage field.

§4 sets the height of the new program template to the value taken from the storage field.

§5 moves the program template so that its top edge is 50 pixels below the top of the monitor screen.

§6 stores the title of the template program in a field on Screen 5 where it can be accessed by scripts there.

§7 locks the size of the template program.

12.4 PROGRAM BACKGROUND COLOUR BUTTONS ON SCREEN 5

The script of one of these buttons is:

```
on mouseDown
  put fld "fp12" of card "PREFSCARD" into STNM      §1
  set the backcolor of stack STNM to "#66CC66"      §2
  put "#66CC66" into fld "fp3" of card "PREFSCARD"  §3
end mouseDown
```

§1 retrieves the name of the template program from the storage field and puts into the string variable STNM.

§2 sets the general colour of the template program to the colour

specified by the 6-figure hexadecimal code (light green).

§3 stores the colour choice in the relevant field on the preference screen.

12.5 SELECTING AN EXIT BUTTON ON SCREEN 6

At this point the programming became highly complex.

An image corresponding to the button selected by the user has to be copied to the template program; but, it has to contain a different script to the actual button selected by the user.

The actual button selected by the user contains the script that instructs the copying process:

```
on mouseDown
do fld "fF" of card "EXITCHOICE"           §1
put "EXIT2" into fld "fP4" of card "PREFSCARD" §2
copy img "EXIT2" of card "EXITCHOICE" to card 1 of stack NICK §3
move img "EXIT2" of card 1 of stack NICK to 50,50 §4
put NICK into fld "fSTKID" of card "EXITBPOZ" §5
set the name of img "EXIT2" of card 1 of stack NICK to "XX" §6
set the lockscreen to false §7
set the visible of img "W3" of card "EXITCHOICE" to false §8
go next §9
```

end mouseDown

§2 puts the name of the button image to be copied into the relevant field of the preferences screen.

§3 copies the chosen button image to the template program.

§4 moves the copied button image to a 'parking space' on the template

program screen.

§5 copies the name of the template program across into a field on Screen 7 ready for Exit button positioning.

§6 renames the copied button image to a generic name.

§7 returns screen access to the user.

§8 hides the progress image .

§9 moves to Screen 7.

Field “fF”

| | |
|--|----|
| do fld "fHIDE" of card "EXITCHOICE" | §1 |
| put fld "fP12" of card "PREFSCARD" into NICK | §2 |

§2 retrieves the name of the template program from its relevant field on the preferences screen.

Field “fHIDE”

| | |
|--|----|
| set the visible of img "W3" of card "EXITCHOICE" to true | §1 |
| set the lockscreen to true | §2 |

§1 makes the progress image visible:

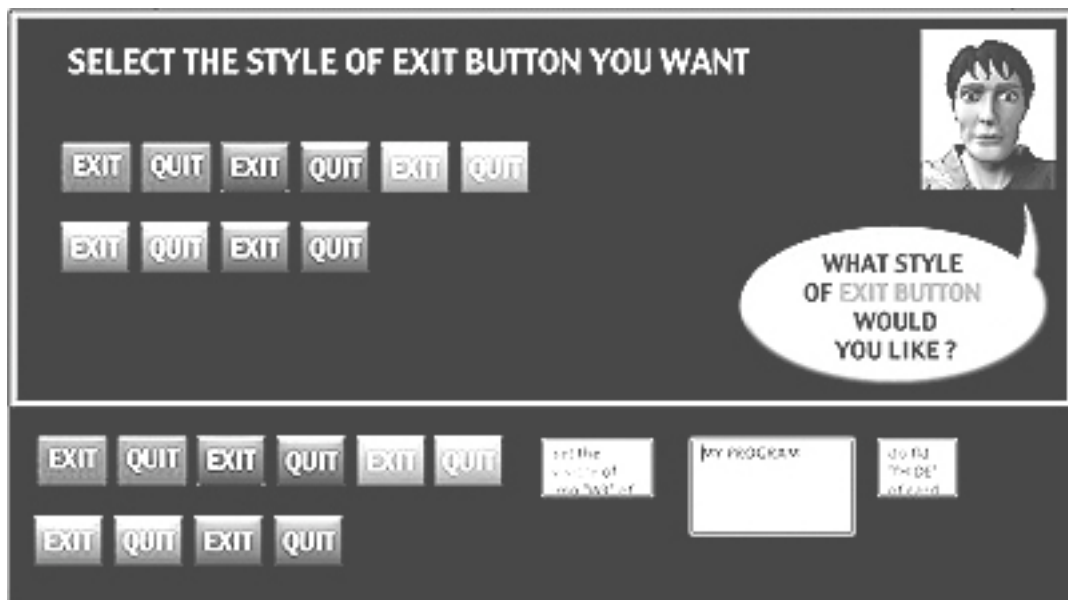


Screen 6: progress image

§2 locks the screen to stop user access whilst the copying process goes on.

The problem of needing similar button images with different scripts was effected by having a secondary series of images hidden on Screen 6 which contained the necessary Exit button code:

```
on mouseDown
    quit
end mouseDown
```



Screen 6 expanded to show hidden button images and fields

12.6 CHOOSING AN EXIT BUTTON POSITION ON SCREEN 7

Choosing a position button for the Exit button results in the execution of a script which moves the Exit button to the position chosen (the example used here is for the left-hand-bottom-corner choice:

```
on mouseDown
  do fld "fLOCKUP" of card "EXITBPOZ" §1
  put "LEFT" into fld "fP5" of card "PREFSCARD" §2
  put height of stack fld "fP12" of card "PREFSCARD" into
  HITE §3
  put HITE - 25 into HHITE §4
  move img "XX" of stack fld "fP12" of card "PREFSCARD" to
  40, HHITE §5
  set the lockscreen to false §6
  set the visible of img "W4" of card "EXITBPOZ" to false §7
end mouseDown
```

§2 sends the chosen preference for the Exit button position to the relevant field on the preference screen.

§3 takes the height of the template program and puts it into the variable HITE.

§4 calculates the vertical position of the Exit button.

§5 moves the Exit button image to its required position.

§6 unlocks the screen, allowing the user control.

§7 hides the progress image.

Field "fLOCKUP"

set the visible of img "W4" of card "EXITBPOZ" to true §1
set the lockscreen to true §2

§1 makes the progress image visible:



Screen 7: progress image

§2 locks the screen to stop user access whilst the copying process goes on.

13 WORKSHEETS FROM WORKSHOP #1

AGENTS: workshop worksheet 1

JOHN R MATHEWSON

When you open Runtime Revolution in the Windows Start Menu you will see a screen with 3 ovoid buttons:



This is a front-end to a modified version of Runtime revolution with a choice of 3 Graphic User Interfaces.

Your goal is to prepare a simple program with a title, an exit button, a basic page colour and a title picture something like this:



1. Please try to accomplish this by clicking on the AGENT button.
Please write down any impressions you have in the box below:



2. Please try to accomplish this by clicking on the STANDARD button.

Please write down any impressions you have in the box below:



AGENTS: workshop worksheet 2

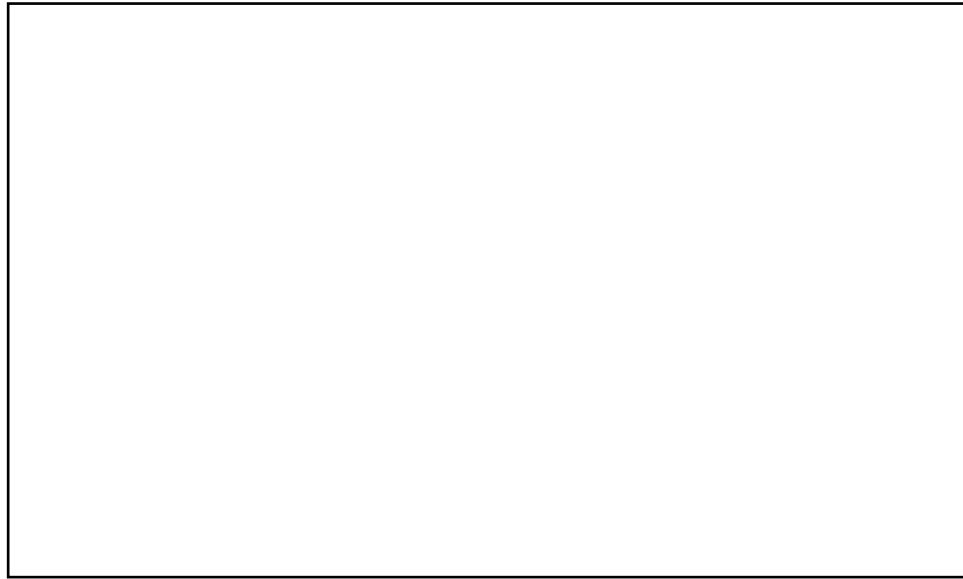
JOHN R MATHEWSON

You will see a program featuring 7 Agents paired with a number of voices (indicated by the blue buttons).

After clicking on the buttons and listening to the agents speaking please enter a maximum of 3 marks (1,2,3) in the grid below to indicate which ones you prefer.

| | A | B | C | D | E | F | G |
|--------|---|---|---|---|---|---|---|
| AGNES | | | | | | | |
| VICKY | | | | | | | |
| SNEZHA | | | | | | | |
| CYBORG | | | | | | | |
| BRUCE | | | | | | | |
| FRED | | | | | | | |
| RALPH | | | | | | | |
| JOHN | | | | | | | |

other observations:



14 WORKSHEET FROM WORKSHOP #2

WORKSHOP #2 JOHN R MATHEWSON

This workshop is an opportunity for you, the participants in the first workshop, to see whether I have successfully implemented the ideas and advice that you gave me during that workshop.

PART 1

First of all I would like you to use the interface to design a language program matching French word to pictures:

The finished program should look something like this:



- The program should be 600 x 600 pixels in size,
- The program should have a light, coloured background,
- The program should have a title page with a picture, [a series of suggested title pictures has been supplied]
- The program should have 2 pages after the title page.

You will find that pictures are stored in the folder 'MATHEWSON' on the 'C' drive of your computer.

To speed things up; here is a list of French words and their English counterparts:-

| | |
|------------------|----------------------------------|
| Poire - Pear | Fraise du bois - Strawberry |
| Banane - Banana | Pomme - Apple Orange - Orange |
| Fromage - Cheese | Tasse de Cafe - Coffee |

[You will find the pictures supplied match these terms exactly]

PART 2

During the first workshop you stated a preference for a 'male' personality over the one supplied:



This female was changed to this male one.



The personality's voice was also changed from a 'female' one generated by a speech synthesiser to a natural, male one recorded by a real person.

Please write what you think about this:

You also stated a desire to have written back ups to what the personality was saying:



Every time the personality speaks his words are supported by a cartoon bubble.

Please write what you think about this:

Please write 3 ways in which you believe the software creation interface can be improved:

15 QUESTIONNAIRE

MATHEWSON'S QUESTIONNAIRE

INTRODUCTORY SECTION

This is a survey designed to ascertain certain aspects of computer use in schools, how comfortable teachers feel using computers for making their own programs, and whether teachers feel they would enjoy and benefit from having access to an innovative type of computer programming interface to help them develop programs for their own to use with their pupils.

- This is an anonymous survey and all responses will be treated in complete confidence.
- You are under no obligation to answer any questions in this survey.

SITUATIONAL QUESTIONS

A.1 How long have you been a teacher ? years

A.2 How old are you ? 25-30 ☐ 31-40 ☐ 41-50 ☐ 50+ ☐

A.3 What is your gender ? Female ☐, Male ☐

A.4 What age pupils do you teach ?

A.5 Is your school:

| | |
|----------------|--------------------------|
| Co-Educational | <input type="checkbox"/> |
| Girls Only | <input type="checkbox"/> |
| Boys Only | <input type="checkbox"/> |

A.6 Is your school:

| | |
|--------------------------|--------------------------|
| State sector | <input type="checkbox"/> |
| Private sector | <input type="checkbox"/> |
| Other (e.g. grant-aided) | <input type="checkbox"/> |

A.7 What is the average number of pupils in the classes you teach ?

A.8 If you use computers what is the average pupil-computer ratio ?

computers to pupils
If you are a Secondary/High School teacher please answer the following :

A.9 What subject do you teach ?

SECTION 0 INDIVIDUAL COMPUTER USAGE

0. Have you ever used a computer ? YES / NO

0.1 What kind of computer have you used ?

0.1.1 ACORN ARCHIMEDES

☐

0.1.2 MICROSOFT WINDOWS PC

☐

0.1.3 APPLE MACINTOSH

☐

0.1.4 OTHER (please name computer used)

☐

0.2 If you use a computer at present what kind of computer do you use ?

0.2.1 MICROSOFT WINDOWS

PC

☐

0.2.2 APPLE MACINTOSH

☐

0.2.3 OTHER (please name computer used)

☐

0.3 Which type of computer are you most comfortable using ?

0.3.1 MICROSOFT WINDOWS PC  ☐

0.3.2 APPLE MACINTOSH  ☐

0.3.3 OTHER (please name computer type) ☐

0.4 Do you enjoy using computers ? YES / NO

(if you answered NO to this question please write a few words on why you don't enjoy using computers)

0.5 Do you have a computer in your home ? YES / NO

(if you answered YES to this question please answer the following questions)

0.5.1 Do you use your computer to connect to the internet ?
YES / NO

0.5.2 If you have children do you allow them access to the internet ?
YES / NO

0.5.3 Do you play computer games ? YES / NO

0.5.4 If you have children do you allow them to play computer games ?

YES / NO

0.5.5 Do you use your home computer for preparing

educational materials for school ? YES / NO

0.5.6 Do you use your computer for listening to music ? YES / NO

0.5.7 Do you use your computer for watching films (such as VCDs or DVDs) ? YES / NO

0.5.8 Do you have a printer at home ? YES / NO

0.5.9 Do you have a scanner at home ? YES / NO

SECTION 1 EDUCATIONAL COMPUTER USAGE

1. Have you ever used a computer program with pupils ? YES / NO

1.1 What type of program was it ?

1.1.1 A word-processing program (e.g. Microsoft Word) ? ☐

1.1.2 A presentation development package
(e.g. Microsoft Power Point) ? ☐

1.1.3 An educational CD-ROM ? ☐

1.1.3.1 A CD-ROM-based encyclopaedia ? ☐

1.1.3.2 A tightly focussed educational package ? ☐

If you answered **YES** to 1.1.3.2 what was the name of the package ?

1.1.4 A topic-specific program developed by a member of staff
at your school ? ☐

1.1.5 If you answered **YES** to 1.1.4 were you the developer ? ☐

1.1.6 A computer game ? ☐

1.2 Why do you use computers with your pupils ?

1.2.1 Because they serve to focus pupils on task/topic ☐

1.2.2 Because I am expected to use computers for a certain
number of hours per week with my pupils ☐

1.2.3 Because I find that using computer programs enhances
pupils' understanding of topics ☐

1.3 Have you ever developed your own computer program
for use with your pupils ? **YES / NO**
If you answered **YES** to this question please complete SECTION 2
If you answered **NO** to this question please answer Question 1.4

1.4 Have you ever developed a computer program for some
other purpose than for use with your pupils ? **YES /NO**

If you answered **YES** to this question please complete SECTION 2
If you answered **NO** to this question please answer Question 1.5

1.5 Would you like to learn to make small topic-specific
computer programs to use with your pupils ? **YES / NO**

If you answered **YES** to question 1.5 please complete SECTION 3

1.6 Do you believe that some computer games have educational value ? YES / NO

1.6.1 Do you believe that some concepts can be taught through the medium of games ? YES / NO

1.6.2 Have you ever used a computer game with pupils for educational purposes ? YES / NO

If you answered YES to this question please write the name of the game in the box below

SECTION 2 PROGRAMMING EXPERIENCE

Please ONLY COMPLETE this section if you answered YES to Question 1.3 or 1.4.

2.1 Which of the following Authoring/Programming packages have you used:

2.1.1 HYPERCARD

☐

2.1.2 HYPERSTUDIO

☐

2.1.3 MACROMEDIA AUTHORWARE

☐

2.1.4 MACROMEDIA DIRECTOR

☐

2.1.5 SUPERCARD

☐

2.1.6 TOOLBOOK

☐

2.1.7 SERF

☐

2.1.8 OMO

☐

2.1.9 iBUILD

☐

2.1.20 iSHELL

☐

2.1.21 VISUAL BASIC

☐

2.1.22 METACARD

☐

2.1.23 RUNTIME REVOLUTION

☐

2.1.24 OTHER

☐

(please give the names of any other packages you have used)

2.2 How did you learn to use the Authoring/Programming packages you have used ?

2.2.1 I used the manual to teach myself.

☐

2.2.2 I dived right in and learnt by trial-and-error.

☐

2.2.3 A friend/colleague helped me to get started.

☐

2.2.4 I attended a course.

☐

2.3 How would you rank your skill level with the Authoring/Programming packages you have used ?

2.3.1 Highly competent

☐

2.3.2 Moderately competent

☐

2.3.3 Just sufficient for my needs

☐

2.3.4 Not competent enough for my needs

☐

2.4 How do you feel about computer manuals ?

2.4.1 They are all the help I need

☐

2.4.2 They are useful for 'need-to-know' reference

☐

2.4.3 I have never used one (because I would rather

☐

find out for myself)

2.4.4 I have never used one (because I find them difficult to understand)

☐

2.4.5 They are a complete waste of paper

☐

2.5 Have you ever made/programmed the following ?

2.5.1 A slide presentation

☐

(if so, with which package ?)

2.5.2 A multiple-choice questionnaire

☐

2.5.3 A mathematics program

☐

2.5.4 A quiz

☐

2.5.5 A crossword

☐

2.5.6 An interactive MultiMedia presentation

☐

2.6 Do you know what **HYPERTEXT** is ?

YES / NO

2.7 Have you ever programmed for a website ?

YES / NO

If you answered **YES** please answer the following questions:

How did you make the website ?

2.7.1 I programmed in HTML code

☐

2.7.2 I used a web-design package

☐

Please tell us the name of the package you used


2.8 Would you like to expand your programming capabilities to help you make small topic-specific computer programs to use with your pupils ? YES / NO

If you answered YES to question 2.8 please complete SECTION 3

SECTION 3 INTERFACE FAMILIARITY

These questions are concerned about how you relate with computers when you work with them. Please take your time answering them.

Questions 3.1.x are concerned with Macintosh HYPERCARD: If you have no experience with Hypercard please go onto question 3.2.

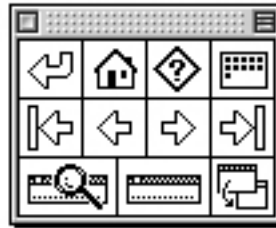
3.1 How many years is it since you last used  HYPERCARD ?

3.1.1 What is this ?



This is . .
.

3.1.2 What is this ?



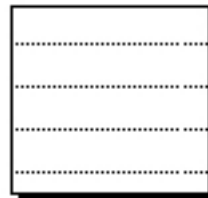
This is . . .

3.1.3 What is this ?



This is . . .

3.1.4 What is this ?



This is . . .

3.1.5 What is HYPERTALK ?

3.1.5 Please draw lines with a pencil / pen from the **red squares** on the **left-hand palette** to what you feel might be equivalents on the **right-hand palette**.



Questions 3.2.x are for all teachers who have some experience with at least one of the packages mentioned in 2.1.x.

3.2 How many years is it since you last worked with one of the programming packages named in 2.1.x ?

3.2.1 Please try to match up the terms from the **lists** on the left and right with icons on the **toolbar** by drawing pencil / pen lines between them.

Label Field

Scrolling List Field

Report Object

Pulldown Menu

Image

Radio Button

Menu Item

Scrollbar

Quicktime Player

Tabbed Button

Reshape Polygon



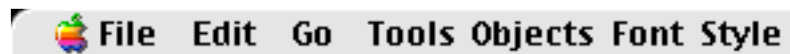
3.3 Which of the following two interfaces do you feel suit your needs most ?

(Please indicate your choice by marking the check box)

3.3.1



3.3.2



3.4 Have you ever seen any of the following ?

(Please mark the check boxes of all you have seen)

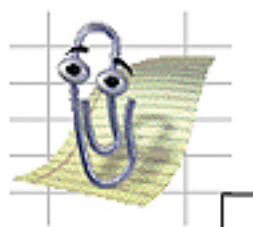
3.4.1



3.4.2



3.4.3



3.4.4



3.4.5

☐

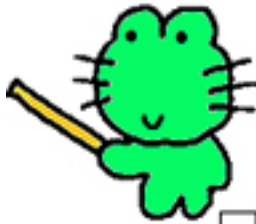
3.4.6

☐

3.4.7

☐

3.4.8

☐

3.4.9

☐

3.4.10

☐

3.4.11

☐

3.4.12

☐

3.4.13

☐

3.4.14

☐

3.4.15

☐

3.4.16 What are these things ?

3.4.17 Where have you seen them ?

SECTION 4

VIRTUAL AGENTS AND YOU

4.1 Have you ever used a Virtual Agent of any of the types in Section 3 in connection with computer work you have done? YES / NO

(If you answered YES to question 4.1 please answer questions 4.1.x, if you answered NO then please answer question 4.2)

4.1.1 Was this with:

- | | | |
|---------|-------------------------------|--------------------------|
| 4.1.1.1 | An Application Helper Agent | <input type="checkbox"/> |
| 4.1.1.2 | An Internet Web Search Helper | <input type="checkbox"/> |
| 4.1.1.3 | A Virtual Secretary | <input type="checkbox"/> |
| 4.1.1.4 | One of the other types | <input type="checkbox"/> |

4.1.2 If you used any of the first 3 types of Agent did you feel that:

- | | | |
|---------|-------------------------------|--------------------------|
| 4.1.2.1 | It helped me considerably | <input type="checkbox"/> |
| 4.1.2.2 | It was moderately helpful | <input type="checkbox"/> |
| 4.1.2.3 | It did not help me at all | <input type="checkbox"/> |
| 4.1.2.4 | It distracted me from my work | <input type="checkbox"/> |
| 4.1.2.5 | I hated it | <input type="checkbox"/> |

4.1.3 Do you prefer:

- | | | |
|---------|---|--------------------------|
| 4.1.3.1 | Agents that communicate via text messages | <input type="checkbox"/> |
| 4.1.3.2 | Agents that communicate via some type of speech synthesis | <input type="checkbox"/> |
| 4.1.3.3 | A plain-vanilla interface with a toolbar like the one shown in question 3.3.2 | <input type="checkbox"/> |

4.2 Do you think you would like to work on a computer with some type of Virtual Agent to help you ? YES / NO

16 WORKSHOP CONSENT FORM

CONSENT FORM

I, John Richmond Mathewson wish to conduct a workshop to see how you respond to various computer interfaces especially as it relates to your work as a teacher.

You are under no obligation whatsoever to attend all or part of this workshop. Should you decide to attend you are welcome to leave at any time.

Results are completely anonymous and as such there is no risk of your privacy being compromised.

Results will be tabulated in a grid with each participant in the workshop given an alphabetic identifier, and results recorded as numbers; therefore no connection may be made between you and any results that may be recorded relating to your participation in the workshop.

The results of this workshop are to be used to further my pursuance of an MSc at the University of Abertay relating to educational software programming. Please find a copy of the MSc proposal attached.

I am unaware of any health risks associated with my workshop; however, if you suffer from epilepsy or are currently taking any medication I would ask you not to attend as the flickering of computer screens can sometimes induce fits.

I should be grateful if you could read the above and if you agree to attend part or all of the workshop to sign your consent below and return this sheet to me prior to attending the workshop.

John Richmond Mathewson.

you may contact John Mathewson at any time on 01334 475137 or at richmond@mail.maclaunch.com.

I consent to attend all or part of the workshop and understand all of the above:

signed _____

date:

17 QUESTIONNAIRE CONSENT FORM

CONSENT FORM

I, John Richmond Mathewson wish to conduct a survey relating to your computer experience and use, especially as it relates to you work as a teacher.

You are under no obligation whatsoever to answer any parts or all of this survey. There are no incentives offered for completion of this survey.

The survey forms are completely anonymous and as such there is no risk of your privacy being compromised.

Once a survey form has been completed it will be lodged with me and all its content (i.e. both the text of the survey and your responses) will belong wholly and only to me. Before you return the survey form to me you are entitled to make 1 photocopy for your own reference alone; either prior or after completion by yourself.

As survey forms are completely anonymous any opinions you may express as part of your completion of a survey form will also be anonymous and as such cannot be used in any way to your detriment.

The results of completion of survey forms are to be used to further my pursuance of an MSc at the University of Abertay relating to educational software programming. Please find a copy of the MSc proposal attached.

I should be grateful if you could read the above and if you agree to

complete part or all of the survey form to sign your consent below and return this sheet to me prior to working on the form.

Each survey form is provided in an anonymous envelope. Please place your form in the collection box provided in the school secretary's office. This is to prevent matching of consent forms with survey forms.

John Richmond Mathewson.

you may contact John Mathewson at any time on 01334 475137 or at richmond@mail.maclaunch.com.

I consent to answer all or part of the survey form "MATHEWSON'S QUESTIONNAIRE" and understand all of the above:

signed _____

date:

This consent form will be removed, before passing to Mr Mathewson, in order to maintain anonymity.