MULTIMEDIA AUTHORING SYSTEM

by

Phillip White
Supervisor: Prof. Jane M. Fritz

TR94-083 April 1994

CS4997 Senior Honours Thesis

Faculty of Computer Science
University of New Brunswick
P.O. Box 4400
Fredericton, N.B.
Canada E3B 5A3

Phone: (506) 453-4566
Fax: (506) 453-3566

Senior Honours Thesis

Multimedia Authoring Systems

Phillip White

April 20, 1993

University of New Brunswick
Faculty of Computer Science
Fredericton, New Brunswick
E3B 5A3

Table of Contents

1.0 Abstract

Over the last several years the stakes of state-of-the-art human computer interfaces have risen. New advances in audio and graphical hardware make possible *multimedia* applications. These applications incorporate some combination of graphics, text, hypermedia controls, animation and sound to produce an interface that presents information in a manner that takes advantage of human capabilities such as hearing and color vision. Such applications generally involve sophisticated hardware devices such as graphical terminals, digital audio and audio synthesizer devices, CD-ROM audio players, and video capture/overlay devices. A multimedia application is a piece of software that controls these various devices to coordinate and present information to a user based on the design of the application and input from the user.

This report is about multimedia authoring systems: programs and collections of utilities that allow the creation of multimedia applications. The intent of this report is to determine the features and abilities desirable for a multimedia authoring system (and the multimedia applications created) and to look at several real world examples of such authoring systems. Over the course of researching this report four authoring systems were examined and will be used in this discussion. The systems examined can be considered a sampling of authoring systems available for the DOS, DOS/WINDOWS and OS/2 environments. These systems were:

- **Storyboard Live** from International Business Machines (IBM) for DOS.
- **Audio Visual Connection** from IBM for OS/2.
- **Toolbook** (with **MediaBlitz** multimedia extensions) from Asymetrix Corporation for the Windows 3.1 environment.
- **Linkway** from IBM for DOS.

The authoring systems were "put through their paces" by implementing the same multimedia application under each. The intent was to examine their usability, flexibility and capabilities for exactly the same application. This provided a basis for evaluating the authoring systems against each other and provided actual experience using the authoring systems for determining desirable features.

This report is divided into several section concentrating on various parts of the project and different concepts of multimedia authoring systems. Sections included are:

- An overview of some of the desirable features for and goals of multimedia authoring systems.
- A discussion of the specific multimedia application implemented for this project.
- Discussion and evaluation of each multimedia authoring system used in the project.
- A more detailed examination of some concepts for multimedia authoring systems including overall structuring of applications and hardware independence.
- Summary and conclusions about multimedia authoring systems.

Appendix B contains more information about the application and Appendix C details how to install and run the application under each multimedia authoring environment. Appendices D and E show examples of code used with various authoring systems. Appendix F details some of the specific problems encountered using each authoring system. A glossary of technical terms used in this report is included as Appendix A.

2.0 Authoring Systems (so you want to write a multimedia application)

The first question to ask before tackling the subject of multimedia authoring systems is: what exactly is multimedia anyway? There is no formal definition which exists for the term. However, for the purposes of this project a multimedia application can be considered to be an application which incorporates some combination of:

- Hypermedia controls
- Textual information
- Graphics and / or animation
- Sound (CD-ROM audio, waveform audio, MIDI synthesized music)
- Video (videodisc player, VCR, television)

A side effect of this definition is that it would include most video games produced during the last decade. A multimedia authoring system is an environment which allows a developer to construct application programs which use multimedia features. Having established a working definition of multimedia let us now consider authoring systems specifically.

There are, of course, several approaches that could be used to create a multimedia application. One possibility is to dust off a C compiler (or compiler/interpreter of language of choice), study the control interfaces to the desired hardware devices, decide upon memory organization and file formats and then write a multimedia application. This approach tends to be not only impractical but requires programmers experienced with user interface design and device control to create an effective application.

The other approach is to use a multimedia authoring system. A multimedia authoring system is essentially an application that has been programmed to control various multimedia hardware devices, interface with users and provides some format for data organization. In this respect it is very similar to the "do-it-yourself" multimedia program described earlier. However, as an authoring system it allows users to create "applications". It presents a simplified interface to the operating environment and provides tools for constructing and arranging the various facilities to accomplish a specific goal : the presentation of information. It is essentially a reusable, reconfigurable multimedia application by which other applications can be created.

The advantages of the authoring system are obvious. It is presumably much easier to deal controlling hardware devices under the authoring system rather than programming the devices directly. Authoring systems usually includes support tools for structuring applications under construction and for creating, importing and organizing various type of data for use with the application.

2.1 Desirable features for Authoring Systems

There are any number of features that could be "desirable" for any application. The list presented here is by no means all-inclusive of every possible feature, but based on experience gained with the four authoring systems would seem to be a reasonable starting point:

- An integrated development environment where the tools the authoring system uses are accessible (easily) from within the authoring system. For example, a programmer should not have to exit the authoring system to run a graphic file format conversion utility if it is supplied with the authoring system - it should be accessible from within the authoring system. Desirable tools for the development environment include:
  - Text / graphics editing and conversion tools.
  - Sound sampling / conversions / editing tools.
  - Video capture utilities.
- An authoring mode and a "user" mode where the application under construction can be tested from within the authoring system.
- On-line help assistance for authoring system features (preferably context sensitive).
- Support for a wide range of hardware devices , including (but not limited to):
  - Video devices:
    - Various resolutions of graphic display adapters.
    - Video capture and / or overlay hardware (both still frame and live motion).
  - Audio devices:
    - MIDI music synthesizer playback devices.
    - Digital audio devices.
    - Compact disk audio.
- Assistance in creating the application's user interface.
- Logical association of control elements of an application (e.g. on-screen buttons or controls) with their intended actions during the authoring phase.
- Tutorials and/or examples applications created with the authoring system that are available for examination.
- Compatibility with other tools / applications:
  - By supporting a wide range of file formats for sound, graphics, animation, etc.
  - By being able to invoke and/or control them directly (for example DDE under Windows and named pipes under OS/2).
- Convenient packaging of the final product for distribution (i.e. it is preferable to have the final product integrated into a small number of files rather than 1 file per element of the application).
- Inclusion of some kid of runtime program so the target site does not need to have the entire authoring system.

2.2 Limitations of Multimedia Authoring Systems

Although Multimedia authoring systems can do many things to automate the production of multimedia application they do have their limitations. They can provide the tools necessary to integrate the various hardware devices and assistance in organizing and assembling applications. They cannot, however, compensate for all human factors.

A multimedia authoring system does not automatically make the user an artist or sound engineer nor instill a sense of aesthetics about what "works" and what does not in an application. A person who cannot draw well with pencil and paper is unlikely to be able to draw well with a graphics tablet or mouse. This illustrates the need for some of the support functions needed in an

authoring system: help in designing the "look" of the application and insuring consistency between the various parts. Ideally the environment should be able to assist even the novice user in designing (simple) applications that look and work reasonably well.

Essentially the final application is going to depend on the developer. While simple tasks and elements can be created in an automated fashion by the authoring system, more complex operations are the responsibility of the developer. The more complex the task the greater the responsibility of the developer. For example, the simple task of switching between screens can be totally automated by the authoring system. However, switching between screens and invoking an animation and sound events or rearranging objects on the screen is likely to require some developer intervention to accomplish. An authoring system may be able to give an application a consistent look and feel but cannot guarantee that information is presented in a rational manner or makes any sense.

2.3 Goals of Multimedia Authoring Systems

At first glance the goals of multimedia authoring systems seem very simple: the creation of usable multimedia software. However there are really *two* goals for such systems:

- Provide assistance, integration and tools for creating multimedia applications.
- Facilitate the creation of aesthetically pleasing and usable applications.

These two goals are oriented towards the two distinct "phases" of the authoring system: authoring and use of the application. These goals may be in conflict with each other. An authoring system that is too easy-to-use in that it automates all functions for the sake of a pleasing and consistent application may not have the flexibility to create complex application programs. An authoring systems really needs two levels of performing tasks: an automated, simple, consistent method for creating simple operations and the flexibility to implement a function in a more sophisticated manner when the preprogrammed methods are inadequate.

3.0 Crunchy Pops[1]

As previously stated, a small multimedia application was created with each authoring system. This application formed that basis for a comparison of the authoring systems for implementing a "real world" project. The application involved a presentation of information about a fictional breakfast cereal - "Crunchy Pops".

When actually evaluating the authoring systems it is important to bear in mind the distinction between the application actually created and the capabilities of the authoring system. A poor multimedia application does not necessarily imply a poor authoring system - the developer could also be at fault. Also, the application created for this project was, necessarily, limited in scope. It

---

[1]Crunchy Pops, the Cruncgy Pops box design, Sucrose Industries and the Sucrose Industries logo are copyright © 1993 by Phillip White and may not be reproduced or reused without expressed consent.

was not possible to test every nuance of each authoring system, only to get a general overview of their capabilities.

3.1 Why Crunchy Pops

The actual intention behind implementing the multimedia application under the various authoring systems was to provide a basis for comparing their authoring features. In this light, the actual application was (relatively) unimportant. All the application had to do was make use of the various multimedia and authoring features being tested in some rational manner. It was therefore easier to simply "make up" the facts surrounding this product to fit the desired testing parameters. An entirely fictional presentation was created. All elements of the presentation from box design to the company history were created for the sake of convenience and testing various authoring aspects.

The fictional nature of the application created does not invalidate the test results. The application was approached with the intention of creating a "real-world" like implementation. The complexity and features of the application approach those that would be expected of a similar real-world application. However it should be noted that it was not possible to test *every* feature of the authoring systems. Several of these environments provide enormous flexibility and power in terms of the applications they can create (notable Toolbook). Also the time that could be allotted to this project was limited and was divided between learning the four separate environments and implementing the same application under each. The features tested can be viewed as a subset of the most desirable and useful features for an authoring system.

It is extremely difficult to describe a multimedia application in words effectively (after all that's why the application was created *multimedia* to begin with). However, Appendix B contains an overview of the Crunchy Pops application. For the most accurate demonstration the four implementations of the application accompanying this report should be viewed.

3.2 Tested Features

The tests performed by implementing the Crunchy Pops application fall into two categories: tests of the multimedia authoring systems themselves and tests of the applications created. The tests of the authoring systems were performed during the process of creating the applications. As previously noted it was not possible to test all aspects of every authoring system. The actual features tested can be viewed as a subset of the most desirable features for any authoring system and included such aspects as:

- Flexibility and structuring of the development environment.
- Compatibility (and ease of interface) with the various multimedia hardware devices used, including:
  - Digital Audio capture/playback card (IBM M-Audio Capture/Playback adapter).
  - Compact disk audio playback drive.
  - VGA graphics capability.
- Ability to integrate multimedia elements into the application.

- Sound and graphics file conversion capabilities.
- Included development support tools (e.g. sound/graphics processing applications).
- Ability to easily and logically structure the application.

The tests of the applications created included such aspects as:

- Attractiveness of the interface and visual elements supplied by the interface.
- Response speed of the application.
- Interface of the application with the user.
- Portability to other computers/hardware configurations without [significant] modification.

4.0 Tested Authoring Systems

This section discusses the four authoring systems tested in preparing this report. The notable features of each package are listed and a discussion of the relative strengths/weaknesses of each is presented. This section is intended to be an illustration of how some real-world multimedia authoring system tackle the problems and criteria discussed for such systems in part 2.1 .

Note that it is not possible in a paper of this scope to present an exhaustive summary of each package. The discussion will be limited to a brief overview of each and a discussion of relative strengths/weaknesses. The packages can be divided into two basic groupings as to how they approach creating multimedia applications: the "master" script approach and the object-oriented approach. Storyboard Live and Audio Visual Connection each use the master script approach and will be examined first. Toolbook and Linkway use the object-oriented approach and will be examined last.

4.1 Storyboard Live

Storyboard Live is a multimedia authoring system from IBM for the DOS environment. It is essentially an integrated environment for creating multimedia presentations (as opposed to sophisticated interactive applications). It features a set of tools to assemble presentations composed of graphics screens, limited animation, digital audio, and video.

This package is primarily a oriented for presentations using full page graphics screens. It features a scripting language for controlling the sequence of operations. It includes a graphics editor, a screen capture utility, a limited proprietary animation editor, a sound processing facility and a script editing facility. Most of the features of the package were tested to some extent during the preparation of this report, with the exception of the video capture / overlay abilities. Storyboard Live provides its own graphical user interface which is used for authoring and running applications.

4.11 Strengths

Although not a particularly powerful package in terms of the applications that it is capable of creating Storyboard Live is quite functional in what it does. There is good integration between the various tools of the package. Although they cannot appear on the screen at the same time the drawing editor, animation editor and sound editor can be invoked from the story creation facility (without loosing the current edits being performed on the story).

The graphics editor of Storyboard live is quite powerful. It contains all the essential features for working with bit mapped pictures: line/circle drawing, filled areas, patterns, zoomable editing facility, etc. It includes useful facilities for "cleaning up" existing pictures by altering certain colors in the picture, the ability to rescale portions of a picture and the ability to edit two picture simultaneously and easily exchange data between them. One of the most useful features is the ability to define one color of any picture as "transparent". When used with in a story, screens (or

portions of screens) may be overlaid onto previously displayed screens. These overlays can choose to use the transparent color, in which case the onscreen data "behind" the transparent color show through - the transparent color is not drawn. The graphics editor does include a graph and chart generator although this feature is probably not powerful enough to be of any serious use in a presentation. There are a number of predefined graphics objects (clip art) which may be included in presentations.

The graphics editor can also load and save in seven additional graphics formats other than the default format for the package. Several of these formats (notable bitmaps and, TIFF and GIF formats) are quite common for the exchange of graphical information and thus the package can import and use a wide range of graphics data.

Storyboard is very good at displaying full screen graphics and popping up portions or other graphics over previously displayed screens. It features a number of configurable dissolves between screen to add variety to a presentation. Storyboard also provides a simple screen sequence editor (separate from the script editor) to quickly generate short, presentations consisting of a series of screens.

One of the best aspects of Storyboard Live is the ability to integrate small animations (called *sprites*) into the presentations. Sprites are small graphical picture that appear as overlays on graphics screens (without affecting the picture). They consist of a number of frames arranged into one or more sequences of animation. They are limited to a certain maximum size (width by height) an to a maximum of 64 individual frames. Sprites also have a limitation on the amount of memory that they may occupy. A complex sprite (more colors) uses more memory that a plain sprite and may run out of memory before the 64 frame limit. The sprite editor allows the animation of simple, predefined types of sprites. Text sprites can be made to "fly" out of the distance or rotate. Various simple line and pie graphs can be animated. Sprites are also included which may be used as buttons, but they are generally unattractive.

4.12 Weaknesses

The greatest weakness of this package is that it is so limited in what it can create. A Storyboard Live application is defined by a single "master" script which specifies all activities for the application. It is not designed for large interactive applications. It is most proficient with a simple, linear sequence of screens and events with limited choices and branching. Although the Crunchy Pops applications was a "random access" structured application this was accomplished with the use of many "goto" statements in the scripting language yielding a finished script that is difficult to interpret (as goto's generally tend to make code). There is no facility for accepting complex input (i.e. character strings) and acting on it. Most interactions is limited to simple buttons or single keystrokes.

The script defines the application in this package. However this leads to an "abstracted" feel to development. Creating a script involves entering the commands and the stepping through the story to determine if the commands produce the desired results. Likewise with the use of buttons. The locations of buttons are defined by script statements (although they may be graphically "set"

onto a picture - use the mouse to indicate a position which is then recorded automatically in the script file). Buttons are merely clickable areas of the screen (that is - areas in which a mouse click is significant). Another part of the script (or the picture itself) is responsible for somehow indicating to the user that buttons occur at a particular position. See section 5.21 for a more thorough analysis of this master script approach to authoring.

There are a number of arbitrary limitations in Storyboard Live. There may be a maximum of 10 buttons defined on a screen at any one time (this limitation was not revealed in the documentation - it was determined by trail and error). Memory available for sprites, sounds and scripts is also limited. The Crunchy Pops application, which is by no means a complicated, in-depth application, used approximately 20% of available script memory. Storyboard Live can only use the "low" 640K memory of a PC microcomputer and does not take advantage of expanded memory that might be available. Also the amount of low memory free determines the graphics mode that Storyboard Live can operate in: VGA, EGA or CGA.

Although the graphics editor is powerful it, and indeed the rest of Storyboard Live, is lacking a true text type. Any text typed onto a picture is converted to a bitmapped representation making future editing of the text impossible (without erasing and retyping).

There is relatively little assistance provided for creating a consistent "look" for an application. Artistic expression is almost totally left up to the developer. Some of the clip art objects are useful but many default sprite types that can be created and the graphing tools do not produce attractive output. There are few example applications provided with Storyboard Live and little in the way of tutorial assistance.

Another major weakness of Storyboard Live is the user interface of the package. Most interfacing with the various tools is done through the mouse. The left mouse button initiates an operation and the right mouse button (simultaneously clicked) terminates an operation. Releasing the left mouse button without clicking the right button aborts an action. This is a fairly clumsy (and non-standard by IBM's own SAA definition) way to handle input from a mouse. Also, although there are many menus in the story editor, graphics editor and sprite editor there are absolutely no keyboard shortcut key sequences defined, forcing the developer to always use the mouse (another violation of SAA).

Script input is a slow process under Storyboard Live. All commands are entered using drop-down menus for an onscreen script editing facility. As previously mentioned, some data fields for scripts (for example location of graphical popups and size/position of buttons) can be "set" on a graphical screen using the mouse and that data automatically recorded in the script.

Although it could be a strength, the current implementation of Storyboard Live's sound processing facility is not. This tool allows sound to be recorded from a digital audio capture card and incorporated into the presentation. Several IBM proprietary soundcards are supported as well as Creative Lab's Soundblaster audio card. However the tool is very limited in scope. Sound may be recorded but there is not facility for editing it to remove unwanted sections (pauses at the beginning or end of samples for example). There is no indication of the relatively loudness of the

sample recorded to show where the input sound level of the card could be boosted for better quality of recording. Sounds must be played back on hardware compatible to that which recorded them: for example a Soundblaster sample cannot be played on an IBM M-Audio card (although it may be played over the internal speaker of the PC). The tool did not appear to work very well. In preparing the Crunchy Pops application an IBM M-Audio proprietary sound card did not produce satisfactory results when used with this tool.

A Storyboard Live application is composed of all of the separate source files which are used in it. All of these files remain in one directory, there is no "building" of one application file. Thus, a developer must be careful to distribute all of the pieces of an application.

4.2 Audio Visual Connection

Audio Visual Connection [AVC] is the second authoring system examined which uses the master script approach. It is by IBM for the OS/2 environment. This product is very similar to Storyboard Live in overall capabilities: it is an integrated development environment for creation presentation type applications. It includes an integrated script editor, graphics editor, sound and video utilities.

The AVC authoring system initially starts in a full screen text mode from which a user can run or edit an existing application or create a new application. Upon selecting to edit an application AVC switches to another full text mode screen listing what IBM terms the "bill of materials" for an application. This is a table that lists every script, image, sound, etc. file that is used in the application. All objects used in the application *must* be listed in this table. Each object is listed along with its type (sound, image, script, etc.) By selecting any object the developer can invoke the appropriate editor for that object. This provides the integrated development environment for applications. Although the editing environment is character mode (except for the graphics editor) the final application does run in graphics mode for the video adapter installed.

One of the biggest curiosities about Audio Visual Connection is that while it is for the OS/2 environment it does not take advantage of the Presentation Manager graphical user interface of OS/2. This restricts the video hardware that AVC is compatible with to that hardware which IBM chooses to provide specific drivers for, currently limited to 8514/A and VGA compatible video adapters. Likewise, it does not take advantage of the MMPM/2 environment for controlling multimedia devices[2]. In fact, the two words that best describes AVC would be "inflexible" and "IBM". It is an environment that appears to be aimed as IBM PS/2 computers running OS/2 with IBM audio and IBM video hardware. This is not necessarily a *bad* approach but does not lead to good integration with other environments.

[2]The lack of MMPM/2 support is understandable as AVC was developed before MMPM/2 existed and the versions of AVC tested predates MMPM/2

4.22 Strengths

AVC, as stated, is intended for use with IBM hardware and software and in that light integrates very well. It includes inbuilt support and drivers for various hardware (e.g. IBM M-Audio card and IBM M-Motion and Video capture cards). When used with the M-Audio card for this project the sound editor worked flawlessly (in fact, this was the only environment in which the sound card worked to its full potential).

AVC incorporates a fairly flexible scripting language for controlling an application. The scripting language is approximately equivalent in power to a simple implementation of BASIC and includes statements for looping, file I/O, true character string input and functions as well as the necessary statements for audio/video output control. As was the case in Storyboard Live, scripts are used to define and control the flow of an application. They can interact with a user via the mouse or keyboard and make decisions about how to proceed with the application. Scripts are entered on a full text mode screen that is divided into fields. The developer types commands and values into various fields and can invoke drop down menus which prompt for legal values for the fields. Again, this type of scripting does not lend itself to developing a complex hypermedia application although AVC is considerably more flexible in this regard than Storyboard Live.

The scripting facility integrates fairly well with the other facilities of AVC. For example, in a similar manner to Storyboard Live, when creating a statement to overlay a portion of a graphics picture onto the current screen the developer can invoke a "viewing" facility. AVC will display the source graphic file and destination screen and allow the developer to use the mouse to selection the portion of the source file and position it on the screen. When the operation is completed this information is automatically put into the correct fields of the overlay script command.

It is difficult to classify the scripting language of AVC as strictly a strength or weakness. It is a flexible tool with much greater capability for controlling an application than that of Storyboard Live. As with Storyboard Live, there is a cycle of editing a script and then running it to determine the overall effect of changes. This leads to an "abstracted" feel to the development cycle. It is impossible to determine what an application is by examining the script alone.

Instead of defining clickable areas on the screen with the script as in Storyboard Live, they are defined on a graphic picture. Using the graphics editor a rectangular area may be defined on a picture and given a name. When an application is running and a named area is clicked the system script variable "@tf" (stands for trigger field) is set to the name of the clicked area. This is a definite improvement over the Storyboard Live method of defining areas with separate statements and then explicitly checking each area to see which one was clicked. It neatly encapsulates the concepts of buttons. With careful application of this technique a hypertext-like presentation could be developed with some minimum of effort. The online help application which comes with AVC is an example of this type of hypertext reference. Also, in a similar manner to buttons, true text fields may be defined for a graphics screen. They are displayed with the picture and may be edited later with the graphics editor.

AVC does take advantage of the OS/2 multitasking environment in some ways. It has a facility where by it can communicate through "named pipes" to other programs running one the system. A named pipe is essentially a message system through which one process can write data to the pipe and another process can read data from the pipe. This allows an AVC application the capability of communicating with other programs in the OS/2 environment. For example, clicking an AVC button could cause it to write a message to another program requesting some service that is beyond AVC's capability. A user could type text into an entry field and that text could be sent along a named pipe to another program.

AVC has the capability of calling functions in DLLs (Dynamic Link Libraries). DLLs are generally written in a language such a C or C++ and contain functions. AVC can use this to extend its base capabilities by invoking DLL functions. Note that this is really circumventing the authoring system: using a DLL means that AVC does not have the capability to perform a task itself. Also the creation of a DLL is significantly more difficult than that of a multimedia application under AVC and would probably require an experienced programmer.

There is a fairly extensive (albeit extremely slow) online help facility available from within AVC. Also included are several applications which can be used as tutorials to study the capabilities of the package. The AVC documentation does supply some ideas for using the package but is mainly limited to describing the capabilities and use of the environment.

4.22 Weaknesses

The most glaring flaw of AVC is an extremely poor user interface for the development environment. Ignoring the existence of the Presentation Manager GUI for OS/2, IBM designed AVC with a character mode authoring interface. The initial "main menu" of the system, the "bill of materials" screen for an application, the sound editing and script editing tools of the system all use 80x25 character mode.

AVC does make use of a mouse during the editing phase but again in a strangely limited fashion. The mouse moves a hilite bar up and down on the screen, even on the script editing screen. Clicking and releasing the left mouse button activates the strip menus, which may be navigated again by the mouse. Clicking and releasing on a menu item will invoke that action. This is a very slow and clumsy process. The mouse may be used to cut and paste text in the script editor but the developer must select the "copy" option off of the menu to begin marking an area then click when complete. Standard OS/2 procedure would be to mark an area of text and then select the operation to apply to that area.

The menu strip may alternately be activated by pressing the "F10" key and the arrow keys/mouse used to select an item. Most menu items do provide a shortcut key combination, ALT in conjunction with another key. Even when menu items shortcuts coincide with activities defined for general OS/2 applications the shortcut key sequences are non-standard. For example the standard OS/2 shortcut key sequence (as defined by IBM) for a "Copy" operation is Control-Insert, AVC uses ALT-C. The menu system under AVC is so clumsy that the developer is forced to use the shortcut keys for the sake of efficiency, yet the keys do not adhere to OS/2 standard

definitions. As a side note, AVC does not use the standard OS/2 clipboard and cannot share text and graphics with other applications.

The graphics editor is even worse in terms of a poor interface. The mouse must first be positioned to the desired location for a drawing operation. Clicking the mouse or pressing F10 activates a strip menu from which a drawing operation may be selected. The mouse is then used to perform the operation - clicking the mouse again terminates the drawing operation. Alternately a shortcut key may be used to invoke drawing operations. This is a very frustrating method for editing graphics files and backwards to most similar programs. Despite having all of the necessary features for such a tool the user interface is extremely clumsy and slow for accomplishing work.

While running an application AVC does operate in graphics mode. It provides a mouse cursor when clickable areas are defined on a screen. The mouse cursor changes shape to indicate clickable areas but not very well. The default mouse pointer is a crosshair. When over a section of the picture that can be clicked on the cursor changes to a crosshair in the middle of a square. This change is very difficult to see on the screen.

AVC lack support for any type of animation in its presentation (except for using the script language to successively paste graphical elements to the screen). AVC supports a limited number of graphics formats (IOCA, TGA, RGB and TIFF) along with its own "._IM" format. Interestingly enough it does not support the OS/2 bitmap format for files. AVC does supply both DOS and OS/2 utility programs to "capture" graphics screens from application but these applications were not tested in this project. Storyboard Live was used to save files into AVC format as AVC could not convert from Storyboard Live .PIC format. AVC does not support any sound format other than its own ._AU/._AD format. It was not possible to convert waveform audio files for use with AVC - the sound editor was used to re-record all samples.

AVC insists on using a specific directory structure for its applications. Each application must have a separate directory immediately under the root directory of a disk. For example the Crunchy Pops application was names "POPS" in AVC. AVC created a directory of the name "POPS.!AP" under the root directory to hold the application. Also, all of the separate pieces of an application are used from their original files. No centralized single database is created for the application.

The last significant drawback of AVC is that it is limited to a few very specific IBM hardware multimedia devices which it supports. It supports IBM video capture/playback and audio cards. It also supports generic VGA and 8514/A compatible video adapters as well as several IBM specific video adapters found on IBM PS/2 machines. There is no support for any other types of multimedia devices, including CD-ROM or multimedia hardware devices from any third party.

4.3 Toolbook

Toolbook is a multimedia authoring system from Asymetrix Corporation. It runs under the Windows 3.1 environment (or Windows 3.0 with Multimedia Extensions). This version of Toolbook was used in conjunction with MediaBlitz - a set of extensions for Toolbook to assist in

creating multimedia "clips" (sound, animation and video) and inserting these clips into Toolbook applications.

As stated Toolbook runs under Windows and uses the native Windows GUI for authoring and applications created. Toolbook includes a large number of tools for creating and manipulating buttons, bitmapped graphics, vector graphics[3], text etc.

Toolbook is designed around a book metaphor and is an object-oriented environment. The book metaphor means that a Toolbook application (called a book) is divided into a number of "pages" (separate screens of the book). Generally it is intended that each page of a book convey a set of related information or provide some functionality for the application. Toolbook as an object-oriented environment means that each button, text, picture (even page and the book itself) are treated as distinct entities - objects. Any object may have a series of scripts defined for it which are invoked whenever the object receives a "message" (a request to perform some action). An object's scripts may send messages to other objects. The interaction of objects and messages defines the structure of a Toolbook application. See section 5.22 for further discussion of the object-oriented approach to authoring.

Objects are associated with a page of a book and are displayed on screen whenever that page is displayed. A special page called a "base" page may be shared by one or more pages. Objects contained on a base page appear on every page which shares that base page. There may be any number of base pages in a Toolbook application. A Toolbook application (with the exception of MediaBlitz clip files) is stored as one compact form: there is little chance of "loosing" parts of the application.

Toolbook operates in two modes: author and reader. Author mode allows pages of the book to be formatted, scripts to be modifier, etc. Reader mode is the mode where the application is actually used (the end user uses this mode).

Because of the complexity of the package and the limited time which could be allotted to any one authoring system the capabilities of Toolbook were not fully examined. Indeed the flexibility of Toolbook is such that extremely detailed and complex applications can be created with it. Only a subset of its features were considered: those features which it has in common with the other multimedia authoring packages examined (the "base" or most necessary set of authoring features).

4.31 Strengths

The most important strength of Toolbook is the user-friendly WYSIWYG development environment. Beginning with a blank book, a developer a an easily accessible set of "tools" to create objects of every type. The object are easily positioned and sized on a page of the book. There is no guesswork as to how the final application will appear - the authoring view is exactly the same as the appearance that will be presented to a later user of the system. Many Toolbook objects are designed to look like standard Windows objects (buttons, input fields, lists, etc.).

[3]Lines, circles, filled areas, boxes which are not stored as bitmaps but are drawn are runtime. The may be rescaled without the loss of detail found in bitmapped graphics.

Toolbook can make use of all fonts installed in a Windows system. Therefore a well-designed Toolbook application will maintain the native "look and feel" of Windows. This assists users in using applications: they are familiar with how such controls are used under other Windows applications.

Toolbook offers good integration with the Windows environment, it is able to import various Windows file formats for graphics and use the Windows clipboard to exchange text and graphics with other Windows applications. Using Windows DDE capabilities, a Toolbook application can exchange data with, control or be controlled by other applications running in the Windows environment. Toolbook applications can call functions in DLLs to perform tasks outside the scope of Toolbook's capabilities. The Windows Media Control Interface (MCI) extensions offer Toolbook excellent compatibility with a wide range of multimedia hardware, including CD-ROM drives, waveform audio devices and videodiscs. See section 5.23 for an extended discussion of MCI under Windows.

Toolbook use of object-oriented concepts along with its powerful scripting language provide much of the environment's flexibility. Each element in a book is an object. This includes such things as buttons, text, menus, pages, the book, bitmaps, etc. Sections of text may be selected as "hotwords" which are also treated as objects. Two or more objects may be joined in a "group" - a logical association in which the group itself becomes one object. Every object is treated equally and may have a script associated with it to determine its behavior based on messages sent to the object. An object may receive messages from the Toolbook system in response to events (such as a keystroke or a mouse click) or another object may explicitly send a message. A message can be considered similar to a function call in a traditional programming language - it is a request for an object to perform some action.

The scripting language of Toolbook, OpenScript, is very well-developed and powerful. It contains functions for manipulating objects, sending messages to objects, math functions, variables, string manipulation. A script can create Window's windows, add function to menus of windows, perform DDE with other applications, or use MCI functions to control hardware devices directly.

Although the OpenScript language can be complex to use, Toolbook provides assistance. There is a "recorder" function which allows a developer to record a series of actions performed with mouse and keyboard. This could be manipulation of an object, changing to another page of the book, etc. When done the recorder will generate an OpenScript script for any object that will perform those same functions. Thus, for simple tasks, the developer need not know anything about OpenScript, the environment will automatically generate necessary scripts.

Another example of this type of assistance is the MediaBlitz package. MediaBlitz is an add-on for Toolbook which allows a developer to interactively create clips (sequences of animation, sound or CD-Audio). A clip can be associated with a Toolbook object and whenever that object is activated (clicked on) MediaBlitz will be invoked to play the clip. The developer does not need to know how to interact with an MCI hardware device - MediaBlitz does it automatically. If desired

the developer always has the options to use the MCI calls from OpenScript to explicitly control devices but MediaBlitz will fill most simple multimedia needs.

Toolbook does not provide any facility for creating animation except for using a script to control movement and positioning of objects on the screen. The MediaBlitz extensions do support several true animation formats and these may be incorporated into a Toolbook application (although Toolbook can not be used to create or edit them).

Toolbook applications are stored in a single file containing all text, graphics, objects, etc. The exception to this is MediaBlitz clip files which are stored separately. A file may be password protected against unauthorized changes. As a book is stored as one file it is impossible to change it without knowing the password.

Finally, Toolbook offers a substantial amount of online help for authoring features, MediaBlitz and OpenScript. The online help extends to interactive debugging of OpenScript scripts. System prompts and information windows appear on the discovery of a script error to assist in debugging. Also included with the package is excellent printed documentation and a number of prebuilt applications to act as tutorials and illustrate possibilities for applications.

4.32 Weaknesses

Although there are a large number of strengths in Toolbook there are also a points which could be improved. The most serious point is the complexity of the environment. A developer must adapt to the object-oriented philosophy and learn the OpenScript language to take maximum advantage of Toolbook. This is not to say that it impossible to create applications without knowing either - Toolbook provides a good set of functions for defining books and automating simple tasks. However more complicated applications will require a greater depth of involvement for the developer.

Learning the subtle nuances of Toolbook represents a substantial learning curve. OpenScript is a very diverse and flexible language. Often the is a number of possibilities for solving the same problem using different combinations of messages and objects. Several times during the course of preparing the Crunchy Pops application complex operations were created using several pages of OpenScript code. Later this code was replaced by just a few lines when a better method was discovered. The problem can be likened to "information overload". There is such a wealth of possibilities available that even with the documentation and examples available it is difficult to determine the best method to solve a problem.

Likewise the sum of an application is represented by the various diverse object which it is composed of There is no central repository that can be consulted to determine the way in which an application works. Each object must be examined individually. This is a side effect of the WYSIWYG object-oriented authoring environment. This concept is explored further in section 5.22. On a related note it is possible for Toolbook objects to be visible or hidden. Once hidden they may be forgotten about (or inaccessible if the name of the object is not known). This has the tendency to generate "lost" objects in the application.

Toolbook can import several graphics formats, namely .BMP (Bitmap), .DIB (Device Independent Bitmap), .CGM (Computer Graphics Metafile), .DRW (Micrografx Draw), .EPS (Encapsulated PostScript), .TIF (Tagged Image File Format) and .WMF (Windows Metafile). However, aside from clipping and scaling these images Toolbook provides no facility for editing them. It depends on using the Windows clipboard to exchange these pictures with another program for editing (say the Windows Paint utility). This is certainly a workable option but it could be handled better with either an integrated bitmap editor program or using DDE to directly control the Windows Paint program as if it were a part of Toolbook. Essentially the process of editing a bitmap is somewhat clumsy under Toolbook. It is not taking full advantage of the integrated Windows environment in this case.

Toolbook provides only a limited "special effects" transition between pages of the book. Where some of the other authoring packages offer very elaborate and varied dissolves between screens Toolbook offers only one (and a very slow on under Windows at that). Toolbook is not primarily designed for simple presentation type applications so this omission is understandable.

There are also some minor difficulties using scripts under Toolbook. During the course of this project no method was discovered to halt script execution. Also, scripts for the "enterpage" message (a message sent to a page object when that page is displayed) are executed automatically, even in author mode. These two factor in combination make modification of certain pages difficult if the "enterpage" script for the page object moves the user to another page. There is effectively no way to modify these pages because it is impossible to stay on them.

As elements are objects in Toolbook, and all objects are treated equally ,it is not possible to indicate clickable objects. Any object can receive and process a mouse click. Thus the mouse cursor never changes to indicate objects that have specific behaviors when clicked.

Finally Toolbook requires the developer to explicitly change from author mode to reader mode to test how an application functions. This is not a major inconvenience (it can be accomplished by a single shortcut key) but , as will be illustrated by Linkway, there is slightly quicker way to test the behavior of an object. Linkway allows a double click on an object to activate that object in author mode. This is a very minor point but none the less a nice convenience.

4.4 Linkway

Linkway is the third multimedia authoring system examined during this project. It is from IBM and for the DOS environment. Linkway is similar to Toolbook in overall concept: it is an object-oriented package. The application, called a "folder", is divided into pages which may contain buttons and text (objects). Linkway provides its own integrated graphical user interface for authoring and running applications. An application is edited onscreen in WYSIWYG style.

Linkway has a number of tools which allow editing of graphics screens, recording and playback of sound, and graphics conversion. The package is primarily oriented towards the construction of simple hypermedia applications. Many of the different tools of the package were tested during

this project although not all of these tests are reflected in the Crunchy Pops application implemented under Linkway.

In the same manner as Toolbook, Linkway offers an authoring mode (for construction and editing of objects) and a user mode (for testing and actually running an application). Actually Linkway provides several degrees of control between these two modes but these are the most important ones. Linkway folders may be protected by password from being changed.

4.41 Strengths

The greatest strength of Linkway is its hypermedia environment and WYSIWYG authoring ability. A developer can quickly create and structure applications using Linkway's tools. As with Toolbook applications are divided into pages which contain objects (buttons, text, popups, etc.). Linkway provides a base page to define objects which will appear on all pages of a book (folder as it is referred to in Linkway).

Linkway uses an object-oriented approach similar to that of Toolbook, but to a lesser degree than Toolbook. Every object is specialized and performs exactly one function. For example buttons are divided into several categories including link buttons, text buttons, script buttons and picture buttons. Link buttons transfer a user to another page of the application when clicked. Text buttons display a text popup when selected. Picture buttons display a graphic popup. Script buttons run a script in Linkway's scripting language. There is exactly one script for a script button which is run from beginning to completion. The script may invoke other buttons. Objects are therefore much less generic and flexible then in Toolbook and suitable for simpler applications. See section 5.22 for a more thorough discussion of object-oriented concepts in multimedia authoring.

The scripting language supplied with Linkway is fairly flexible, providing a number of commands for manipulating objects and the screen, accepting mouse and keyboard input, invoking external DOS programs, reading/writing external files and communication with other buttons. The script language is limited to a CALL() (subroutine call of another button) and JUMP (read "goto") and IF statements for sequence control. The worse problem with Linkway's scripting language is that a script may not itself contain function or subroutines, have any significant control structures, and is limited to 4000 bytes in length.

There are true text and graphics objects in Linkway as well as text and graphics popups (when implemented using the appropriate special purpose buttons). Unfortunately their implementation leaves much to be desired as will be discussed in the next section. Linkway scripts do allow for defining and using small popup menus to query a user and control an application at run time. This is a very useful feature.

During the execution of a Linkway application Linkway supplies a mouse cursor. The course cursor changes shape (a an actually visible fashion) when it is over an object that can be clicked. This is one area where Linkway succeeds and the other environments fail.

Linkway does provide for multimedia capabilities in the form of CD-AUDIO, waveform audio and videodisk control. Only a videodisk may be directly controlled from within a script by issuing commands to it. CD-AUDIO and waveform audio support is implemented by an external DOS program which must be called from a Linkway script. This will be discussed in more detail in the next section.

Linkway provides several demonstration applications illustrating how to accomplish various tasks (such as multimedia hardware control). It also features a complete on-line interactive tutorial illustrating how to use many of the features of Linkway. The printed documentation only deals with the use of the application.

The final major strength of Linkway is a very good integration of the authoring/test environment. It is not necessary to change from author mode to reader mode to test the application (although this is possible). Clicking on any object in author mode selects it for editing. Double-clicking an object activates it as if the developer were running in reader mode. In this fashion new objects can be tested immediately and efficiently.

4.42 Weaknesses

Although the basic concepts behind Linkway are sound the actual implementation leaves much to be desired. Linkway is, unfortunately, extremely inflexible in many areas which limits its usefulness. Major problems include arbitrary limits on certain properties of objects, poor overall look for the package and poor integrated support for many multimedia operations.

Many objects have arbitrary limitations or restrictions which affect applications created with them. For example text popups and graphics popups are limited to an arbitrary X,Y size which hinders their usage. There is a limit of one base page per application, all pages of the folder must share the same base page. The only way to have separate base pages for various parts of an application is to split the application into several folders and navigate between them in the application.

The implementation of text and graphics popups is limited as well. A developer defines a rectangular area that will be occupied by the popup. The button which activates that popup must be in one of the four corners of that area (or alternately the entire area may be a button). There is no facility for creating a button and having a popup appear at an arbitrary position on the screen.

Linkway does not have true hypertext capabilities for applications. The exception to this is the help facility (a series of text popups) which may have links defined for portions of text to automatically invoke other help popups. However a developer is forced to manually edit the help text and put in control codes which represent the help files to be loaded when a particular section of text is clicked on - hardly an automated solution.

Overall the appearance of Linkway leaves much to be desired. The buttons available for use are not particularly attractive. Choices of fonts for text objects are limited to a few styles. Text popups are worse, using a very unappealing style of scrolling text box (it actually has to be seen to be fully appreciated - see the Crunchy Pops Linkway application). All text popups present a

flashing cursor marking the current typing position - even those popups which cannot be edited by the user. The script editing facility of Linkway is likewise limited to the same text popup boxes. Editing scripts is a very slow, painful process. There are not even simple tools for cutting and pasting text in scripts. The only form of animation directly supported by Linkway is to use a script to move objects around a page - there is no native animation format. The use of a base page does allow some consistency for the "look" of an application.

Linkway applications are stored as the original files which were used to generate them. As previously mentioned folders can be protected from unauthorized modification with the use of passwords. However, as the graphics and text files are stored separately, there is nothing to prevent them from being modified.

The last major problem with Linkway is that it does not present a well integrated solution for controlling multimedia devices. With the exception of laserdisk players, all multimedia hardware is controlled by external programs, there is no internal support. For example, external DOS programs are provided to record from on and back digital waveform audio on an IBM M-Audio capture board. Likewise an external DOS program controls a CD-ROM player. This is a very clumsy solution to the problem of multimedia device control for two reasons:

- DOS is a single task environment. While a DOS program is executing under Linkway, Linkway itself is not running. Thus animation (for example) and digital waveform audio cannot be used simultaneously.
- Linkway itself provides no assistance in preparing multimedia "clips" (CD-AUDIO and waveform audio). The developer must learn and use the syntax of the supplied DOS commands. Proper clips are created through trial and error using the various parameters.

The waveform audio tools use the same format as Audio Visual Connection and thus may share sounds. However the tools supplied with Linkway provide no facility for editing sound clips to remove unwanted sections. The standalone DOS sound program is designed for several types of IBM audio boards. The CD-ROM control program is designed to use Microsoft's MSCDEX CD-ROM extensions for DOS (an industry standard interface for controlling a CD player) and thus should work with many brands of CD player (although considerable trial and error of command line parameters may be necessary).

For digitizing video the approach is even worse (almost comical, really). The Linkway manual lists that the IBM Video Capture Adapter/A is an optional piece of equipment for this product. The following directions appear in the Linkway manual for using the Video Capture Adapter/A (page9-4):

1. Install the IBM Video Capture Adapter/A according to the instructions with the adapter.
2. Install IBM Storyboard Plus Version 2.0 according to the instructions in the manual. This software is used to capture the image from video.
3. Connect the input equipment (videotape player, videodisc player, or video camera) to the Video Capture Adapter.
4. Locate the image you want to save on the Video Capture Adapter.
5. Follow the Storyboard Plus instructions to capture the image.

6. Use SBconvert, the Linkway conversion program. This program is used to convert pictures within the same graphics mode from Storyboard Plus to Linkway [... description deleted ...].
7. The captured image is now ready for your graphics presentation in Linkway. [... description of usage deleted ...].

In other words Linkway requires Storyboard Plus (or Storyboard Live) to digitize images for it. There is absolutely no integrated support for this hardware, not even a stand-alone DOS program.

4.5 Portability

Over the course of this project each of the authoring systems (and applications) was run on two separate PC compatible microcomputers. This provided a test of the portability of the systems between different hardware. The configuration of the two systems was:

System 1:	IBM PS/2 Model 80	**System 2**:	Generic PC clone
	80386 processor (25 Mhz?)		80486 processor 33 Mhz
	10 MB main memory		8 MB main memory
	120 MB hard drive		240 MB hard drive
	IBM VGA graphics		Boca SVGA graphics
	IBM M-Audio sound card		Gravis Ultrasound sound card
	IBM CD-ROM player		no CD-ROM player
	Microchannel system bus		ISA system bus
	DOS 5.0, Windows 3.1, OS/2 1.3		DOS 5.0, Windows 3.1,
			OS/2 2.1 Dec /93 beta

All authoring environments ran successfully on both platforms using standard VGA graphics mode for applications. Due to the dependence of Linkway, AVC and Storyboard Live on IBM proprietary hardware they were not able to make use of the Ultrasound board. However Toolbook, running under Windows 3.1, was able to make use of the board without any modification.

The other portability consideration is speed. On the PS/2 the Toolbook and Storyboard Live platforms suffer from performance problems. On the '486 system (with 2 to 3 times the total system performance of the PS/2) response of each authoring environment was excellent. For multimedia authoring system it is certainly worth the money, in terms of performance, to invest in the fastest hardware available.

4.6 Conclusions about authoring systems evaluated

After implementing the Crunchy Pops application under the various authoring environment and discussing some of the strengths and weaknesses of the various packages, some conclusions can

be made. This section will, briefly, assess the potential of each authoring system for implementing applications.

Storyboard Live is most suitable for linear full screen presentations and simple interactive application (without any text input capability). It offers a fairly simple array of tools for creating such presentations. The graphics editor is powerful enough to provide the basic functions of creating screens and the package can convert between a wide variety of graphics formats. The limited animation (sprite) capability adds flavor to applications. It includes the basic features necessary to create simple applications. Sticking to simple applications the rather clumsy interface for editing scripts should not impose too great a limitation. Storyboard Live is most limited in the multimedia hardware which it supports. There is no capability to support a CD-ROM drive and the waveform audio support is limited.

Audio Visual Connection, similar to Storyboard Live, is most suitable for simple presentations. Unfortunately the poor user interface for the authoring functions make it difficult to actually recommend this package for any use. Its scripting language is much more structured and powerful then that of Storyboard Live and could conceivable be used to create fairly complex applications. It has good capabilities but it is questionable as to whether the effort required to harness those capabilities is worthwhile. Also, the package is limited to supporting IBM multimedia hardware making portability difficult. Finally, the package is very limited in its ability to import data files (sound and graphics).

Toolbook is easily the most powerful and flexible package evaluated here. It makes good use of the Windows environment, is user friendly and extremely flexible. Toolbook is suitable for both simple applications (which can be created in a highly automated fashion) and very complex applications. Toolbook is able to make use of many multimedia devices and, as will be discussed in later sections, is very flexible in terms of device independence. Toolbook is the only package evaluated here which could actually, by itself, be used to create a substantial professional quality application. Although the various parts of Toolbook can be daunting in their complexity to a new user, the rewards for mastering those functions are well worth it.

On the surface Linkway should have the same capabilities as Toolbook. It is hindered by an overall unattractiveness of its elements, arbitrary limitations on many objects and lack of integrated support for any type of multimedia hardware. It is suitable for simple hypermedia applications but lacks the complexity to implement truly substantial applications. Toolbook could implement any application that Linkway could be used for and would do a much better job.

In closing it is interesting to note that the three IBM packages are the weakest examined. Toolbook's capabilities easily outpace all of them. Most of the IBM packages concentrate on IBM hardware to the exclusion of all else. The user interface is totally different between all three IBM packages even down to such basic functions as the use of the mouse in the paint module (which all three supply). Also the potential for exchange of data files between the three IBM programs is very bad (with the exception of Storyboard Live with graphics files). It is interesting to contemplate that three such radically different and generally incompatible solutions for similar application areas could come from the same company.

5.0 Multimedia Authoring Concepts

This section discusses some of the concepts for multimedia authoring systems discovered during this project. These consist of different approaches to solving similar problems that the various authoring system use. Specifically more attention will be given to the issue of the "master" script authoring systems vs. the object-oriented authoring systems. Also, the concept and implementation of device independence will be examined for multimedia applications.

5.1 Application Structuring

There are two predominant methods used to structure applications in the four authoring systems examined. Storyboard Live and Audio Visual Connection used what could be best called a "master" script approach. They have a scripting language and use one single script to control all aspects of an application. Toolbook and Linkway use an object-oriented approach. All elements which appear in the final application are objects which can be manipulated by the developer using the authoring system. The interaction between objects defines the application. Individual object behaviors may or may not be defined using a scripting language.

Each of these approaches offers distinct advantages and disadvantages as discussed in the following subsections.

5.21 Master Script Approach

The master script approach is similar in concept to traditional programming languages. A single script is created which defines and controls all aspects of an application. As previously discussed both the Storyboard Live and Audio Visual Connection environments use this type of control for applications. The script for these authoring systems handles such activities as:

- Displaying graphics and text on the screen.
- Accepting user input from mouse, keyboard and other sources.
- Making decisions about how to proceed in the application based on user input.
- Controlling multimedia devices.

The developer is responsible for communicating, through the script, every aspect of behavior for the application. The entire structure and sequence of the application is defined by the script.

Scripts may contain such programming language elements as variables, control structures and the concept of functions and subroutines. The script language is actually a simple programming language with specific constructs for multimedia applications. Although the term "master" script is used in describing these type of environments this does not mean that there is only one script per application. In both the case of AVC and Storyboard Live scripts are capable of invoking other scripts to performs tasks or jumping to subroutines and functions within the current script.

This approach lends itself well to simple applications. It is possible to quickly define an application which shows screen X, Y and Z in succession waiting for a mouse click in between. It is also possible to define various subroutines and functions to handle distinct aspects of the application. This approach to application development works best for simple, fairly linear applications. Applications can be structured very rapidly when remaining within these limitations.

The master script does not lend itself well to very complicated applications. True hypermedia capabilities are not possible (they could be simulated to some extent but it would greatly complicate the script and make future modification and enhancement difficult). This is primarily due to the density of information that may be present is a hypermedia presentation. A screen may have many links to other screens, which themselves have many links. The number of scripts and complexity of the program quickly gets out of hand. Also hypermedia implies the use of "goto" type structures, not structured programming. This greatly increases the complexity and hurts the readability of a script.

The most serious disadvantage to using a script to control all aspects of an application is a loss of immediacy when creating the application. It is possible to determine the control transitions by looking at a script but it is not possible to determine the actual appearance of the application. The development cycle consists of making changes to the script and then executing it to determine if the desired behavior has been accomplished.

It should be noted that both AVC and Storyboard Live do offer some on-screen assistance for creating applications. For example, when trying to position a graphical popup on a screen a certain point in a script both authoring systems:

- Allow the used to select the portion of a picture to use as a graphical popup.
- Interpret a portion of the script that occurs before that point.
- Try to determine what the screen would look like at that point based on the interpreted script.
- Display the interpretation of what the application screen would look like and allow the developer to position the popup on that screen.
- "Fill in" relevant information fields in the script representing the choices the developer made in selecting and positioning the graphical popup.

This does allow some measure of interactive control when creating an application but it should also be noted that neither environment is always successful in "estimating" what the actual application screen would look like at a given point. The actual screen may be determined by the values of variables and the execution of control structures at run time. Therefore it is still necessary to actually run the application to determine if changes made to the script have the desired effect.

Appendix D contains some code examples of the scripts used for the Crunchy Pops application for the AVC and Storyboard Live authoring systems.

5.22 Object-Oriented Approach

The object-orient approach to authoring systems differs radically from the master script approach. Under this approach every element of an application becomes an individual entity with its own properties - an *object*. Objects include such things as bitmaps, buttons, text, vector drawings, etc. The application becomes a "book" and the individual screens become "pages" of the book (to use the Toolbook terminology). The book and its pages can be considered objects as well. Object such as text, bitmaps and buttons are associated with specific pages of the book and appear on the screen whenever a given page is displayed. Both Linkway and Toolbook used the object-oriented approach to creating applications. Generally the object-oriented approach implies that a WYSIWYG environment is used for the authoring system (this is not a *requirement* but it is the most logical way to proceed). Both Toolbook and Linkway use a WYSIWYG environment.

The power of this approach to authoring become clear when creating an application. The pages of an application appear to the developer exactly (or almost exactly) as they will to the end user. The developer uses various tools to create and position objects on pages of a book. The "look" of the application is immediately visible on the screen. There is no guesswork as to whether objects are positioned properly or not. This is possible because the objects define the look of the application and a WYSIWYG environment allows direct on-screen manipulation of objects. With the master script approach the script defines the application so it is the script which is edited on screen.

The other major difference is the method used to control the application. There is no central script which guides the application. The interaction between objects defines the behavior of an application. Objects receive "messages", in the form of mouse clicks or keystrokes or explicit messages sent from the script of another object. These messages cause an object to perform some type of behavior. This interaction defines the application.

In specific implementation of the object-oriented approach Toolbook and Linkway differ. Linkway has special single purpose object, for example: "picture buttons" which display graphical popups when clicked, "link buttons" which display another page of the book when clicked, and "script buttons" which run a single script of Linkway's scripting language when clicked. A script may send messages to other objects to invoke their behaviors but the essential point is that each object has only one behavior in of itself and invokes that behavior when it receives a message. The objects define the flow of control and behavior of an application.

Toolbook takes a much more general and flexible approach to objects. It has a very powerful scripting language called OpenScript. Every object may have a script in which it defines behaviors for any messages that it wants to respond to (a message in OpenScript is a character string, e.g. "message"). An object may define behaviors for any number of messages. Any object may send any message to any other object. For example a button object, a text object and a bitmap picture object may all define behaviors for the message that is automatically sent to them whenever they are clicked on by a mouse. A button object could send a message to a picture object to hide itself or show itself whenever the button was clicked. For that matter a picture

object could send a message to a button object to hide itself whenever the picture is clicked. All objects are treated equally in Toolbook which yields much of the flexibility of the environment.

Furthermore objects may be "grouped" in Toolbook. A group is a logical association between two or more objects. The group is treated as a single object. The group may have a script defined for it and respond to any messages and the object in a group may have scripts and respond to messages directed to them.

The Toolbook environment automatically sends message to objects when various events happen. Whenever a page is displayed the old page object receives a "leavepage" message and the new page object being displayed receives an "enterpage" message. Page object may have scripts defined for them like any other object and could invoke specific behaviors upon reception of these messages. Very subtle and complex behaviors may be defined for applications based on the interaction of objects and messages within the application.

The object-oriented approach offers two major advantages over the master script approach. There is a large degree of immediacy in the environment as objects are placed and edited as they will appear to the end user. The use of objects and messages allows extremely complex applications to be created. Indeed with the ease of creating applications in object-oriented authoring environments even very simple, linear applications are as easy to create as they are for a master script based authoring system.

However the object-oriented approach is not without its disadvantages. There is no central script nor any other centralized facility which can be consulted to determine the flow of control of an application. Only execution of the application or thorough examination of object will reveal how the application works. As the interaction between objects may be very complex and it may be difficult to debug an application or modify an application at a later date. In the case of Toolbook such modification may literally involve checking the script of each object to determine the overall flow of control. Linkway would be somewhat easier to modify as its object are all single purpose and thus there is much less interaction between them.

In the case of an environment such as Toolbook where objects may be hidden it is possible to create objects, hide them and then forget about them. This creates "lost" objects which, while not detrimental to the application, is still annoying.

Appendix E contains samples of the OpenScript language. This language is used by Toolbook to define behaviors for objects in a book based on messages received from other objects.

5.2 Hardware Device Independence

A multimedia authoring system is most effective when applications created under it are compatible with a wide variety of hardware devices. This allows applications to be ported easily to other computers with different hardware. For example, if an authoring system supports digital audio playback then it would be desirable to have it work with Brand A, Brand B and Brand C digital audio playback adapters. An application could be created on a computer with a Brand C adapter

and moved to another computer with a Brand A adapter and the application would still function properly.

Essentially there are two methods by which this hardware independence can be achieved. An authoring system can supply drivers for every hardware device with which it can work (alternately hardware manufacturers may supply drivers for a particular authoring system) or an authoring system may make use of an industry standard device interface protocol.

5.21 Authoring system specific hardware drivers

An authoring system which supplies drivers for specific hardware devices has at least one distinct advantage: that authoring system can take advantage of every specific feature of that hardware device to full capability. However there is a significant disadvantage: the authoring system is limited to a specific set of hardware devices which restricts portability. It is unlikely that an authoring system will continually release new drivers for every new multimedia hardware device from every manufacturer. It is equally unlikely that hardware manufacturers will release drivers for every individual multimedia authoring environment on the market. Thus, support for new hardware devices is very limited.

The three IBM authoring systems used in this project take this approach to device control. They supply drivers for various IBM manufactured audio devices (with the exception of Storyboard Live which also supports Creative Labs Soundblaster compatible adapters). It is therefore unlikely that Storyboard Live, Audio Visual Connection and Linkway will support new multimedia hardware without a software upgrade. However, Audio Visual Connection, which was designed to use the IBM M-Audio capture adapter did work extremely well with the board without any special configuration (and was the only authoring system which this capture board worked to its full potential without considerable configuring).

5.22 Industry Standard drivers

An alternate approach to hardware independence is to have an authoring system adhere to some industry standard for device control. This places that onus for support on the hardware manufacturer: the manufacturer supplies drivers which adhere to the industry standard and the authoring system can take advantage of the device. Generally such "industry standards" are just that: several software/hardware manufacturers agreeing on an interface and publishing documentation about the specific details.

This approach offers advantages and disadvantages. A major advantage is good compatibility between software and various hardware. A hardware manufacturer is likely to provide drivers for a well-established industry standard, in fact it often become a selling point for the device. An authoring system only has to support the standard and need not worry about the specific devices it is actually used with.

The main disadvantage to this approach is that it leads to a "lowest common denominator" form of support for hardware devices. A new audio adapter may offer an extremely powerful, new

hardware option but if a given industry standard does not recognize and make allowances for this capability then existing software would not be able to make use of it. This is not to say that the audio adapter itself could not be used with used with a given application, only that the functionality that it provides would be limited to whatever the industry standard allows for.

There are three methods for overcoming this type of "special feature" problem:

1. Update the industry standard. This will allow support but will require an software upgrade of existing applications and authoring systems to be able to use the new version of the standard.
2. Allow the authoring system to "know" some specific features of hardware devices it is actually using in addition to supporting the industry standard interface[4]. This solves the problem but leads into the area of hardware dependency which is what the industry standard is trying to solve in the first place.
3. Make the initial definition of the industry standard very wide ranging and broad in scope. Essentially try to encompass as many capabilities for a given class of device as possible. Provide the facility for an application to query a hardware device (actually the drivers for that device) to find out which capabilities it actually supports[5]. With this information an application can make maximum use of a given device. Note that an application still does not need to know any hardware specific details of a device only which parts of the standard are implemented and available. This approach is basically raising the "lowest common denominator" for a standard when the standard is defined. Manufacturers may choose to support those facets of the standard which they desire to.

There is another form of adhering to an industry standard which should be mentioned. A manufacturer may directly emulate, at a hardware level, another brand of multimedia device. From the point of a multimedia authoring system both devices appear to be exactly the same (no special treatment for the "new" device needed). This is essentially the same situation as discussed in the previous section: a multimedia authoring system must directly support this hardware configuration with all the associated advantages and disadvantages. However an authoring system which only supports the hardware being emulated cannot take advantage of any specific features of the "new" device.

A good example of this type of hardware emulation is found in the wide number of audio adapters which emulate the Creative Lab's Soundblaster series (which itself emulates Adlib Incorporated's Adlib audio card). An authoring system which supports an Adlib card would also work with a Soundblaster or any of the devices which emulate it. The authoring system would not be able to use the Soundblaster specific features of these devices when treating them as an Adlib card. Another example of this type of hardware compatibility exists for video display cards. The VGA standard is well established by IBM and there are many third-party video adapters which implement VGA compatibility at a hardware level.

---

[4]It is worthy of note that Toolbook actually takes this approach for some devices along with using MCI.
[5]Windows 3.1 MCI actually allows this capability for devices.

The specific case of Windows 3.1 using the MCI protocol for multimedia device control will be considered in the next section.

5.23 The MCI Advantage

One of the major enhancements included with Windows 3.1 (over Windows 3.0) was that of Media Control Interface (MCI)[6]. MCI defines a set of control protocols and functions for dealing with various multimedia hardware devices, including Compact Disk drives, digital audio devices and MIDI devices. Any such device that supplies an MCI driver for use with Windows can automatically be used by MCI aware programs.

For example, MCI commands for digital audio devices include commands to record and play back digital sound samples. An MCI aware application can open a digital audio device, send the commands to play a sound (supplying the sound sample) and the device will perform the operation. The application never needs to know the hardware details of the device actually playing the sound, only the MCI commands to access it. Thus MCI provides a layer of abstraction between multimedia hardware and presents a consistent interface to application programs.

The following example illustrates the advantages of the MCI concept. The Crunchy Pops applications were originally developed on an IBM PS/2 model 80 microcomputer. This computer has an Microchannel internal system bus and used an IBM M-Audio Capture and Playback adapter to play back digital sound samples. The applications were then installed on a generic PC-clone 80486 microcomputer with an internal ISA system bus and a Gravis Ultrasound soundcard from Advanced Gravis Industries. The Linkway, Audio Visual Connection and Storyboard Environments were not able to take advantage of this sound card: neither they (nor Gravis) supplies the appropriate drives for compatibility.

However, Gravis *did* supply an MCI driver for use with Microsoft Windows 3.1 . Once this driver was installed the Toolbook implementation of Crunchy Pops was immediately able to make use of the sound card *without any modification*. This provides a degree of device independence that is highly desirable for any multimedia application. A Toolbook application can be easily ported to a difference machine with the same capabilities as the development system but different specific hardware. By choosing the Microsoft Windows 3.1 platform as the base for Toolbook, Asymetrix is able to exploit the power of MCI, giving it excellent compatibility with diverse multimedia hardware.

IBM has taken a similar step with the introduction of Multimedia Presentation Manager/2 [MMPM/2] for its OS/2 2.x platform. MMPM/2 provides a similar abstraction for the control of various multimedia hardware. MMPM/2 is actually a superset of Microsoft's MCI offering all of that functionality plus additional abilities such as coordinating programs and multimedia events, automatic configuration of input/output lines and close captioning. The version of IBM's Audio Visual connection used in this report was developed before MMPM/2 was introduced and thus

[6]MCI was initially introduced as a separate package "Multimedia Extensions for Windows 3.0" but became part of the base environment with Windows 3.1 .

does not take advantage of those capabilities. It is likely to be a while before applications are available that take significant advantage of MMPM/2.

6.0 Summary and conclusions

Having now considered an overview of desirable features for multimedia authoring systems, a specific implementation evaluation for four authoring systems and some of the issues confronting the implementation of said authoring systems some conclusions can be drawn.

- There in no formalized definition for the terms "multimedia". For the purposes of this project it was defined as some combination of hypermedia controls, textual information, graphics, animation, sound and video.

- Multimedia authoring systems are intended to perform two tasks. They provide a simplified interface for controlling multimedia devices, creating and structuring multimedia applications. They also interact with a user by running a created multimedia application to present information or perform some desirable activity.

- There are any number of desirable features for multimedia applications. Many of these features (such are ease of use) are based on common sense and are applicable to any application program. Some more notable desirable features include: a good integrated development environment, good multimedia hardware support (including device independence), and compatibility with a wide variety of data file formats.

- A multimedia authoring systems does not in itself guarantee that created application will be attractive, well structured or present information in a coherent manner. Although an authoring system can assist a developer the final form of the application still depends on the developer.

- The Crunchy Pops multimedia application was created to test specific features of the four multimedia authoring systems examined in this project. Although it could not test every conceivable feature of the various authoring systems it provides enough of a "real-word" test for the purposes of this project.

- There were two general approaches used for constructing multimedia applications: the "master" script approach where a single, central script controlled and defined the entire application and the object-oriented approach where an application was defined as a collection of interacting objects.

- The master script approach proved useful for simple, linear applications but not suitable for complicated, hypermedia based applications.

- The object-oriented approach adapts naturally to hypermedia applications but can be difficult to understand (from a developer's point of view) and modify as the sum of the

application is contained in all of the separate objects. There is no centralized repository of information to consult.

- The object-oriented approach lends itself to a WYSIWYG development environment which eases the task of assembling visually appealing applications and placing object in relation to each other (as all object are available on the screen to be manipulated).

- There were two approaches used for the control of multimedia hardware devices: device dependent authoring systems and device independent via industry standard drivers.

- Authoring systems using device dependent drivers were able to use hardware devices to their full potential as they "knew" the capabilities of the device. However this approach greatly limits the portability of these applications.

- Industry standard drivers offer portability of applications and hardware independence but may also (although not necessarily) restrict the flexibility of using those devices.

- The Storyboard Live authoring system offer a fairly complete integrated development environment and good compatibility among a large number of file formats. Due to the use of the master script approach it is most suitable for simpler, linear full screen graphics presentations.

- The Audio Visual Connection authoring system is hampered by a generally poor user interface. It offers a flexible and powerful scripting language (master script based environment) in which it is actually possible to support limited hypermedia applications. However, the user interface problems coupled with a total dependence on IBM hardware and very limited options for importing data files make this an undesirable environment to implement most applications.

- The Toolbook authoring system was by far the most flexible and powerful environment to implement applications. The object-oriented approach in conjunction with WYSIWYG editing, a powerful scripting language, good support for novice and experienced users and good hardware independence make this an ideal environment for implementing almost any type of multimedia application.

- The Linkway authoring environment offers an WYSIWYG and a limited object-oriented interface. It also has limited hypermedia capabilities. Linkway suffers from a generally unattractive interface, arbitrary limitations on certain objects and a total lack of integrated support for multimedia hardware. It is suitable only for simple, basic hypermedia applications.

Recommendations: use Toolbook and the most powerful computer available.

Appendix A - Glossary of Terms

AVC Audio Visual Connection. One of the four Multimedia Authoring systems studied in this report.

CD-ROM Compact disk read only memory. Compact disks are generally used in two ways in multimedia systems. They may be regular audio CDs which can be played by any commercial CD player: this is referred to as "redbook CD audio" and is used strictly to play sound. A compact disk may also be used to hold data files (sound, animation, text) etc. which can be read and used in multimedia applications. A variety of data formats exist for this purpose. A data CD can hold over 600Mb of files.

DDE Dynamic Data Exchange. A protocol for communication amongst Windows applications. It allows applications to exchange data with and execute commands in other Windows applications.

DLL Dynamic Link Library. Under Windows and OS/2 a "library" of functions compiled from some language (C, Basic, etc.). Functions within a DLL may be invoked by Windows or OS/2 applications. DLLs may be shared by several applications simultaneously.

GUI Graphical User Interface. In general terms a user interface running in "graphics" mode which integrates the use of a mouse, multiple overlapping independent display areas called windows, icons (small bitmapped graphical objects), menus, etc. Examples: Presentation Manager for OS/2, Microsoft Windows 3.1 for DOS.

Hypermedia Various type of information "linked" together in a non-structured fashion. For example, certain words in a text passage may be linked such that when selected in some fashion (by a mouse click or keystroke) a definition is presented (which might contain links of its own). Buttons might be linked to display certain pages of a Toolbook book, play a sound sample or display text. Essentially hypermedia defines a link between various type of information and those links make be traversed by a user in any fashion.

IBM International Business Machines Corporation. 'Nuff said.

ISA Industry Standard Architecture. A specification for the internal system expansion bus of IBM PC compatible microcomputers. Designed in the early 1980's, it support 8 and 16 bit devices at a bus speed of 8MHz. One of the most common buses found on low to midrange PC compatible computers, it is slowly being phased out in favor of more sophisticated and faster designs.

MCA See Microchannel architecture.

MCI Media Control Interface. A protocol developed for Microsoft Windows that defines various functions for controlling multimedia devices (for example digital audio, MIDI devices and CD-ROM players). Any hardware device that supplies an MCI driver may be used by any software which supports that type of MCI device. MCI defines an abstraction between the raw device and a common set of control functions.

Microchannel Architecture A design for the system expansion bus of IBM PC compatible microcomputers. It was introduced by IBM in the 1980's as a faster replacement for the ISA bus. It is commonly found in high end IBM PM/2 microcomputers (as well as computers from a few other manufacturers).

MIDI Musical Instrument Digital Interface. A format for recording music in the form of "events" which my be played back through a digital musical instrument (generally a form of synthesizer). More specifically it defines a protocol for communication between digital musical instruments (and, interestingly enough, may be used to control lights as well).

MMPM/2 Multimedia Presentation Manager/2. A protocol developed for OS/2 2.x by IBM for controlling various multimedia hardware devices (e.g. digital audio, MIDI devices and CD-ROM players). Any hardware device that supplies an MMPM/2 driver may be used by any software which supports that type of device through MMPM/2. It was developed to contain superset of the functions present in MCI for Windows 3.1.

Multimedia No formal definition exists of multimedia, but it generally applies to applications that incorporate some elements of hypermedia interface, video or audio. (Note that under this definition most video games of the past decade would be classed as "multimedia".)

OpenScript The scripting language of Asymetrix's Toolbook. See Appendix E for examples of OpenScript code and usage.

OS/2 Operating System/2. IBM's 32 bit, multitasking operating system intended as a replacement for DOS (and Windows). Features Presentation Manager GUI and MMPM/2 multimedia extensions.

Presentation Manager The graphical user interface for OS/2. Windows/Icon/Mouse/Pointer concept similar to most modern GUIs such as those found under X-Windows systems and Microsoft Windows 3.1.

Waveform Audio The process of recording sound as a series of digital samples. These samples may later be used by an audio playback unit to reconstruct a representation of the original sound.

Windows A graphical user interface developed by Microsoft to run under and extend DOS. It features a standard GUI environment, cooperative multitasking, and the ability to share information between programs. It also offers MCI for multimedia device control and wide portability between different graphics adapters. Despite Microsoft's claims, Windows 3.1 is not an operating system in itself.

WYSIWYG Acronym for "what-you-see-is-what-you-get". For the case of a multimedia authoring system this can be considered to be an environment in which objects are placed and manipulated on screen interactively. The application appears, during the authoring phase, exactly as it will to the end user. Toolbook and Linkway use this approach.

Appendix B - Crunchy Pops Multimedia Application

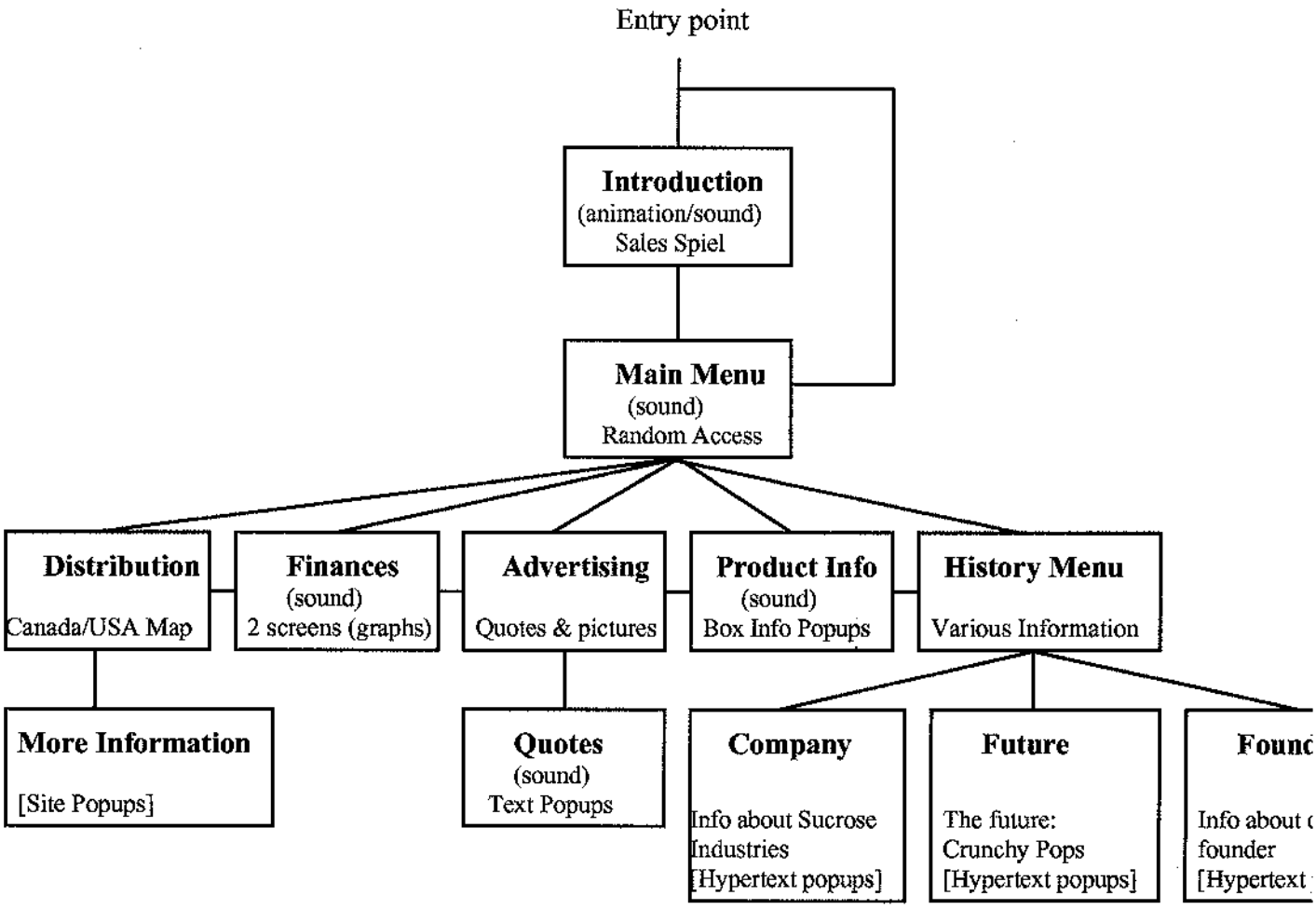
The premise of the Crunchy Pops[7] application is that of introducing a new breakfast cereal and providing specific information about that cereal and the company which is marketing it. The application includes:

- An opening sequence including sound and animation presenting the product.
- A main menu allowing random access to various branches of the application.
- Various screen presenting "facts" about the company, its products, founder and future. These screen are intended to test such features as sound, graphics and text popups and hypermedia links.

Sound support includes CD audio and digital audio (sampled wave playback). Note that features such as sound support and hypermedia links are not supported under all authoring systems and are therefore not present under all implementations of the application.

The following chart illustrates the branches of the Crunchy Pops application but running the actual applications accompanying this report provides the best description possible. Please consult appendix C for instructions for installing and running the Crunchy Pops application.

[7]Crunchy Pops, the Crunchy Pops box design, Sucrose Industries and the Sucrose Industries logo are copyright © 1993 by Phillip White and may not be reproduced or reused without expressed consent.


Entry point
Introduction
(animation/sound)
Sales Spiel
Main Menu
(sound)
Random Access
Distribution
Canada/USA Map
Finances
(sound)
2 screens (graphs)
Advertising
Quotes & pictures
Product Info
(sound)
Box Info Popups
History Menu
Various Information
More Information
[Site Popups]
Quotes
(sound)
Text Popups
Company
Info about Sucrose Industries
[Hypertext popups]
Future
The future:
Crunchy Pops
[Hypertext popups]
Found
Info about
founder
[Hypertext

Appendix C- Demonstration Applications

Enclosed with this paper are several 3.5" 1.44Mb diskettes containing the Crunchy Pops application for the four authoring system. Each diskette is labeled with the authoring system supported. This appendix contains a description of how to install and run the application under each of the authoring system

The multimedia PC currently contains all four authoring systems and the four versions of the Crunchy Pops applications. The directories containing the data files for the applications are:

C:\PDW\SBLIVE - Storyboard live version
C:\PDW\LINKWAY - Linkway version
C:\PDW\TOOLBOOK - Toolbook version
C:\POPS.!AP - Audio Visual Connection version

When using digital audio playback the files are included with the application. When CD audio sound is used the audio CD 'Spectacular Sound Effects Volume Two' from EMI Records LTD was used and should be inserted into the CD drive (if available).

Audio Visual Connection

There are 2 disks containing this implementation of the application. This application and AVC have been tested running under OS/2 1.3, OS/2 2.0 and the December 93 OS/2 2.1 beta. Notable features of this application include:

- Good integration of sound with the application. All sound is recorded in digital samples for playback through an IBM M-Audio Capture/Playback adapter. Vocal narration is included with the introduction sequence.
- A variety of dissolves between the various screens of the application.
- The mouse pointer changes from a "plus" to a "plus" with a circle around it when the pointer is over a clickable area.

To install and run this application:

1. If Audio Visual Connection and the Crunchy Pops application are installed skip to step 6.
2. Install Audio Visual Connection following directions from the manual and configure according to the graphics and sound hardware available.
3. Run AVC (type "AVC" on the command line of an OS/2 command window). On the main menu of installed applications create one called "Pops". Edit the application (this will cause the AVC to create a new directory for the Pops application). Exit AVC.
4. Using an OS/2 command window **CD** to the directory created for the Pops application (on the C: drive this would likely be the directory **C:\POPS.!AP** .
5. Copy the contents of the two diskettes into the directory for the Pops application.
6. Run AVC.

7. Select and edit the "Pops" application. (Move the hilite bar and press ALT-E) This will switch to a screen listing the various components of the Crunchy Pops application.
8. Select the "Pops" story and run. (Move the hilite bar and press ALT-P)
9. After the introduction sequence has finished the application may be exited at any time by returning to the main menu and selecting Exit. The ESC key will also quit the application at any point.

Linkway

There is one disk containing the Linkway application. Notable features of the Linkway implementation include:

- All sound is implemented via CD audio using an external DOS CD player control program (included with Linkway).
- True (and extremely ugly) text objects as well as graphics objects.
- True button objects. Mouse pointer changes shape when over a clickable object.

To install and run this application:

1. If Linkway and the Crunchy Pops application are already installed skip to step 4.
2. Install Linkway following the directions in the Linkway manual. Insure the Linkway directory added to the **PATH** statement of **AUTOEXEC.BAT** . Reboot the PC.
3. Create a directory to hold the Crunchy Pops application. Copy the contents of the diskette into this directory.
4. At the DOS prompt **CD** so that the directory containing the Crunchy Pops application is the current working directory.
5. Type the command "**linkway /v pops**". This will start Linkway in 640x480 VGA mode and run the Crunchy Pops application. Select 'Exit' from the main menu to quit the application.

Storyboard Live

The Storyboard Live version of the Crunchy Pops application is contained on one diskette. Notable features of this implementation include:

- Limited animation capability using "sprite" objects.
- A variety of dissolves between various sections of the application.
- No sound support for either CD audio or digital waveform audio (due to lack of support for CD audio and inability to get satisfactory results with waveform audio).

To install and run this application:

1. If Storyboard Live and the Crunchy Pops application are installed skip to step 4.
2. Install Storyboard Live according to the instructions in the Storyboard Live manual. Insure that the Storyboard Live directory appears in the **PATH** statements of **AUTOEXEC.BAT** . Reboot the PC.
3. Create a directory to hold the Crunchy Pops application. Copy the contents of the diskette into the directory.

4. At the DOS prompt, change the working directory to that of the Crunchy Pops application. Type the command "**st pops**" at the DOS command prompt to run the story. This will invoke the standalone story teller program to run the Crunchy Pops application.
5. To run the interactive Storyboard Live environment and edit/show the application: at the DOS prompt change to the directory of Storyboard Live (generally **C:\SBLIVE**). Type the command "**sbmenu**". Select Story Editor. Use the menus of the story editor (open command) to change to the directory of the Crunchy Pops application and load it. Run the application from the menus.
6. The Crunchy Pops application may be quit by selecting 'Exit' from the main menu.

Toolbook

The Toolbook version of the Crunchy Pops application is contained on one diskette. Notable features of this version include:

- True text and graphics objects.
- Enhanced interface over all other versions of the Crunchy Pops application. This includes new popups and true hypertext capabilities.
- Audio support for both CD audio and digital waveform audio using the **MediaBlitz** enhancements.

To install and run this application:

1. If Toolbook, Mediablitz and the Crunchy Pops application are installed skip to step 4.
2. Install Toolbook and Mediablitz using the directions supplied with the packages.
3. Create a directory for the Crunchy Pops application. Copy the contents of the diskette into this directory.
4. Start Windows 3.1. Start Toolbook from the Toolbook group.
5. Using the Toolbook 'File' menu select to open a book. Change to the directory containing the Crunchy Pops application and load the Pops book (file **POPS.TBK**).
6. The Crunchy Pops application will start running immediately upon loading. Selecting 'Exit' from the main menu of the application will close the application and exit Toolbook. **Note**: if prompted about saving changes to the POPS application when exiting select **NO**.

Appendix D - Master Script Authoring Examples

This appendix contains code samples from Storyboard Live and AVC - the two authoring systems which used the "master script" concept for application. In each authoring system a single script controls the entire presentation and interaction with an application.

The code presented here represents a portion of the Crunchy Pops application created under each authoring system. Specifically this code sample:

- Displays the opening animation and sound sequence introducing the product. This consists of a sequence of graphics screens, text and sound.
- Displays the main menu screen.
- Waits for user input (selection from menu).
- Branches to an appropriate routine (another part of the script) based on user input.

The first code sample is from Storyboard Live. There are several points of interest for this code sample:

- Except for line 1 the parameters associated with displaying graphics screens have been omitted to save space.
- This implementation of the Crunchy Pops application does not support sound (due to problems using the environment with the audio hardware).
- Storyboard Live offers true animation capability which is invoked with the "sprite" statement.
- The nature of this scripting language forces the use of a "goto" statement for transfer of control to various portions of the script.
- Buttons - clickable areas on a screen - must be explicitly defined with a "button" statement. After user input each button must be checked to determine if it was pressed.

NO.	LABEL	COMMAND	PARAMETERS	TIME	WAIT
1	INTRO FADE	/DISPLAY* | NONE | NONE | FULL	GREYBACK.PIC- 11- 21 | 101-116	0 | 532-175	0
2		/SPRITE*	PRESENT .SPR	MED	3
3		/CLEAR*		0	0
4		/SPRITE*	PRESENT .SPR	0	0
5		/DISPLAY*	MORNING .PIC-	0	2
6		/SHOW*	POPTXT2 .PIC-	0	5
7		/SHOW*	POPTXT1 .PIC-	0	5
8		/SPRITE*	CP .SPR	0	0
9		/SPRITE*	BOX .SPR	0	0
10		/SPRITE*	BOX .SPR	MED	0
11		/SPRITE*	BOX .SPR	MED	0
12		/SPRITE*	BOX .SPR	MED	0
13		/SPRITE*	BOX .SPR	0	0
14		/SPRITE*	CP .SPR	0	2
15		/CLEAR*		0	0
16		/*		0	0
17		/*		0	0

```
18 | MMENU | /DISPLAY*              | MAINMEN .PIC-     |    0 |    0 |
19 |       | /BUTTON*INTRO          |  19-208 152-241   |    0 |    0 |
20 |       | /BUTTON*DISTR          |  19-245 153-278   |    0 |    0 |
21 |       | /BUTTON*FINAN          |  19-284 153-316   |    0 |    0 |
22 |       | /BUTTON*ADVER          |  19-322 152-354   |    0 |    0 |
23 |       | /BUTTON*PCKAG          |  18-359 152-392   |    0 |    0 |
24 |       | /BUTTON*HSTRY          |  19-397 153-431   |    0 |    0 |
25 |       | /BUTTON*QUIT           |  18-435 153-468   |    0 |    0 |
26 |       | /INPUT*                |                   |    0 |    0 |
27 |       | /BUTTON*               | DISABLE ALL       |    0 |    0 |
28 |       | /IF*INTRO              | GOTO INTRO        |    0 |    0 |
29 |       | /IF*DISTR              | GOTO DISTR        |    0 |    0 |
30 |       | /IF*FINAN              | GOTO FINAN        |    0 |    0 |
31 |       | /IF*ADVER              | GOTO ELVNX        |    0 |    0 |
32 |       | /IF*PCKAG              | GOTO PRDCT        |    0 |    0 |
33 |       | /IF*HSTRY              | GOTO HSTRY        |    0 |    0 |
34 |       | /IF*QUIT               | END               |    0 |    0 |
35 |       | /GOTO*                 | MMENU             |    0 |    0 |
```

The second code sample is from Audio Visual Connection. It performs exactly the same function as the Storyboard live sample. Points of interest for this implementation include:

- Full support for two channels of digital audio (A and B).
- No true animation support.
- Variables and functions. Here the application is divided up into a series of functions which display the various branches of the system. The contents of the variable "Control" determines the transition of control within the application.
- Clickable areas of graphics screens are defined on the graphics screens themselves using the graphics editor. They are not defined in the script. After a click is made the system variable "@tf" contains the name of the area clicked.

```
/*AVC*/  /*  Text file created from POPS on Sun Apr 18 18:52:16 1993  */
Control = "Intro"
mainmenu:
do forever
   select
      when Control = "Main" then do
         play 'FANFARE'       {,,,,,,,}
         show mainmen         {STRIPES,up,,FAST,TRIGGER,full,,,,,}
         Control = @tf
      end

      when Control = "Intro" then
         Control = Intro()

      when Control = "Distr" then
         Control = Distr()

      when Control = "Finan" then
         Control = Finan()

      when Control = "Finan2" then
         Control = Finan2()

      when Control = "Adver" then
```

```
            Control = Adver()

         when Control = "Adver2" then
            Control = Adver2()

         when Control = "Packa" then
            Control = Packa()

         when Control = "Histo" then
            Control = Histo()

         when Control = "Exit" then do
            play 'crash'                {,,,,,,,}
            wait '10.0'
            exit
         end

         otherwise do
            say "Error unknown trnasition"
            say "destination " Control
            Control = "Main"
         end
      end
   end


   Intro:
      show first          {STRIPES,down,,FAST,0,FULL,,,,,,}
      play 'FANFARE'      {,A,,,,,,}
      WAIT  9.0
      show morning        {stripes,up,,fast,0,full,,,,,,}
      play 'GULLS'        {,A,,,,,,}
      wait 2.0
      paste popstxt2
{stripes,down,,,0,PART,19.0,10.2,61.7,41.8,20.7,56.8}
      play 'WHEN'         {,B,,,,,,}
      wait 7.0
      show morning        {stripes,up,,fast,0,full,,,,,,}
      wait 1.0
      paste popstxt1      {stripes,down,,,0,PART,11.0,16.2,78.5,58.9, 11.8,
38.9}
      play 'PRESENTS'     {,B,,,,,,}
      wait 7.0
      show morning        {stripes,up,,fast,0,full,,,,,,}
      wait 1.0
      paste pops          {stripes,down,,,0,PART,40.6,0,19.2,38.5, 41.0,
61.6}
      paste popstxt3
{stripes,down,,,0,PART,23.5,21.6,47.9,10.8,30.7,13.5}
      play 'CP'           {,,,,,,,}
      wait 7.0
      show morning        {stripes,up,,fast,0,full,,,,,,}
      wait 2.0
   return "Main"
```

Appendix E - Toolbook OpenScript Example

These code examples illustrate the use of Toolbook's OpenScript language. Scripts defined for two objects are presented in this appendix. Both objects are found on the page of the application which provides information about initial distribution sites for the Crunchy Pops cereal.

The specific code presented here handles the small "popups" for the distribution sites. Whenever the user clicks on a distribution site on the map a small box is displayed indicating what site was clicked. If another popup was already displayed then that popup is first hidden so that there is a maximum of one popup on screen at any one time.

The first script is that of the small graphical star indicating the location of Boise, Idaho on the map. When clicked the star graphical object is sent a "ButtonDown" message. Its script responds to that message by sending a message "showgroup" along with one parameter - the name of the group (the popup containing the information about Boise, Idaho - "BI"). This message is sent to the page object, if not handled by the page object it is then sent to the book object, etc. upwards through a hierarchy of objects until one object acts on the message.

```
to handle ButtonDown
   send showgroup "BI"
end ButtonDown
```

The second script is that of the page object - the page that the distribution map occurs on. It defines two message which it handles. When the user leaves the page it receives a "leavepage" message in which case it sends a "showgroup" message to hide any popup currently displayed on the page. The page object also defines a handler for the "showgroup" message - thus the "showgroup" message is not propagated further than the page object. This handler does several things to properly display/hide popups:

- It checks a static variable to determine if a popup is currently displayed on the page. If there is a popup displayed then it hides that popup.
- It shows the group listed at a parameter with the message (if null the parameter then no group is shown).
- The static variable is assigned the name of the group shown.

```
to handle leavePage
    hide group "MapInfo"
    send SHOWGROUP null
end leavePage

to handle SHOWGROUP fgroup
    system svShown
    if svShown is not null
        hide group svShown
    end
    if fGroup is not null
        show group fGroup
    end
    if svShown is fGroup
```

```
            break
        end
        set svShown to fGroup
    end
```

Appendix F - Problems with Specific Authoring Systems

This appendix details some of the difficulties encountered using the four authoring systems during this project. Note that solutions for some of these problems might exist but were not encountered during the course of preparing this project.

Storyboard Live

- Sprites (animated pictures) may be a maximum of 64 frames. Also the amount of memory that a single sprite may occupy is fixed. A large, complex sprite may run out of memory well before 64 frames. One sprite used in this project could only be about 34 frames due to memory limitations.
- The IBM M-Audio Capture and Playback Adapter did not work satisfactorily with this product and thus no sound is incorporated into the application. Note that under Windows it was necessary to adjust a microphone filter setting on the adapter to get satisfactory recording performance. There does not appear to be a similar option for Storyboard Live.
- Although Soundblaster and IBM audio adapters are both supported, sounds recorded under one adapter cannot be played under the other type of adapter.
- Clumsy, non-standard user interface for graphics editing and assembling scripts.

Toolbook

- It is necessary to have the Windows 3.1 drivers for the IBM M-Audio Capture and Playback Adapter installed before Toolbook can make use of this device. The record filter must be set to "high" for the card to work properly.
- There does not appear to be any way to interrupt a script during execution. This is problematical for long scripts when a bug is discovered at a certain point.
- Even in author mode scripts for the "enterpage" and "leavepage" messages for a page would be activated by going to various pages. This, coupled with not being able to interrupt a script during execution, causes problems where the "enterpage" script performs some action and then moves the user to another page. It is difficult to edit a page as the author is "kick off" by the script.
- The cooperative multitasking of Windows makes coordination between Mediablitz and OpenScript scripts problematical. When a Mediablitz action is linked to an object (say an action to play a sound sample), the Mediablitz action does not get to run until the script has completed (the script will not share the CPU with Mediablitz). This makes coordination of multimedia events and scripts difficult. This problem could be solved by using Toolbook MCI calls directly.

Linkway

- Buttons for text and graphical popups must be located in one of the four corners of the rectangular area in which the popup will appear.
- CD-ROM and soundcard control provided by external DOS programs, no integrated control.
- Extremely poor script editing facilities.
- Poor overall appearance of final application when using Linkway default objects.

- Only one base page per folder (book).

Audio Visual Connection

- Poor, character based, clumsy user interface for all authoring features. This includes script editing, shortcut keystrokes, strip menus and graphics editor.
- Very slow online help facility.

Bibliography

Asymetrix Corp, **Using Toolbook**. Asymetrix Corporation, Bellevue, WA, 1991.

Asymetrix Corp, **Using OpenScript**. Asymetrix Corporation, Bellevue, WA, 1991.

IBM Corporation, **Audio Visual Connection User Manual** Third Edition. IBM Corporation, USA, 1990.

IBM Corporation, **IBM LinkWay User Manual** Fourth Edition. IBM Corporation, USA, 1991.

IBM Corporation, **Storyboard Live 1.0 User Manual**. IBM Corporation, USA, Boca Raton, Florida, 1990.

Miller, Michael J, "Multimedia" **PC Magazine**, Volume 11 Number 6, March 31, 1992.

Poor, Alfred, "Author, Author!". **PC Magazine**, volume 11 Number 6, March 31, 1992.

Yager, Tom, "Information's Human Dimension" **Byte**, Volume 16 Number 13, December 1991.

Yager, Tom, "OS/2's Multimedia Extensions". **Byte**, Volume 18 Number 4, April 1993.

Sources used for evaluation of Crunchy Pops application:
Mr. Gregory Meldrum

Products used for creating Crunchy Pops concept:

Cow Brand Carpet & Room Deodorizer, Church & Dewight Ltd.

Lysol Brand Toilet Bowl Cleaner, Lehn & Fink Products Group, Sterling Drug Ltd. Aurora, Ontario.

Raid, S.C. Johnson and Son , Limited. Brantford. Ontario.

Shreddies, Nabisco Brand Ltd. Etobicoke, Ontario.

Spic and Span, Proctor & Gamble Inc. Toronto, Ontario.

Trounce, DL Group/Banite Inc. Toronto, Ontario.

Zero, Boyle-Midway Canada Ltd. Toronto, Ontario.