MCPS - The Multimedia Computing Presentation System

SOREL REISMAN
California State University, Fullerton

Multimedia personal computing (MPC) technology and products are evolving at a rate that is almost overwhelming. The demand for MPC applications is particularly challenging for software developers who must specify target hardware and functions in order to program and test their applications. Traditional paradigms and design methodologies can be used to develop useful MPC applications that can evolve as technology and hardware changes. The Multimedia Computing Presentation System (MCPS) was designed for application experts to be able to easily implement MPC applications by focusing on content rather than on enabling multimedia technology. By using design philosophies based on practices that have evolved from instructional technology and information systems design, MCPS-developed applications can be upgraded as multimedia technology evolves.

The introduction of personal computers in the early 1980s created a wave of new product announcements that has become almost overwhelming for software developers. More recently, the proliferation of multimedia personal computing (MPC) “standards” has exacerbated the situation. Although these waves of announcements may seem to be particularly frantic today, the problem is not new. Mainframe and minicomputer software developers encountered similar situations in the past, but they were able to produce software systems that proved to be so viable, that some of them continue to be used 20, or even 30 years later.

The key to developing information systems that are useful and have longevity, even as hardware evolves, is to thoroughly understand the needs of the end user. This is well understood in the design of education and training materials. In fact, many traditional, computer-based information systems and applications have been designed with philosophies and paradigms developed by instructional technologists.

Consider for example, Norman Crowder’s model of multiple choice, branching programmed instruction (PI) which was based on B.F. Skinner’s Stimulus/Response (S-R) learning paradigm (Frase, 1975). Initially, PI was implemented using paper-and-pencil-oriented workbooks; later, this kind of instruction was administered via electro-mechanical teaching machines. Still later, PI evolved into computer assisted instruction (CAI) (Reisman & Carr, 1991).

Interestingly, CAI’s S-R paradigm is implicit in the design of many traditional, transaction-oriented, computer-based information systems. These systems present the user with a screen of information (the stimulus), and the user enters data or selects from a menu of presented options (the response). The consequence of that response is still another stimulus, with the cycle repeated until the...
task is complete.

In today’s world of object orientism, windows and icons are modern renditions of menu options (stimuli). The act of pointing, dragging, and clicking is today’s way of responding to these stimuli. The usual consequence of icon-selection is a change of information on the screen (another stimulus). The change may be as subtle as a change in the color of the icon, or it may be as dramatic as the display of a new window of information. In any case, the S-R paradigm is as valid for today’s event-driven screen interactions as it was in the text-only world of electro-mechanical teaching machines.

Perhaps the S-R paradigm does not account for applications that are not so machine “directive.” For example, in hypertext-based applications users determine their own paths of system execution. Yet even those kinds of applications follow learning models that were the basis of Mitre Corporation’s early CAI system, TICCIT (Hunter, et al., 1975), or Seymour Papert’s LOGO programming environment (Papert, 1980). Instead of being directed through rigorous S-R processes, learners used computers to investigate paths and information of their own choosing.

Certainly, to attempt to explain the design of all information systems applications in terms of one or another of these models would be contrived. Such an argument would preclude the design of completely innovative applications that creatively draw upon new technologies to produce such personal computing applications as word processors and spreadsheets. Nonetheless, there still remains a very large number of “traditional” applications whose basic structure does or can build upon established learning models.

The Problem of Multimedia

In today’s rapidly changing personal computing industry, application developers are trying to answer the following question: “How do you design and develop multimedia computing application software for hardware platforms and functions that are continually evolving?” An objective examination of the phenomenon of “multimedia” in personal computing reveals that for the most part, MPC is not a new class of computing. Rather, MPC is a set of functions that improves the end user interface. Earlier interfaces evolved from text to graphics, monochrome to color, and single images to animation. Multimedia computing enriches that interface through the addition of sound and full motion video. Thus, in the context of S-R-based computer applications, multimedia can be viewed as functions that enrich the nature of presented information (stimuli).

With this perspective in mind, it is possible to develop many multimedia applications using traditional system design methodologies. This can be accomplished by 1) designing a robust application structure that addresses the requirements of the task, and 2) building user interfaces that utilize selected and/or available multimedia function. Note that in this strategy, application design is easily separated from design of the interface. In designing the application, the end user would play a significant role in helping to specify the content and structure of the application; programmers would be instrumental in developing the actual application. With regard to the interfaces (whether or not they include multimedia), their implementation would be the responsibility of technical personnel; end users would be instrumental in testing those interfaces. As newer or richer multimedia products become available, their inclusion or integration would not require a complete redesign of the information system.

Application Development Systems

In many ways, the approach suggested above is not so removed from textbook descriptions of the methodologies of systems analysis and design (see, for example, Pressman, 1985). However, there is often little relationship between theory and practice. One of the drawbacks to traditional system development methodologies is that end users mainly play a consultative role in application design, and hardly any role at all in implementation. While object-oriented languages such as C++ are used for the development of many “modern” applications, they do little to involve today’s computer-literate end users in the actual development process. In the earliest days of CAI, special purpose, high level languages were invented to enable instructors to develop CAI courseware. The first versions of these languages permitted instructors to model their courses after paradigms such as Crowder’s. In many ways these languages resembled today’s end-user-oriented Fourth Generation Languages.

Unfortunately, over time, the early CAI languages “evolved” by incorporating more and more complex syntactical structures for the development of applications with extremely sophisticated screen interfaces. For example, earlier versions of a language would permit an instructor to simply display a screen containing a graphic image followed by text. Later versions of the language required programmers to program the image using