RECONFIGURABILITY AS A SYSTEM DESIGN FEATURE

Sam R. Hollingsworth
Susan B. Hollingsworth
Daniel W. Miles

Hughes Simulation Systems, Inc.
Advanced Systems Facility
Minneapolis, Minnesota

ABSTRACT

Weapon systems and their associated training devices must be reconfigured periodically to meet evolving threats and take advantage of new technologies. Changes can be expensive, and can lead to difficulties in maintaining training system and weapon system concurrency. Such problems can be reduced if the need for reconfigurability is planned for early in system development. This paper presents a general approach to providing user interface reconfigurability as a system design feature, describes a specific architecture for reconfigurable interactive systems (ARIS) that supports reconfigurability requirements, and discusses applications and benefits of the approach. Under ARIS the appearance and behavior of a user interface are defined in a database that can be created and modified without changing or recompiling underlying software. ARIS has been used in the development of combat vehicle command and control simulations, an intelligence and electronic warfare console, and an embedded training delivery system.

INTRODUCTION

The user-system interface is one of the most challenging and controversial areas in the design of modern weapon systems. Conflicts often arise over the relative weight that should be attached to ease of use versus ease of development. Software developers, human factors engineers, and program managers often have competing ideas about what constitutes an effective and attractive user interface. User interfaces for computer based weapon systems with complex display and control requirements are particularly difficult to specify completely and in enough detail to guide system development adequately. User interface design guidelines help designers avoid many serious problems, but cannot provide prescriptive solutions to specific, detailed design problems, nor can they necessarily resolve issues of aesthetic taste. (4)

The MANPRINT initiative promises to improve the situation. (2) It emphasizes the importance of early design work and empirical testing and evaluation to demonstrate that a system, including its user interface, will meet performance requirements within the manpower and personnel constraints of the system's target audience description. Such precautions improve the probability that a system will be fielded successfully, without the requirement for immediate, drastic redesign of the user interface. However, the MANPRINT process must be continued throughout a weapon system's life cycle because the system must continually evolve to meet changing threats, doctrine, and personnel characteristics, and take advantage of technological advances.

This evolution is the stimulus for a reconfigurability requirement. If reconfigurability can be made a system feature, refinement of the system over time can proceed smoothly. Unfortunately, evolution tends to be very difficult if reconfigurability is not designed in from the beginning, in part because of the rigid and close coupling between software architecture and details of the user interface. In many systems, user interface changes, no matter how minor, require software changes. This leads to rigidity in the evolutionary process because users (or their MANPRINT representatives) must specify the required changes and communicate them to the software team who must interpret and implement them. Any changes that are made must be documented in DOD-STD-2167A format, potentially duplicating the initial effort by the MANPRINT team to document the change requirements. (1) This process inserts the software team between the MANPRINT team and the system configuration, depriving the MANPRINT team of direct influence over the portion of the system for which they have the greatest interest and responsibility.

Training devices and simulators suffer an additional problem because they cannot be fully designed until the design of the operational system is complete. They also cannot be updated until the features of any new system capabilities are well defined. These time lags make concurrency between the training devices and actual equipment difficult, yet training simulators must often be available to train users before the first system is fielded.

Our recent work in the development of crew display and control testbeds, training simulators, and intelligence and electronic warfare workstations for several Army agencies has led us to address reconfigurability requirements directly. The testbeds and simulators we have developed have an inherent requirement to adapt rapidly to design changes that cannot be clearly predicted. We needed a way to streamline the user interface definition and implementation process radically. The method we developed was to create a database system in which the appearance and dynamics of the user interface are stored in data files that are separate from the software that interprets the files. Most changes to the user interface are made by changing the database, not the software underlying the database. Software extensions are necessary only when fundamental new functional capabilities are required.

Although our work was initiated in the context of simulators and testbeds, it has become evident

that our approach to reconfigurability is equally appropriate for operational weapon systems and for training devices. The approach enables system designers to separate underlying system-specific functions (e.g., radar signal processing, fire control, command and control) from the interface features that give the users access to the functions. Development of the functions and the user interface can then be performed by separate software and MANPRINT teams. The teams may focus on separate issues and, to a large extent, operate in parallel with one another.

In the remainder of this paper we discuss the major characteristics of a reconfigurable architecture, offering our implementation as an example of how reconfigurability can be achieved as a design feature. We then discuss how this approach can speed the development process by enabling software development and user interface development to proceed in parallel. Finally, we discuss the implications of such an architecture for embedded training and configuration management.

FEATURES OF A RECONFIGURABLE ARCHITECTURE

General Model of Interactive Systems

The basic input-process-output model shown in Figure 1 can be used to depict the transactions that occur in an interactive system. The user provides input to the system, the input is processed, and output is generated giving feedback to the user. The input and output blocks in this model represent the user interface portion of the system. The goal of a reconfigurable interface architecture is to build in as much separation between the software used by the process and the software used to create the user interface. If a high degree of separation is achieved, the system can be decomposed into two distinct modules that can be independently designed, built, tested, and documented.

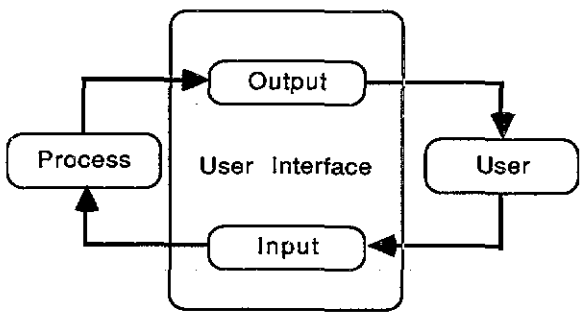

Figure 1. General model of interactive systems.

The user interface portion of a system can be further divided into content, interaction, and execution layers (Figure 2). The content layer includes the text, graphics, sound, and other forms of information presented to the user, determining the "look" of the interface. The "feel" of the interface is determined by the interaction layer, which defines the behavior of the system in response to user inputs and other events. The combination of the "look" and "feel" of the interface determines the "user-friendliness" of the system.

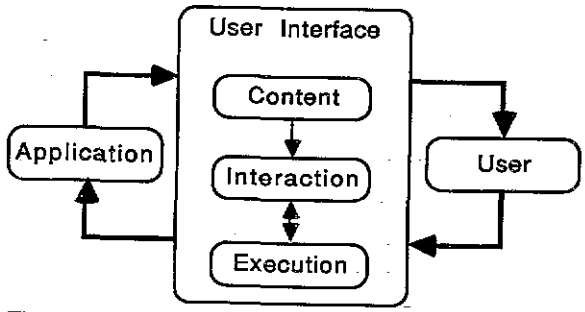

Figure 2. General model of the user interface.

The execution layer includes the software to make the interface operate in conjunction with the application program. This layer connects specific functions in the application program to corresponding functions in the interface module, manipulates the components described in the content layer, and operates on the links defined in the interaction layer.

With this model, reconfiguration of the user interface can occur at any level. Changes at the content and interaction levels directly affect the look and feel of the interface. Changes made at the execution level affect the performance characteristics of the interface, but may not directly affect look and feel.

Traditional Architectures

In a traditional architecture (Figure 3) the content, interaction, and execution layers of the interface are contained (hard coded) in software. Often the interaction between components of the interface is entwined with the code used to execute the various interface components. Changes to any one of the layers come with the expense of using software engineers to design, modify, debug, test, and document changes to the code. In this situation, reconfiguring the human interface is driven and limited by the feasibility and cost effectiveness of the software engineering effort. If the original software author, or someone who has experience maintaining the system, is unavailable to support the reconfiguration effort, the costs involved may become prohibitive.

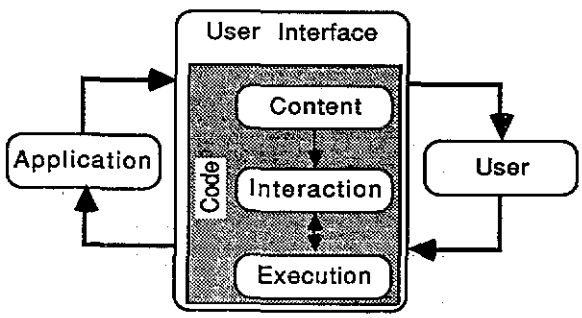

Figure 3. Traditional architecture: Content, interaction, and execution layers are hard coded.

Toolbox Architectures

Toolbox architectures standardize the interaction of the interface by providing pre-coded software modules that can be used by a variety of applications. The content of the interface is contained in a database that is accessed by the toolbox modules (Figure 4). Properly implemented, a toolbox architecture meets some of the reconfigurability requirements by allowing the content of the interface to be changed without changing software. However, the toolbox approach can make the process of going beyond the bounds of the toolbox difficult. Changes in the general appearance and behavior of toolbox elements themselves may not be feasible without significant software engineering support.

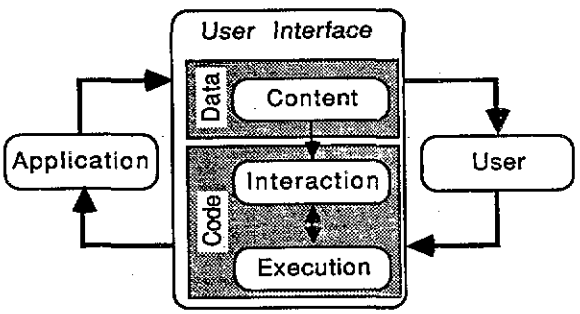

Figure 4. Toolbox architecture: Content is in a database; interaction and execution layers are hard coded.

Reconfigurable Architectures

In a reconfigurable architecture (Figure 5), both the content and the interaction layers of the interface are contained in a database. This method goes one step further than the toolbox architecture by allowing changes to the content and changes to the interactions between components of the interface, both without software modification. This allows the interface designer to tailor both the look and feel of the interface to meet system requirements, and to revise the interface as feedback is received from system users. Software modifications are not required unless new capabilities are needed in the execution layer.

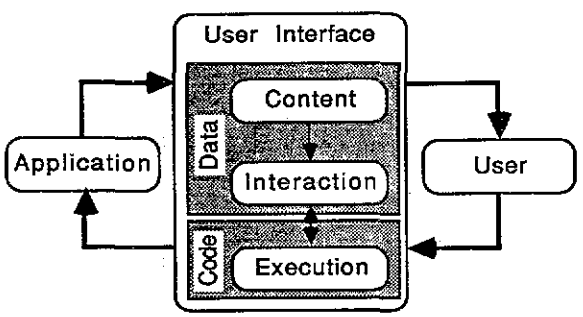

Figure 5. Reconfigurable architecture: Content and interaction are contained in a database; only the execution layer is hard coded.

PARALLEL SOFTWARE AND USER INTERFACE DEVELOPMENT

One of the major advantages of a reconfigurable architecture as described above is that it enables separate software and MANPRINT teams with separate skills and objectives to work in parallel with one another, with minimal communication requirements between the teams. The role of the software team is to provide underlying system capabilities, whereas the role of the MANPRINT team is to configure the capabilities into a specific implementation that conforms to mission requirements and user capabilities.

Color is one example of an interface attribute that can be supported in general through software development but selected and integrated by MANPRINT specialists. Software engineers generally have no professional interest in the actual color of specific display objects; their expertise is focused only on ensuring that multiple colors are available. MANPRINT engineers, however, are directly concerned with selecting the proper colors. With parallel software and MANPRINT teams operating as independently as possible, it is feasible for the MANPRINT team to explore alternative color schemes before settling on one, without disturbing the software team that is working to provide the general color capability.

Similarly, the MANPRINT team may need a variety of display objects such as buttons, pull-down menus, pop-up windows, text fields, and slider bars to create a user interface. The software team can make such objects available without being concerned with the appearance of the objects or the functions associated with them.

We have found the parallel approach to be effective in the development of user-centered interface designs. We can avoid the strictly serial process in which the MANPRINT team develops a design, hands off a specification to the software team for implementation, and then reviews the result before another iteration of the process. Instead, the MANPRINT team can use the resources created by the software team to implement the user interface directly. This reduces the serial dependencies between the teams, and permits each team to focus on what it does best.

Not all serial dependencies are removed, however. Indeed, we have encouraged an active dialog between the teams during system development as the MANPRINT engineers refine functional requirements for general capabilities that are not yet fully developed, or as software engineers provide detailed guidance in the use of the general capabilities to create specific effects. Additional communication is required in the development of the most flexible definitions possible for system-specific functions such as radar imagery display manipulation. The dialog produces synergy between the MANPRINT and software teams that would be difficult to achieve in a more conventional environment in which the software team must be involved in the most minute details of the implementation of the user interface.

The benefits of parallel MANPRINT and software engineering activities are significant in terms of cost and schedule, but there are additional benefits related to weapon system and training system evolution and concurrency when the weapon system and the training system both are constructed with a reconfigurable

architecture. The shortened, less costly change implementation cycle means that user interface enhancements may be defined, implemented, and tested much more rapidly than is usually possible, both in the weapon system and the training system. In fact, candidate revisions may be implemented first on the *training system* and tested under simulated conditions before the revisions are implemented on the weapon system--that is, the training system can lead the weapon system instead of lagging it, and can serve as a developmental testbed (within the constraints of the training schedule, of course). The shortened requirements-implementation-test schedule actually encourages experimentation and incremental improvement or tailoring of the system to changing operational needs.

EMBEDDED TRAINING APPLICATIONS

An embedded training system is a training capability that is embedded in operational equipment. Embedded training makes operational equipment serve as its own training delivery system. Using embedded training, the student receives training information via the equipment displays and enters information into the training system by directly manipulating the controls on the equipment.

Embedded training represents a unique opportunity to apply a reconfigurable architecture. By defining embedded training in terms of interface elements, the training can be decomposed into the three layers of content, interaction, and execution found in the general model for a reconfigurable interface architecture.

From a training viewpoint the embedded training system must be capable of presenting lessons in a variety of formats such as drill and practice, tutorials, simulations, scenarios, and testing. In addition, embedded training lessons must make use of the input and output devices found within the equipment to respond to general student inputs, such as answers to questions, and to respond to stimuli coming from the equipment systems and devices.

This generalized view means that an embedded training delivery system must meet the following requirements: (1) Process user input and system output using the displays and controls available on the equipment.; (2) produce text, graphics, and audio outputs; (3) simulate components found in the equipment and events occurring in the operational environment; (4) interact with equipment components and subsystems; (5) monitor and record student performance.

Using the reconfigurable architecture and an object oriented design approach we created a multipurpose courseware delivery system (MCDS) module that can be ported to a variety of systems and equipment. The MCDS is primarily the execution layer of ARIS. This layer contains objects that correspond to specific functions contained in the operational equipment which must be custom built for each system, and reusable training modules and low-level device drivers.

Following the ARIS approach, external data files are used for the content (graphics, text, audio, etc.) and the interaction between objects contained in the content layer. For example, a frame of training is defined in the database as a set of graphics, text, and data for the system to display (the content layer). The database also contains commands to invoke the appropriate training objects that present the content, record student responses, and branch to the next frame.

Using a reconfigurable architecture as the basis of an embedded training delivery system has several key advantages: (1) Lessons are contained in external data files that can be created and maintained by training specialists rather than by software engineers; (2) authoring tools can be developed to assist training designers in generating text, graphics, and audio for a lesson; (3) the separation of the lesson data files from the embedded training software allows integration of the MCDS and the operational equipment while lessons are being developed; (4) lessons can be changed after the embedded training system is integrated with the operational equipment; (5) lesson development can use a systems approach (design, implement, evaluate, and revise) to experiment with alternate training approaches and styles; (6) the human interface to the embedded training system can be reconfigured to aid in system integration and take advantage of feedback from the system users. These features combine to provide a flexible training system and efficient training development based on sound system engineering principles.

CONFIGURATION MANAGEMENT

An additional benefit of the separation of the MANPRINT and software engineering efforts permitted by ARIS is that configuration management and documentation requirements can be streamlined. The management and control of changes to software configuration items during initial development and throughout the life cycle of rapidly evolving weapon systems is typically a major effort. In a traditional software architecture, the user (or MANPRINT team) requests changes to the system. The software team translates the requests into software requirements and subsequently into software design and code changes. During this process, software documentation in accordance with DOD-STD-2167A (1), MANPRINT documentation in accordance with MIL-H-46855B guidance (3), and other logistics support documentation must be updated to reflect the new status of the design.

Much of the translation process can be eliminated with an architecture such as ARIS. Many changes to the user interface can be made directly by MANPRINT specialists without intervention by software engineers. By eliminating the software translation step from the process, the integrity of initial change recommendations is easier to maintain. Moreover, the overall amount of documentation required for the changes is less because no DOD-STD-2167A (1) revisions are needed. Consequently, configuration management for the user interface can be confined to MANPRINT documents, and may be separated from software documentation. It should be noted that the streamlining of the configuration management and documentation process pays additional benefits because it applies both to the weapon system and to the training system.

CONCLUSION

In this paper we have argued that reconfigurability can be specifically identified as a system attribute and requirement. Doing so can have the benefits of (1) streamlining the system development and modification process by encouraging MANPRINT and software

engineering teams to work synergistically in parallel; (2) facilitating the MANPRINT process, thus improving the adaptation of weapon systems to their users; (3) improving the concurrency of weapon systems and their training system; (4) providing an avenue for the direct support of embedded training; (5) reducing the configuration management burden usually associated with changes in the user interface.

ARIS is presented as an example of a system that provides the required reconfigurability. A major characteristic of ARIS is that it enables system designers to develop a database defining the appearance and behavior of the user interface for complex weapon systems. The interface may be modified extensively by changing the database but without modifying the underlying system software. If new software capabilities are required, their development is accelerated because they can be implemented without attention to details of the user interface.

ARIS is similar to toolbox systems that are becoming available, but it retains a greater degree of flexibility in terms of the graphical appearance of interface objects, their response to user inputs, and the degree of access that is provided to underlying system variables and conditions. Such flexibility is vital both for simulating systems that are already fielded and for developing advanced new concepts for which there are no precedents.

REFERENCES

(1) DOD-STD-2167A -- Defense System Software Development.

(2) Ketchie, Gary. J., Hollingsworth, Sam R., and Busch, Tamara L. "MANPRINT Implementation Model." Proceedings of the 1989 National Aerospace and Electronics Conference, pp. 804-810.

(3) MIL-H-46855B -- Human Engineering Requirements for Military Systems, Equipment and Facilities.

(4) Smith, Sidney L. and Mosier, Jane N. Guidelines for Designing User Interface Software, MITRE Corporation, 1986.

ABOUT THE AUTHORS

Sam Hollingsworth is the lead human factors engineer in the Hughes Advanced Systems Facility. He has contributed to the development of ARIS and has championed its use in a broad range of applications. He holds a PhD in experimental psychology and an MS in software engineering.

Susan Hollingsworth is a principal software engineer in the Hughes Advanced Systems Facility, and the lead architect for ARIS. She holds an MS in software engineering and a BS in computer science.

Daniel Miles is a systems engineer in the Hughes Advanced Systems Facility. He has contributed to the design and implementation of ARIS, and has led its application to embedded training. He holds a BS in computer based education.