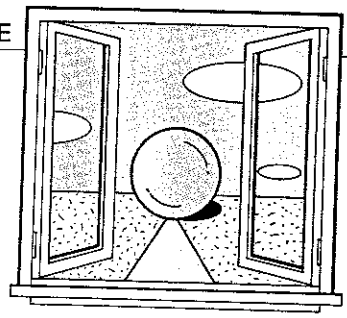
Control and Communication in User Interface Management

H. Rex Hartson

TR 88-3

revised 9/88



User-Interface Management Control and Communication

Rex Hartson, *Virginia Polytechnic Institute and State University*

Almost everyone agrees that dialogue and computation components must be separated. But an ideal separation is hard to define and even harder to achieve.

Historically, interactive software has been designed with the human-computer dialogue and application-computation components combined. The problem with this approach is now obvious: It is extremely difficult to maintain and modify such a monolithic system. With today's emphasis on iterative design, almost everyone now agrees that some kind of separation is needed between the dialogue and computation components.

But this raises new problems: Where and how do you draw the line between the components? How do you achieve communication between the components? What kind of control do you need to coordinate execution and communication?

Figure 1 shows a simple view of the dialogue-computation separation that is common to many UIMSs. A token is the smallest dialogue unit that has meaning to an application system (such as command names, options, parameters, and data values). External dialogue, which occurs between the user and the user interface, is lexically varied: It can display input and output tokens to the user in many ways. External dialogue is transfomed to and from internal dialogue via a mapping, which is provided by the dialogue developer. Internal dialogue comprises normalized tokens, each denoting a standard representation for a variety of specific external appearances of that token, which are mapped to and from application objects and structures. This mapping is provided by the application programmer. This article addresses the internal dialogue between the user interface and the computational component as carried out by the internal dialogue, not external dialogue.

No single approach solves every communication problem. The dilemma is that semantic feedback demands a close connection between the interface's dialogue component and the application seman-

0704-7459/89/0001/0062/$01.00 © 1989 IEEE

tics. But if too much semantic power is moved into the dialogue component, it can become overly complex and less separation is achieved. Less power in the dialogue component yields a cleaner separation, but raises communication overhead and jeopardizes application independence. How do you best separate these components and achieve control and communication among them?

Dialogue and semantics

To decide where to draw the line between dialogue and application semantics, you should first consider what dialogue independence means and understand the functional distinction between dialogue and semantic computation.

Dialogue independence means that design decisions that affect only the user interface are isolated from those that affect the application's structure and computational software. Dialogue independence is crucial for easy modification and maintenance of a user interface.

If you decide that a display's lexical form and appearance (for example, a menu, fill-in form, or graphics) or the grammatical relations among tokens must be improved, dialogue independence lets you change only the external dialogue and its mapping. In such cases, you do not need to change the internal dialogue; the computational components need not be aware of the change.

Of course, if you add a new function to the application, you must add both dialogue and computational parts. Hence the dialogue developer and the application programmer must agree on a standard internal dialogue. Once that is done, dialogue independence requires that the external dialogue and computational part are developed separately as much as possible by the dialogue developer and the application programmer, respectively.

Dialogue. To decide where to separate components, you should first define dialogue and user interface, terms I use synonymously in this article. Dialogue is what the user does and perceives at the input and output of a computer. However, it is an open question whether a user perceives just textual, graphical, and audio form and content or whether a user perceives the processing of input, the computational functions that transform input into output, and the logical sequencing between the dialogue and computation.

If you include transformations and sequencing in your dialogue definition, most of the application would be considered part of the dialogue. This definition does not support the separation necessary for effective user-interface management.

To decide where to draw the line between the dialogue and application, you should first consider what dialogue independence means.

Yet it is equally fruitless to define the dialogue in the bytes and bits of the lowest abstraction level. Here, everything the computer does is computation, except the raw I/O instructions that pass byte streams to and from a physical input device. At a slightly higher abstraction level, few would argue that certain computational functions, such as input parsing, belong anywhere other than in the dialogue.

Thus, the dialogue component does contain computation, but only computation that directly supports dialogue functions – to produce displays and extract valid input. I consider dialogue to include the computation and control sequencing for accepting, parsing, validating, and mapping tokens. It also includes user prompts, error and confirmation messages, and other displays directly associated with extracting user input. If you do not place these functions in the dialogue component, you cannot achieve dialogue independence.

Application semantics. The computational component is composed of the *functional* semantics of an application, also called *application* semantics. The latter term connotes knowledge of (access to definitions of) application objects (their types and attributes) and their behavior (transition functions for changing attribute values).

In linguistics, semantics is the study of the relationship between symbol and referent. Here, referents of interface symbols are the computational objects and operations. Thus, I use the term "semantics" loosely and interchange it with computation of application functions.

Semantics must be included in the user

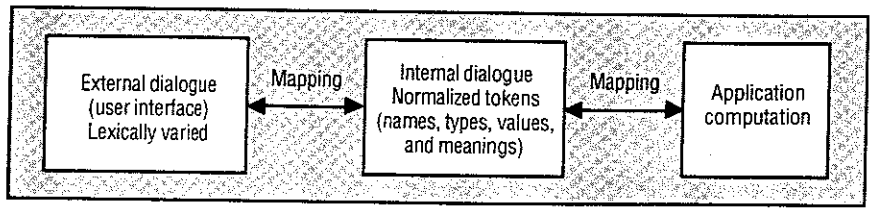


Figure 1. Simple view of dialogue-computation separation.