APPLICATIONS: A DIMENSION SPACE FOR USER INTERFACE MANAGEMENT SYSTEMS

Joëlle Coutaz, Sandrine Balbo

Laboratoire de Génie Informatique (IMAG)
BP 53 X, 38041 Grenoble Cedex, France
email: joelle@imag.imag.fr, balbo@imag.imag.fr

To appear in CHI'91 Proceedings

ABSTRACT

This article presents an abstract space of dimensions which characterize the behavior of applications (i.e. functional cores) with regard to UIMS components. These dimensions such as responsiveness, accessibility, and instantiability, constitute a conceptual framework which captures the notion of functional core in terms adequate for UIMS designers. The dimension space may also be viewed as a requirements list for designing new UIMSs as well as a set of criteria for evaluating existing UIMSs.

KEYWORDS: Application interface, UIMS, classification, dimension space.

INTRODUCTION

The terms *application* and *User Interface Management System* (UIMS) cover multiple meanings and different levels of software services. Generally speaking, an application corresponds to the whole interactive system with which a user can accomplish a set of tasks specific to a particular domain. In the context of software architecture, the application denotes the functional core of the interactive system: it is the code portion which implements domain dependent concepts in an essentially presentation independent way. In this article, an application denotes the notion of functional core.

Similarly, the term UIMS may designate simple run time kernels up to complete environments for designing and developing user interfaces. Whether an interactive system is built from sophisticated development tools or not, the interface between the functional core and the run time components of the user interface is a mandatory passage. Current practices in UIMS technology shows that this interface is still poorly designed.

The purpose of this article is to clarify the nature of functional cores from the perspective of UIMS designers and developers. We propose an abstract dimension space from which one can model the behavior of functional cores at their interface with UIMS components. Two interesting results follow from such a conceptual framework: The dimensions define a set of requirements for designing new UIMSs as well as a set of criteria for evaluating current UIMSs.

As any boundary between two communicating entities, the interface between functional cores and UIMS components implements a suitable protocol. A protocol is characterized by temporal strategies, the nature of data exchanged, and by some linking or correspondence mechanism. These three issues are successively discussed in the following sections. We will then apply our dimension space to the classification of a number of recent UIMSs which explicitly maintain an interface with functional cores.

TEMPORAL ISSUES

The transfer of information between two communicating entities such as the functional core and the UIMS components, must be coordinated over time. The coordination may be fully synchronous or fully asynchronous or may alternate between the two techniques.

Synchronous Coordination

Synchronous coordination implies that the sender waits for the receiver before its own processing can resume. In the context considered here, synchronous coordination models the "mutual servoeing" of the functional core and the UIMS: either one has the initiative but the initiator is directly controlled by its partner.

When considering interaction styles, synchronous coordination results in single threads of dialogue or, at best, in interleaved threads. A thread of dialogue represents a user's task or a subtask. The granularity of a task depends on the case at hand. In system terms, a thread corresponds to a command, or a set of logically connected commands which modify the state of the functional core. Interleaved threads of dialogue allow the user to undertake a new task before completing previous ones. However, at a given time, only one of such tasks can be carried out. Single threads and interleaved threads of dialogue are typical of editing tasks where the UIMS and the functional core take turns.

Asynchronous Coordination

Asynchronous coordination allows communicating entities to exchange information without waiting for each other. With such a communication scheme, the functional core and the UIMS are two equal partners sharing a common enterprise: that of accomplishing a task with the user.

As opposed to synchronous coordination, asynchronous coordination supports multiple concurrent threads of dialogue: the user may issue multiple commands simultaneously (using for example, a combination of voice and hands) while the functional core may have its own processing going on. Asynchronous coordination is well suited to the domain of process control where the UIMS and the functional core have multiple activities to carry on in parallel.

Combination of Synchronism and Asynchronism

Experience shows that many application domains need a combination of both types of coordination. For example, electronic mail is synchronized with the user while a message is being built. It behaves asynchronously when it signals the arrival of a new message. If, on user's request, the notification must be held back, two solutions may be envisioned depending on the behavior of the email functional core: If the functional portion behaves asynchronously, then the UIMS is notified of the new arrival and holds the notification; If a synchronous scheme is used, then the functional core must memorize the fact until the user requires the delivery.

Most of UIMSs such as Mickey [1], FormsVBT [2], UIDE [3], and MacIDA UIMS [4] support synchronous coordination only. In addition, most of them model the functional core as a semantic server: the UIMS has the initiative of the transfer of information (or, according to the classification of P. Tanner, dialog control is external [5]). Serpent, however, supports both techniques [6]. In Serpent, communication is performed through a shared data base which behaves like a blackboard: the functional core and the UIMS deliver or get information from the data base at their own rythm.

Note that synchronicity has been discussed as a dimension of a conceptual framework. As such, implementation issues are out of the scope of our analysis. In particular, we won't consider the underlying mechanisms for transferring the information (e.g. a common data base, function calls, message passing, etc.), nor will we suggest the existence of processes for dealing with asynchronism. Implementation issues are bound by the programming tools and by the services of the underlying platform. In this article, we will keep the discussion at the conceptual level.

NATURE OF DATA EXCHANGED

Data exchanged between a functional core and a UIMS is performed in terms of exported objects. The term *object* should be considered here in the general sense as an entity. It does not suppose any particular implementation technique such as a class, a function, or a shared data structure but covers all of them.

An *exported object* is an object that both the functional core and the UIMS understand. If we consider the definition of the functional core, an exported object is supposed to be presentation independent and to match the mental representation that the user has for a particular concept. Exported objects can be characterized by their type and their semantic purpose.

Type of data exchanged

The type of an exported object may be *elementary* or *constructed* depending on the nature of the concept to be conveyed to the user through the UIMS. The possibility of representing any concept, from simple up to compound information, may have a strong impact on the quality of the semantic feedback. In particular, if the UIMS supports simple data types only, then concepts which "naturally" match compound structures must be artificially decomposed by the functional core into elementary entities.

In order to reconstruct a representation adequate for the user, the UIMS must provide the user interface designer with an abstraction mechanism in the presentation layer. In the absence of such a mechanism, the "glue" must be explicitly programmed. Then, the next problem for the implementor is to identify the appropriate location for such coding. A number of UIMSs, such as FormsVBT, do not support compound exported objects. In addition, most of them do not provide any abstraction mechanism in the presentation layer nor do they convey a clear architectural model to guide the decision process and identify which portion of the interactive system should be in charge of the conceptual glue.

Purpose of data exchanged

The purpose of an exported object may be *specific* to the task domain or it may be of *general interest*. For example, the notion of paragraph, section, chapter, and document are specific to structured text editing. General purpose exported objects refer to notions that cognitive psychologists and human factors people have identified as useful for any interactive system. They include the concepts of:

- error,
- cut-copy-paste,
- undo, redo,
- procedural and contextual help,
- user (for groupware).

The semantic quality of errors, cut and paste, etc. involves both the UIMS and the functional core. So far, UIMS technology has provided very few hooks for the implementation of general purpose exported objects. In particular, UIMSs such as UIMX, Interface Builder [7], Jade from the Garnet environment [8], or Jacob's UIMS [9], which concentrate their functionality on presentation issues, do not deal with these problems. They cannot do so because they do not maintain any explicit knowledge of the functional core. As a result, cut and paste facilities, which are performed at the toolkit level, require the implementor of the functional core to program a type recasting mechanism.

On the other hand, UIMSs such as UIDE and MacIDA UIMS, based on an abstract description of the functional core, are able to maintain a dynamic model of the semantic state. This model provides the necessary foundation for the development of general purpose services. For example, UIDE is able to provide the user with contextual help [10], semantic error checking, and the undo feature. MacIDA UIMS automatically produces and dynamically maintains a graph which shows the user what has been done and what needs to be done to accomplish a task. This graph is a good example of how to support the evaluation stage and the elaboration of the plan of actions identified by Norman [11]. Although UIMSs driven by an explicit description of functional cores are good at semantic feedback, they are weak when considering the flexibility of the presentation layer (but this latter topic is out the scope of our discussion).

So far, we have identified two dimensions for our conceptual framework: synchronicity and the nature of data exchanged in terms of exported objects. We need now to analyze how exported objects relate to entities in the UIMS components.

MAPPING ISSUES

Exported objects model user's concepts. They define the state of the functional core in terms meaningful to the user but their implementation is presentation independent. We call *presentation object* a UIMS component whose purpose is to make an exported object perceivable. Again, the notion of presentation object covers any implementation technique ranging from simple interactors such as menus and radio buttons, up to complex hierarchies of abstractions such as the view controllers of Serpent or the component objects in Jacob's UIMS. In this section, we consider mapping issues between exported objects and presentation objects at the interface between the functional core and the UIMS. We have identified five topics of interest: direction, connectivity, instantiability, accessibility and responsiveness.

Direction

Direction refers to the direction of the transfer of information between an exported object and a presentation object. It may be a *static one-way correspondence* either from the functional core to the UIMS or the other way round; It may be a *two-way correspondence*. A correspondence between an exported object and a presentation object implies that the UIMS is able to automatically maintain the servoeing between those objects. For example, the notion of temperature, which is computed by the functional core, may correspond to a thermometer in the UIMS with a one-way correspondence going from the functional core to the UIMS. Each time the functional core modifies the value of the temperature, the UIMS should automatically reflect the modification to the thermometer. Although implementation issues are out of our discussion, servoeing between tokens and variables in Jacob's UIMS as well as constraint systems such as those embedded in Garnet and Serpent are interesting illustrations of the notion of direction. One-way constraint systems implement one-way correspondences whereas two-way constraint systems such as ThingLab [12] implement two-way correspondences.

Connectivity

Connectivity refers to the cardinality of associations between exported objects and presentation objects. Experience reveals two useful configurations: connectivity 1-to-n as shown in Figure 1 and connectivity n-to-1 as illustrated in Figure 3.

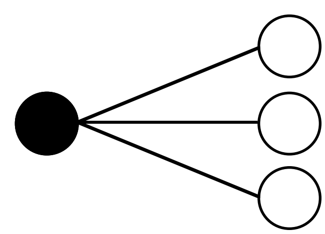

Figure 1: Connectivity 1-to-n between an exported object (black circle) and a set of presentation objects (white circles).

A connectivity 1-to-n may express the decomposition of a compound exported object into simpler units of presentation; it may also model the multiple representation of the same concept. For example, the concept of temperature may be presented to the user with two complementary views: a thermometer and a plot which respectively show the current value and the evolution of the temperature over time. The multiple representation of a concept implies that consistency be guaranteed between the views. The consistency may be maintained by the exported object or delegated to the UIMS. As shown in Figure 2, a specialized presentation object is connected to the exported object in a 1-to-1 correspondence and manages the multiple views. This solution has been adopted by Serpent with the notion of view controller.

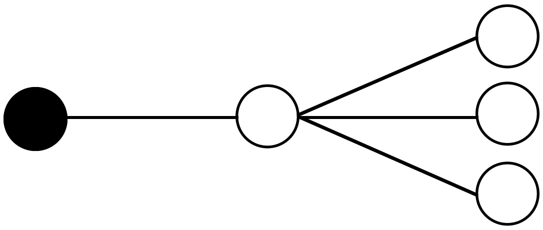

Figure 2: How to delegate the maintenance of view consistency in the UIMS.

Connectivity n-to-1 expresses the composition of multiple exported concepts into a unique presentation object. This presentation object, which defines a new abstraction inside the UIMS, is a convenient way of performing semantic enhancements or semantic repair: A set of exported objects may not individually match with the mental model of the user whereas their combination may be meaningful.

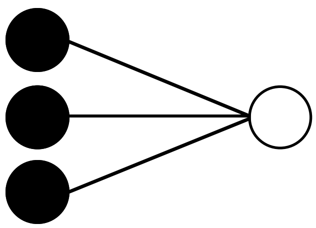

Figure 3: Connectivity n-to-1 between a set of exported objects (black circles) and a presentation object (white circle).

As an example of semantic enhancement, we will consider our own experience with the notion of Status-man. In one of our early implementations of an expert system for configuring Xerox Network Systems [13], we modelled the notion of error as a data structure containing the identification of the exported objects involved in the problem as well as a code to express the gravity of the situation. Error instances were then mapped into English (or French) sentences. Although the system would produce detailed relevent semantic feedback, an early test showed that the user needed a synthetic view of the current state of the system. To do so, we introduced a new presentation object, the status-man, whose purpose was to count the number of errors according to their level of gravity and deliver the results in a suitable form (see Figure 4). All of the error instances were connected to a single presentation object: the status-man.

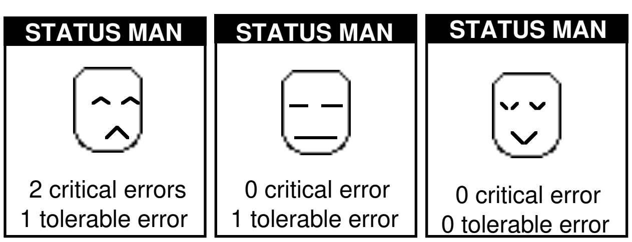

Figure 4: The status man as an example of semantic enhancement. The face is happy when no error has been detected, it is unhappy when fatal errors have occurred.

Most of UIMSs maintain 1-to-1 connectivities. As a result, multiple presentation of a concept, which is a frequent phenomenon and a useful feature, must be explicitly programmed. Again, in the absence of mechanism for abstraction in the presentation layer as well as no clear underlying architecture, such coding becomes a complex task. However, as explained above, Serpent provides the appropriate model for expressing and maintaining any type of connectivity.

Instantiability

Instantiability covers the dynamic creation and destruction of exported objects. Typically, computer aided design involves the dynamic creation and destruction of concept instances. The consequence for UIMSs is to dynamically maintain the correspondence between exported objects and presentation objects. Most of current UIMSs do not handle this phenomenon automatically. Serpent, however, provides the mechanism for dealing with instantiability.

In Serpent, exported objects are modelled as data structures in a shared data base. Slang, a special purpose Turing equivalent programming language, allows the user interface designer to associate exported objects with view controllers (i.e. presentation objects). The creation/deletion of an instance by the functional core is notified to the data base with special purpose primitives. Any insertion or removal of an instance in the data base is automatically detected by Serpent and echoed as the creation/deletion of the associated view controllers.

Accessibility

Accessibility expresses the semantic validity of the connection between exported objects and presentation objects. For example, the connection between the operator "delete" and the menu item "delete" is semantically valid if there exist at least one destroyable entity in the system.

An explicit expression of accessibility allows the UIMS to automatically produce an appropriate feedback and prevent the user from performing illicit actions. For example, the menu item associated with the delete operator would be dimmed until accessibility becomes legal.

Very few UIMSs handle the notion of accessibility. In UIDE, accessibility is expressed with the notion of mutually exclusive actions as well as in terms of pre- and post- conditions associated to object operators. MacIDA UIMS, uses an oriented graph which models the logical behavior of the functional core. Nodes represent exported objects (e.g. functions and parameters). A node is activated (i.e. valid) if all of the parent nodes are active.

Responsiveness

Responsiveness specifies the condition under which communication between exported objects and presentation objects can occur. An explicit expression of responsiveness can be used to control the granularity of semantic feedback.

Dimensions / UIMSs	coordin.		data exchanged				mapping						
			type		purpose		direction		connect.		instantiability	accessibility	responsiveness
	asynchronous	synchronous	simple	compound	specific	general	One-way	two-way	1-to-1	1-to-n			
Mickey	X					X		X		X	X	X	X
FormsVBT	X			X		X		X		X	X	X	X
MacIDA	X							X		X	X		X
UIDE	X							X		—			X
Serpent						X						X	X

Table 1: This table presents a classification of a number of recent UIMSs that explicitly model an interface with functional cores. Columns describe the dimension space whereas rows correspond to the set of UIMSs. An **X** in a cell indicates that the corresponding UIMS does not satisfy the criterion. A "**-**" in a cell indicates that the published description of the UIMS is not explicit with respect to the criterion.

For example, consider the task of a customer filling in an order. The form contains fields specific to the task including the customer's name. Concerning the validity of the customer's name, the designers of the system have two options: either check the validity as soon as the name is specified or perform the verification when the user validates the order form. In the first case, the user is provided with immediate semantic feedback, whereas with the second option, the order is processed in one burst.

To our knowledge, no UIMS explicitly embeds the notion of responsiveness although, in Serpent, it is possible to program responsiveness inside a view controller.

Table 1 summarizes our discussion for a number of recent UIMSs. Given the topic of interest, we have only considered those which maintain an explicit high level interface with functional cores.

CONCLUSION

In this article, we have considered the characteristics of the interface between functional cores and UIMSs. Our analysis relies on the hypothesis that a functional core and a UIMS communicate according to a protocol. Thus, the three components of a protocol define a sound basis for describing this communication. However, the general foundation provided by the notion of protocol must be refined into a set of requirements specific to user interface technology. These requirements have been identified from current practices in the construction of interactive systems. This analysis has led to the following dimension space:

- temporal coordination required by a functional core (synchronous, asynchronous),
- nature of data exchanged with a functional core (simple or compound, specific or general purpose), and
- mapping mechanisms (direction, connectivity, instantiability, accessibility, responsiveness).

As far as temporal coordination is concerned, if generality is a design requirement, then a combination of synchronous and asynchronous communication should be provided by the UIMS. With regard to the nature of data exchanged, generality and provision for general purpose facilities such as help, require UIMSs to support compound types as well as an explicit knowledge of the exported objects.

One-way direction is generally well handled in current UIMSs. On the other hand, connectivity, instantiability, accessibility and responsiveness need further study and development. In most UIMSs, these features must be programmed explicitly without any architectural guidance. One general approach to avoid this problem consists of providing the user interface designer with an abstraction mechanism over a clear architectural framework, as in the Serpent system.

Is our dimension space complete? Clearly, our analysis is restricted to the description of the interface between functional cores and UIMSs. It does not cover important issues in UIMS technology such as presentation and dialogue control. UIMSs considered in this article stress the explicit representation of the functional core at a high level of abstraction. At the opposite, other UIMSs such as Interface Builder, Garnet or Jacob's UIMS, put the emphasis on presentation issues. They are concerned by the flexibility and easiness for creating highly interactive graphical user interfaces. They are not concerned with an explicit knowledge of the functional core. Consequently, the design, selection, or evaluation of a UIMS involves two complementary analyses: that of the interface with the functional core and that of the presentation and dialog component. This article provides a framework for only the first component.

With regard to presentation and dialogue issues, we may consider how models imposed by "de facto standard" toolkits such as Motif [14] fit into UIMS technology. What abstraction mechanisms are required for delegating semantics in the presentation or for building new presentation objects? For example, the Garnet toolkit as well as Jacob's UIMS provide an abstraction mechanism while others, such as UIMX, do not. How far can we go with general purpose presentation objects such as the PAC agents identified in [15]? How do presentation objects support the properties that are desirable at the interface with functional cores? In particular, is the underlying model for presentation objects consistent with an asynchronous communication with the functional core? Jacob's interaction objects as well as gadgets in Garnet have been designed to support such asynchronicity. On the other hand, the callback procedure paradigm embedded in Motif widgets naturally leads to model functional cores as synchronous semantic servers. Finally, how should multimodal user interfaces be supported by UIMSs?

These opened questions suggest the definition of a second dimension space for classifying UIMSs according to their presentation and dialogue components. From there, we could usefully identify relationships between the two dimensions spaces. In particular, we could analyze the implications of a set of properties in the presentation layer on the properties of the interface with functional cores and vice versa.

ACKNOWLEGMENTS

This paper was influenced by stimulating discussions with our colleagues and observers of the IFIP WG 2.7: L. Bass, M. Beaudoin-Lafon, F. Bodart, G. Cockton, M. Harrison, I. Newman, P. Dewan, C. Unger, H. Stiegler, and R. Zobel.

REFERENCES

[1] Olsen, D.R. Jr. A Programming Language Basis for User Interface Management. In *Proceedings of the CHI'89 Conference* (Austin, 30 April-4 May). ACM,1989, pp. 171-176.

[2] Brown, M.H., Avrahami, G. and Brooks, K.P. A Two-View Approach to Constructing User Interfaces. In *Proceedings of SIGGRAPH'89* (Boston, 31 July-4 august). Computer Graphics, 23, 3 (July). ACM, 1989.

[3] Foley, J.D. Gibbs, C. Won Chul, K. and Kovacevic, S. A Knowledge-Based User Interface Management System. In *Proceedings of the CHI'88 Conference* (Washington, May 15-19). ACM,New York, 1988, pp. 67-74.

[4] Petoud, I. and Pigneur, Y. An Automatic and Visual Approach for User Interface Design. *In Engineering for Human-Computer Interaction.* G. Cockton Ed., Elsevier Science, 1990, pp. 403-420.

[5] Tanner, P. and Buxton, W. : Some Issues in Future User Interface Management Systems (UIMS) Development. In *User Interface Management Systems,* IFIP Working Group 5.2 Workshop on User Interface Management (Seeheim, Nov. 1983), Pfaff, G.E. (Ed), Springer Verlag,1985.

[6] Bass, L. Hardy, E. Hoyt, K. Little, R. and Seacord, R. *The Serpent run time architecture and dialogue model*. Carnegie Mellon Technical Report, CMU/SEI-88-TR-6, January, 1988.

[7] Webster, B.F. *The NeXT Book*. Addison Wesley, 1989.

[8] Myers, B. Giuse, D. Danneberg, R.B. Vander Zanden, B. Kosbie, D. Pervin, E. Mickish, A. and Marchal, P.*Garnet, Comprehensive Support for Graphical, Highly Interactive User Interfaces*. IEEE Computer, 23,11 (Nov. 1990). IEEE Computer Society, November, 1990, pp. 71-85.

[9] Jacob, R.J.K. *A Specification Language fro Direct-Manipulation User Interfaces*. ACM Transactions on Graphics, 5, 4 (Oct. 1986). ACM, October, 1986, pp. 283-317.

[10] Senay, H. Sukaviriya, P. and Moran L. Planning for Automatic Help Generation. *In Engineering for Human-Computer Interaction.* G. Cockton Ed., Elsevier Science, 1990, pp. 293-308.

[11] Norman, D. A. and Draper, S. W. *User Centered System Design*. Lawrence Erlbaum Associates, 1986.

[12] Borning, A.H. Defining Constraints Graphically. *In Proceedings of CHI'86* (Boston, April 13-17). ACM, New York,1986, pp. 137-143.

[13] Coutaz, J. *Interface Homme-Ordinateur: Conception et Réalisation*. Dunod, 1988.

[14] OSF/Motif, Programmer's Reference Manual, Revision 1.0 ; Open Software Foundation, Eleven Cambridge Center, Cambridge, MA 02142, 1989.

[15] Coutaz, J. and Nigay, L. Agents and Rules. In *AMODEUS D2 Deliverable,* BRA 3066 ESPRIT project, 1990.