ARCHITECTURAL DESIGN FOR USER INTERFACES

Joëlle COUTAZ

Laboratoire de Génie Informatique (IMAG)
BP 53 X, 38041 Grenoble Cedex, France
fax: (33) 76 44 66 75, tel. (33) 76 51 48 54
email: joelle@imag.fr

Summary
This article discusses software architecture modelling for the design and development of interactive systems. Software architecture models should support properties and constraints prescribed by the software and cognitive psychology communities. From these observations, we derive a multi-agent framework applicable to the software design of interactive systems. An example shows how this general model can be engineered to fit practical problems. We then discuss the ability of the model to support a number of properties and constraints relevant to software engineering.

Keywords
multi-agent architecture, user interface, software architecture.

1. Introduction

Rapid improvements in computer technology have increased the opportunities for developing highly interactive user interfaces. At the same time, progress in psychological studies and human factors have increased the potential for designing high quality user interfaces. These advances, however, have not alleviated the task of the user interface software designer. Requirements continue to overpass current practices. In order to fully exploit innovation in a systematic way, new and clear software architectures are needed.

This article discusses software architecture modelling for the design and development of interactive systems. Software architecture models should support properties and constraints prescribed by the software and cognitive psychology communities. A number of software properties and constraints are presented in section 2. From these observations, we then propose a framework useful for organizing interactive software (Section 3). In Section 4, a complete example is used to explain how the notions conveyed by the model can be engineered to fit a practical problem. In the last section, we analyse the ability of the model along the dimensions defined in section 2.

2. Properties and Constraints

Properties are the idealized characteristics which define the dimensions of a design space. They can be judiciously exploited to define design trade offs. Constraints are imposed attributes which reduce the design space. In general, they cannot be modified. The intent of this section is to enumerate a number of salient properties and constraints related to human computer interaction (HCI) and software engineering. Although the list is not exhaustive, it serves as a basis for evaluating our architecture model and for presenting our terminology.

HCI properties that we will consider here include: the control of the interaction, the notions of feedback and thread of dialogue; As for software engineering properties, we will pay attention to performance, concurrency, reusability, modifiability and portability. The platform used for software development and the existence of the functional core are two interesting constraints.

2.1. HCI properties

Control of the interaction. In HCI, one makes the distinction between system-driven interaction and user-driven interaction. In system-driven interaction, the user must respond to questions from the system. In user-driven interaction, the user is free to choose among a number of currently valid options.

Feedback covers expressions produced by the system in response to user's actions. It may be semantic (i.e. domain dependent) or syntactic and lexical (i.e. presentation dependent).

A *thread of dialogue* denotes some user's task. A dialogue is multi-thread when the user is able to undertake multiple tasks in parallel. Threads are interleaved when multiple tasks can be initiated but only one of them is processed at a time.

2.2. Software engineering properties and constraints

Performance. Response time of the system must match the expectation of the user. For example, datagloves require that the visual representation of the hand on the screen matches the physical hand gesture "instantaneously". Even if the user is capable of adjusting hand and finger movements to the slowness of the system, interaction becomes impossible [Sturman 89]. Clearly, performance is crucial to the usability of the system.

Concurrency is required to support parallism at the user's level: multiple physical actions may be performed simultaneously on a variety of devices and multi-thread dialogue are made possible.

Reusability is a key element in the evaluation of the development cost: Experience shows that the user interface portion of an interactive system represents up to 80% of the whole code. The ability to reuse and to port existing user interface components is thus an important feature.

Modifiability is a necessary property of an interactive system since good user interface design relies necessarily on iterative refinement [Norman 86].

The *development platform* includes the reusable software imposed to the implementer, such as the underlying operating system and tools fro building user interfaces. It is often the case that design decisions must be revisited to take into account the limiting factors of the platform.

Existence of the functional core. An interactive system is composed of two components: the functional core (FC) and the user interface system (UIS). The FC implements domain dependent concepts such as the notions of document, chapter, section, etc., in a document editor. The UIS implements the interaction between the user and the FC. For example, the UIS of the document editor presents the editing commands as well as the document in a WYSISYG manner. (We call UIMS, the user interface tool that helps in building a UIS). The existence of the FC may constrain the properties of the UIS.

The properties and constraints considered so far result from the analysis of the interaction between a human and a computer system. When considering architectural issues, this analysis leads to the following remarks: there are concepts (those modelled by the functional core) and presentation objects (those perceivable to the user and implemented in the UIS). Concepts and presentation objects must be linked by some mechanism. Since they serve different purposes, they may not use the same formalism. In addition to mapping functions, there must be some transformation mechanisms which smoothly bridge the gap between concepts and presentation objetcs. These preliminary observations are refined in the following section.

3. Overall Organization of an Interactive System

Figure 1 shows a general model of an interactive system based on the Seeheim model [Pfaff 85]. One salient property of this model is the symmetry organized around a key component: the dialogue controller. At the two extremes, the functional core and the user play a symmetric role at a high level of abstraction: both of them manipulate task domain concepts. Next, come two interface components: the functional core adaptor (the FCA) and the presentation component (the PC). A second salient property of the model is the multi-agent architecture of the dialogue controller.

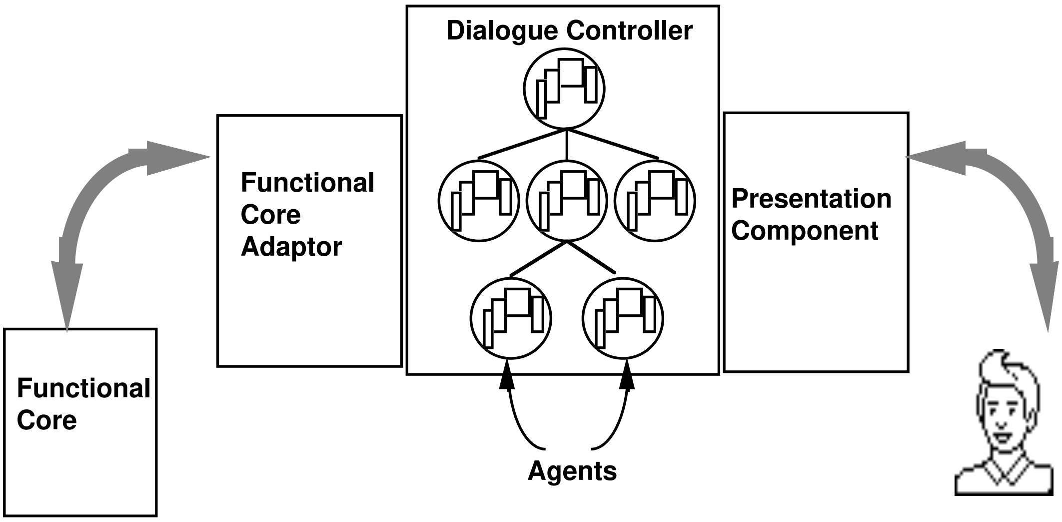

Figure 1: The basic components of an interactive system.

In the following paragraphs, we first consider the two extreme components of the model (the user and the functional core), then we discuss the two interfaces (the functional core adaptor and the presentation component) and close with the key element (the dialogue controller).

3.1. The user and the functional core

The user and the functional core both produce and consume information through the dialogue controller via the interface components. This symmetric view of the general functioning of an interactive system where the key component, the dialogue controller, is equally driven by the functional core and the user does not impose any model on the control of the interaction. The question of external VS internal VS mixed control [Tanner 83] is irrelevant at this level of description. Clearly, the choice between the alternatives is guided by the case at hand.

3.2. The functional core adaptor

The functional core adaptor (FCA) serves as a buffer between the dialogue component and the domain specific concepts implemented in the functional core. It is designed to absorb the effects of change in its direct neighbors. As any boundary, it implements a protocol. A protocol is characterized by temporal strategies, by the nature of data exchanged and by some linking mechanisms [Coutaz 91a].

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

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 control of the functional core and the UIS: either one has the initiative but the initiator is directly controlled by its partner. Asynchronous coordination allows communicating entities to exchange information without waiting for each other. With such a communication scheme, the functional core and the UIS are two equal partners sharing a common enterprise: that of accomplishing a task with the user.

When considering interaction styles, synchronous coordination results in single threads of dialogue or, at best, in interleaved threads. 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.

Nature of data exchanged
Data exchanged between the functional core and the UIS is performed in terms of conceptual 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.

A conceptual object is an entity that the designer of the functional core wishes to make perceivable to the user. Consequently, it is an object that both the functional core and the UIS understand. It belongs to the FCA. Ideally, a conceptual object is the electronic media independent representation of a domain dependent concept. Conceptual objects can be characterized by their type and their semantic purpose.

The type of a conceptual object may be elementary or constructed depending on the nature of the concept to be conveyed to the user through the UIS. The purpose of a conceptual object may be specific to the task domain or it may be of general interest. For example, the notions of paragraph, section, chapter, and document are specific to structured text editing. General purpose conceptual 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.

So far, we have identified two dimensions for the FCA: concurrency and the nature of data exchanged in terms of conceptual objects. We need now to analyze how conceptual objects relate to entities in the UIS components.

View Mapping
A presentation object is a UIS component whose purpose is to make a conceptual object perceivable to the user by using a combination of the human perceptual systems: auditory, visual, tactile systems, and eventually smell when the technology will be available. View mapping defines the correspondence between conceptual objects and presentation objects. This correspondence can be characterized along five axes: direction, connectivity, instantiability, accessibility, and responsiveness [Coutaz 91a].

Direction refers to the direction of the transfer of information between a conceptual object and a presentation object. It may be a one-way correspondence either from the functional core to the UIS or the other way round; It may be a two-way correspondence. A correspondence between a conceptual object and a presentation object implies that the UIS is able to automatically maintain the servoeing between those objects.

Connectivity refers to the cardinality of associations between conceptual objects and presentation objects. Experience reveals two useful configurations: connectivity 1-to-n and connectivity n-to-1. Connectivity 1-to-n may express the decomposition of a compound conceptual 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. Connectivity n-to-1 expresses the composition of multiple conceptual concepts into a unique presentation object.

Instantiability covers the dynamic creation and destruction of conceptual objects. Typically, computer aided design involves the dynamic creation and destruction of instances of concepts. The consequence for the UIS is to dynamically maintain the correspondence between conceptual objects and presentation objects through the FCA.

Accessibility expresses the semantic validity of the connection between conceptual objects and presentation objects. For example, the connection between the function "delete" implemented in the functional core and the menu item "delete" is semantically valid if at least one destroyable conceptual object exists in the system.

Responsiveness specifies the condition under which communication between conceptual objects and presentation objects can occur. Active data react immediately on changes. In some circumstances, immediate reaction need to be delayed.

So far, we have defined the dimension space at the boundary between the functional core and the UIS. This dimension space identifies the properties of the FCA. We need now to investigate the second interface of the model: the presentation component.

3.3. The presentation component

The presentation component implements the physical interaction with the user via hardware and software. It includes a set of presentation objects that defines the “image” of the system [Norman 86], i.e. the perceivable behaviour of the interactive system.

The presentation component is the location for handling low level multi-media events (for input as well as for output). For example, low level input events produced by voice and gesture input devices may be time-stamped and queued in the same way mouse events and keyboard events are currently processed by windowing systems. Multi-media events are combined into higher level events in the same way keyboard and mouse events are currently combined into higher abstractions by presentation objects.

3.4. The dialogue controller

The dialogue controller is the keystone of the model. Contrary to a number of user interface models, the dialogue controller is not a monolithic "obscure potato"! Instead, it is organized as a set of cooperating agents which smoothly bridges the gap between the functional core adaptor and the presentation component. An agent is a complete information processing system [Bass 91, Coutaz 90]. The nature of the agents involved in a particular interactive system and their relationships will be further described in paragraph 3.4.2. For now, we need to justify the overall functionality of the dialogue controller.

The overall functionality

The dialogue controller has the responsibility for task-level sequencing. Each task or goal of the user corresponds to a thread of dialogue. This observation suggests the choice of a multi-agent architecture. An agent or a collection of cooperating agents can be associated to each thread of the user's activity. Since each agent is able to maintain its own state, it is possible for the user (or the functional core) to suspend and resume any thread at will.

The dialogue controller receives events both from the functional core via the functional core adaptor (FCA), and from the user via the presentation component (PC). Bridging the gap between an FCA and a PC has a number of consequences. In addition to task sequencing, the dialogue controller must perform data transformation and data mapping:

1) A FCA and a PC use different formalisms. One is driven by the computational considerations of the functional core, the other is toolkit/media dependent. In order to match the two formalisms, data must be transformed inside the dialogue controller.

2) State changes in the FCA must be reflected in the PC (and vice versa). Therefore links must be maintained between FCA conceptual objects and PC presentation objects. A conceptual object may be rendered with multiple presentation techniques. Therefore, consistency must be maintained between the multiple views of the conceptual object. Such management is yet another task of the dialogue controller.

The dialogue controller at multiple resolution

Bridging the gap between the FCA and the PC covers task sequencing, formalism transformation, and data mapping. Experience shows that these operations must be performed at multiple levels of abstraction and distributed among multiple agents.

Levels of abstraction reflect the successive operations of abstracting and concretizing. Abstracting combines and transforms events coming from the presentation objects into higher level events until the FCA is reached. Conversely, concretizing decomposes and transforms high level data from the FCA into low level information. The lowest level of the dialogue controller is in contact with presentation objects. Since agents should carry task sequencing, formalism transformation, and data mapping at multiple levels of abstraction, it is tempting to describe the dialogue controller at multiple grains of resolution combined with multiple facets.

At one level of resolution, the dialogue controller appears as a "fuzzy potato". At the next level of description, the main agents of the interaction can be identified (see the example described in Section 4). In turn, these agents are recursively refined into simpler agents. This description applies the usual abstraction/refinement paradigm used in software engineering. Figure 1 shows the recursive description of the dialogue controller at multiple grains of resolution.

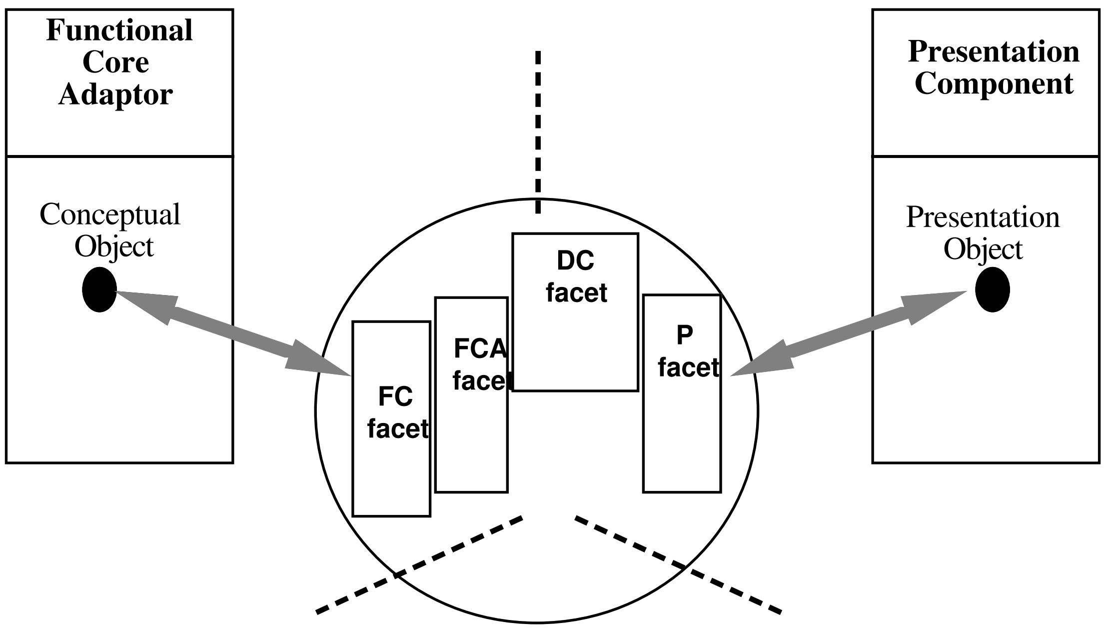

Figure 2: An agent of the dialogue controller. Dashed lines represent possible relationships with other agents. Dimmed arrows show the possible links with the surrounding components of the agent.

In addition to the refinement/abstraction axis, we introduce the "facet" axis. Facets are used to express the different but complementary and strongly coupled computational perspectives of an agent. These perspectives are similar to those identified for the whole interactive system:

• the functional core facet (FC facet) defines the competence of the agent in the chain of abstracting and concretizing. It may be related to some conceptual objects in the functional core adaptor;

• the functional core adaptor facet (FCA facet) performs formalism translation between the FC facet and the dialogue controller facet of the agent;

• the dialogue controller facet (DC facet) controls event sequencing inside the agent, maintains a mapping between the FCA facet and the presentation facet of the agent;

• the presentation facet (P facet) is involved in the implementation of the perceivable behaviour of the agent. It is related to some presentation object in the presentation component.

Figure 2 shows an agent of the dialogue controller and its relationships with the surrounding components. The model presented so far is kept general intentionally. The example described in the following section shows how it can be applied to a practical problem.

4. An example

Let us consider a tourist information system. As illustrated in Figure 3, this system splits the screen into two parts. A schematic view is used to display the map of the whole tourist area at low resolution (details such as town names and churches, are not visible). The detailed view contains a close-up view of a subset of the schematic view determined by a movable rectangle. As the user moves the rectangle with the tracker-ball, the detailed view is updated accordingly.

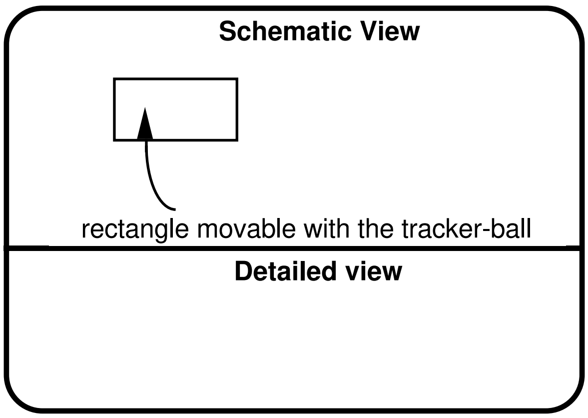

Figure 3: The split screen of the tourist information system.

In the following paragraphs, we describe the steps involved in the software design process of the tourist information system. We show how software design considerations can be combined to the principles advocated by our architecture model. More details on the software design rules can be found in [Coutaz 91b].

4.1. Step 1: Identifying conceptual objects and view agents

The tourist information system must model the notions of road, site of interest, name, etc. as well as their relationships. As shown in figure 4, the relationships between these elements can be modelled as a semantic network. This network, which is an abstract media-independent representation of the map, is maintained by the functional core portion of the interactive system. This functional core is considered here to be a data base server.

The purpose of a data base system is to express logical relationships between units of information such as “road R connects towns A and B”. On the other hand, the notions of “close-up” and “schematic” views, which are user interface issues, may not be modelled in the data base. If so, an intermediary data structure, adequate for the automatic generation of the two complementary views, must be defined. This structure acts as a perspective on the tourist map maintained in the functional core. It adapts the content of the functional core to the need of the UIS. Therefore, it belongs to the functional core adaptor (see Figure 4).

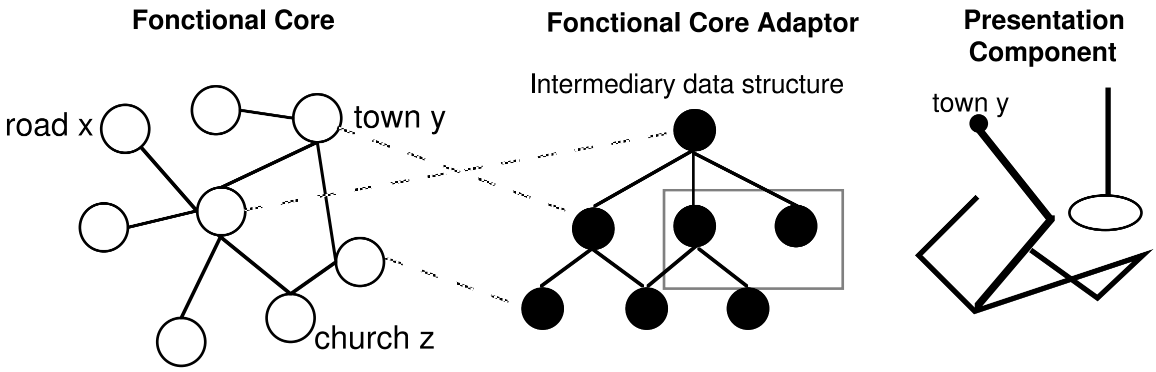

Figure 4: An intermediary data structure maintained in the FCA as the interface between the data-base world and the graphics tools. The rectangle shows the subset of the intermediary data structure that is currently displayed in the detailed view. Dashed lines between the FC and the intermediary elements represent the mapping maintained through the FCA.

So far, we have considered the abstract conceptual side of the interactive system. At the other end of the spectrum, we observe that the tourist information system provides the user with two different views of the map. Since a view acts as an interlocutor of the user, each view should be implemented as an agent: Schematic-View and Detailed-view, which model the schematic diagram area and the close-up detail area respectively (see Figure 5).

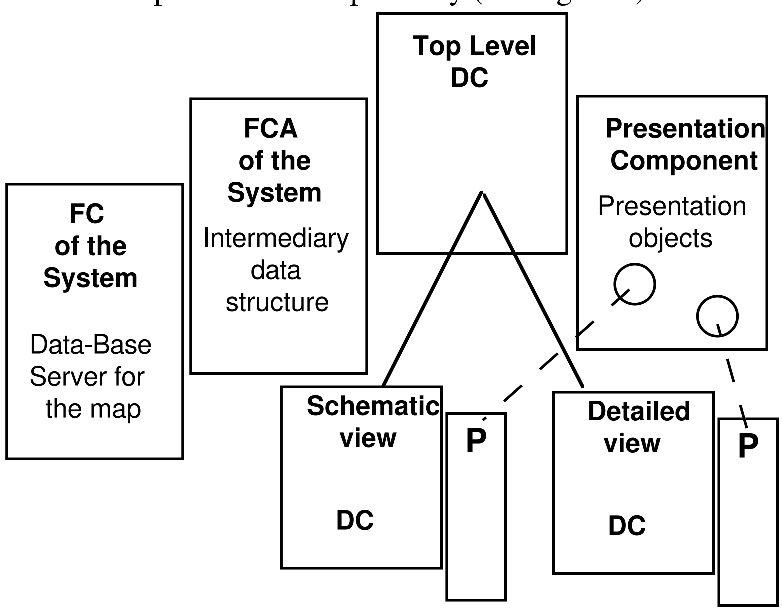

Figure 5: Step 1: Defining one agent per view. Dashed lines indicate the links with the presentation objects. Plain lines denote the expansion of the DC of the system.

According to the model, an agent is modelled along multiple perspectives. In particular:

• The P facet of Schematic-view is linked to a presentation object implemented in the presentation component of the system. This presentation object interprets graphics primitives to generate the visual representation of the map as well as the rectangle attached to the displacement of the tracker-ball. It also interprets input events when the tracker-ball is moved (see 4.5 for more details).

• Conceptually, the FC facet of Schematic-view is the intermediary representation of the map: Schematic-view is supposed to produce an image from the intermediate data structure designed as one step towards rendition. Since this representation holds for Detailed-view as well, it must be factored out (*software design criteria: memory resource sharing*). As advocated above, the FCA of the interactive system is the good candidate for maintaining such sharing.

- The DC facet of Schematic-view translates primitives received about the intermediary representation into graphics primitives interpreted by the associated presentation object.

- Since the FC facet of Schematic-view is empty, its FCA is empty.

An identical design process holds for the definition of Schematic-view. At this stage of the design, we can observe the abstraction/concretization/transformation phenomenon between the abstract representation of a map in the functional core, the intermediate media-independent data structure of the FCA suited for rendering, and the graphical representation of the view agents whose interpretation produces perceivable information.

4.2. Step 2: Introducing control agents

The functions of Detailed-view and Schematic-view are tightly coupled: the first one shows a close-up subset of the second one. Therefore a Multiple-view agent must be introduced to control their relationships. By doing so:
- automatic visual consistency is maintained between the two views, and
- Schematic and Detailed views ignore their mutual existence (*software design criteria: reusability*). (If the views referenced each other directly, they would not be reusable independently.)

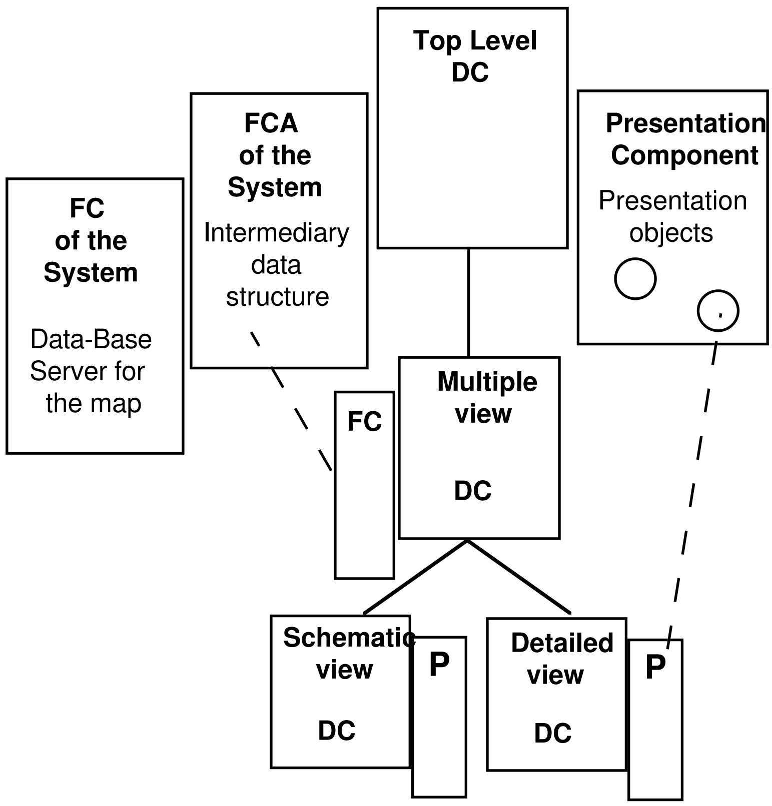

Figure 6: Step2: Introducing a multiple view agent to maintain visual consistency. Dashed lines denote links.

Since Multiple-view controls the multiple rendering of a map, its FC facet becomes a good candidate for maintaining the intermediary representation. There is a conflicting design decision, here: conceptually, the intermediary representation belongs to both the FCA of the interactive system and the FC facet of Multiple-view. Having an explicit interface with the functional core is a strong principle. Thus, the intermediate representation should explicitly sit

in the FCA of the interactive system. As for the FC facet of Multiple-view, the alternative is the following:
1) replicate the intermediary representation,
2) download portions of the intermediary representation and use it as a cache in the UIS,
3) access the intermediary representation when needed.

Clearly, solutions 1 and 2 minimize information transfer and ameliorate response time but require more resource memory. Although this is not the case in our example, they also imply the maintenance of semantic consistency. Solution 3 should be preferred if all the agents share the same address space. In our design, we will suppose that solution 3 is adequate: for example, the FC facet of Multiple-view contains pointers to the intermediate data structure of the FCA of the interactive system.

The DC facet of Multiple-view maintains the consistency between the two views it controls. Given the definition of the FC facet, there is no need for an FCA facet. The P facet is empty since Multiple-view is a controller with no direct representation on the screen. Figure 6 shows the current output of the software design process.

4.3. Step 3: Factoring out functionality

We observe that the FC and FCA facets of Schematic-view and Detailed-view are empty (functions already factored out in the higher levels of abstraction of the system). We also observe that the DC facet of Schematic-view is functionally identical to that of Detailed-View. Both contain the same operator (i.e. layout) with the intermediary structure and the level of detail as inputs. Thus, the DC facet of the two view agents can be factored out into Multiple-view. Since, the view agents are now reduced to a P facet, they can be converted into two presentations for the Multiple-view agent (See Figure 7).

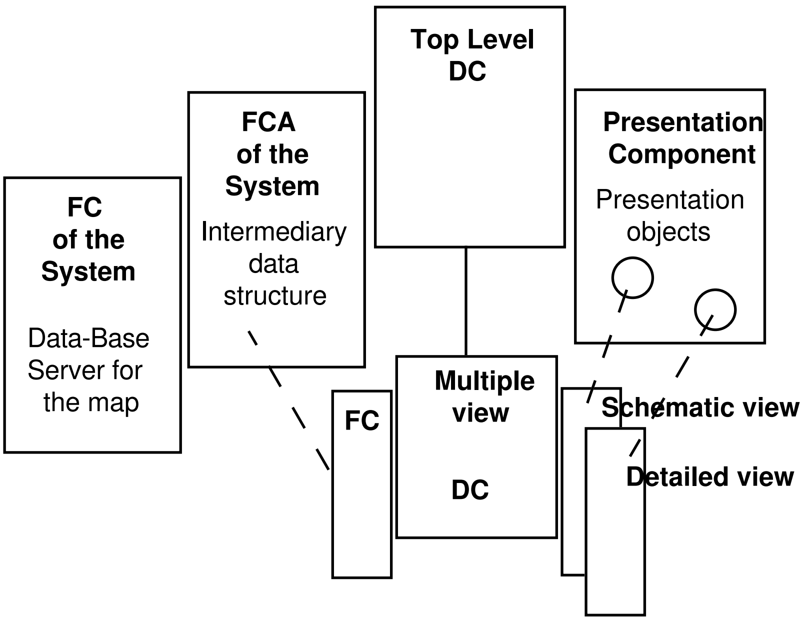

Figure 7: Step3: Converting the two view agents into two P facet for the Multiple-view agent.

4.4. Step 4: Bundling Agents

An agent of the dialogue controller may control multiple agents or perform data transformation (abstraction/concretization). When an agent controls a single subagent, then the functionalities of these agents may be combined. Since the DC of the system is controlling one single agent (i.e. Multiple-view), the functionalities of DC and of Multiple-view can be bundled up into one unit. Figure 8 shows the final architecture.

4.5. The final architecture

As shown in Figure 8, the tourist information system consists of the following components:

- a functional core, FC, for modelling the notion of map at a high level of abstraction,

- a FCA for adapting the abstract FC map to the UIS requirements,

- a DC which 1) includes a translation mechanism for switching between the formalism used in the FCA and the formalism used in the presentation component; 2) maintains visual consistency in the presentation component.

- a presentation component made of two presentation objects: Schematic and Detailed, corresponding to the schematic diagram area and the close-up detail area respectively.

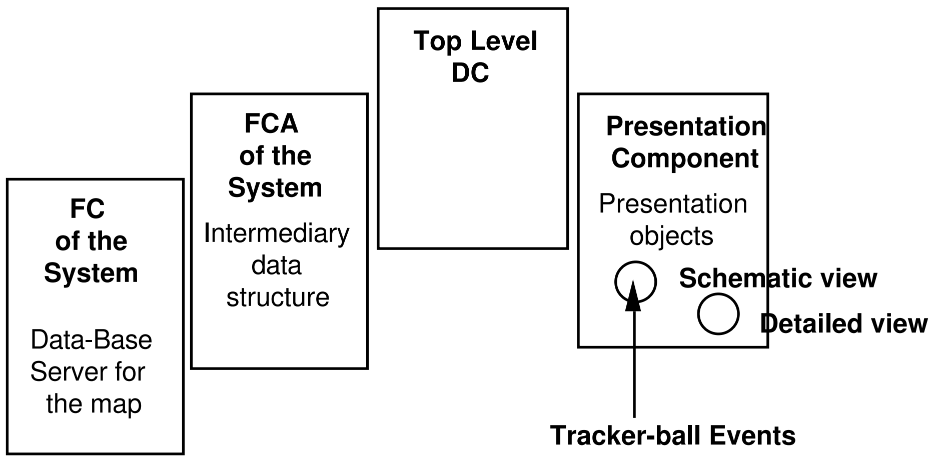

Figure 8: One possible architecture for the tourist information system.

We are now able to describe the information flow as the user acts on the tracker-ball to select a new area of interest:

- When receiving a tracker-ball event, Schematic view computes the relative displacement of the focus of interest (dx, dy). The software designer must then make a choice between two possibilities:
  a) either Schematic notifies DC of the traker-ball event, and DC asks Schematic to move the rectangle,
  b) or Schematic processes the event (it moves the rectangle), then notifies DC of the state modification.

Solution b) is preferable in terms of message passing but solution a) is more appropriate in terms of semantic feedback. In b), Schematic immediately reacts at the lexical level without waiting for potentially useful computationnal feedback: Schematic's behaviour is wired. Since Schematic's behaviour is semantic independent, we will opt for solution b).

- Schematic informs DC of a “new focus”. DC maps the parameters of the focus into the new portion of the intermediate data to be shown;

- DC gets the new portion from the FCA, transforms the FCA description into graphics primitives;

- Graphics primitives are sent to the Detailed presentation object which then is able to redraw the new close-up view.

As discussed in the following section, this design solution may be modified by the properties and constraints of the development platform.

5. Architecture modelling with constraints and properties

This section is concerned with the impact of properties and constraints on the design process. When relevant, the discussion will be illustrated with the design example of section 4.

5.1. The control of the interaction

The balance between the functional core and the user does not imply any scheme for controlling the interaction. This is in constrast with earlier interpretations of the Seeheim model which imposed an *a priori* scheme on the control of the interaction. First generation UIMS's, inspired by the compiler technology, found it very convenient to view the functional core as a semantic server [Hayes 85, Jacob 84, Olsen 83, Schulert 85, Wasserman 85, Olsen 89, Petoud 89]. This external control of the interaction was adequate for domains such as graphics and text editing; it was inappropriate for control process systems where the functional core may produce information in an asynchronous manner from the point of view of the user (e.g., alert messages in a nuclear station security system).

In the tourist information system, the interaction is driven by the user. Thus, an external control is desirable.

5.2. Concurrency

The model refers to concurrency at two levels: in the functional core adaptor with the notion of protocol, and in the dialogue controller with the notion of agents. The model is general enough to support both synchronous and asynchronous communication.

Concurrency at the dialogue controller level makes it possible to implement concurrency at the user's level. For example, in the case of the tourist information system, concurrency at the user's level would allow simultaneous verbal requests (such as "show me this") and physical actions (such as the selection of the location of interest on the screen). Concurrency at the FCA level, allows for parallel processing between the functional core and the UIS.

The development platform may not provide facilities for the expression of parallelism (e.g. the Macintosh system). Then, in order to handle concurrency between the functional core and the UIS, the UIS is constrained to periodically poll the functional core. This functionality may be handled by the DC facet of the system (i.e. the top level DC).

In the domain of user interface, concurrency control has a strong impact on the usability of the system. As discussed in [Robertson 89], agents in animation systems have different time constants and vary in their computational requirements. If the underlying platform is not able to support these variations, then some animated objects may jump instead of showing a smooth evolution over time, or they may appear too fast or too slow. Again, a top level DC facet can be implemented to control computational resources between the agents of the UIS.

5.3. Reusability

The model stresses reusability in a number of ways. First, with the notion of adaptors, then through a judicious organization of agents. The advantages of the adaptors will be developed in the next paragraph. Cement and multiple view agents are good examples of reusability.

The cement agent controls a number of sibblings (say a palette and a scratch area). It behaves like a syntax analyzer which controls the local automata maintained in its subagents. By doing so, input for syntax analysis may come in any order from multiple sources. If, in addition, the cement agent is able to handle the construction of several commands simultaneously, it provides an easy way to implement multi-thread dialogues.

By doing so, none of the sibblings has a privileged role over the others and they need not be aware of each other. Being unaware means that they do not send any message to each other. Being aware would make them dependent; They would not be reusable individually but as a cluster only. The multi-agent framework provides a natural way for satisfying agent reusabilty through the use of intermediary control agents.

Similarly, the multiple view agent is driven by reusability considerations. Agents which implement the views of a concept do not need to know each other. Considering that the views are created dynamically by the user, it may be difficult for the view agents to track their alter ego. Consistency maintained by a special control agent solves the problem in a simple way.

5.4. Modifiability

The model suports modifiability in multiple ways: the existence of adaptors and the possibility for semantic enhancement.

Adaptors
The two adaptor components, the functional core adaptor and the interaction toolkit adaptor, enhance modifiability and portability. They alleviate the constraints imposed by the functional core and the interaction toolkit available from the development platform.

The functional core adaptor defines a clear interface with the functional core. As long as the modification of the functional core does not affect this interface, the rest of the system is left unchanged. If the interface is to be modified, then the modification is necessarily concentrated in one location. This centralization facilitates the maintenance of the interactive system and limits development cost.

As shown in Figure 9, the interaction toolkit adaptor defines a virtual toolkit used for the expression of presentation objects. This expression is then mapped into the formalism of the actual toolkit used for a particular implementation. Switching to a different toolkit requires rewriting the mapping rules but the expression of the presentation objects remains unchanged.

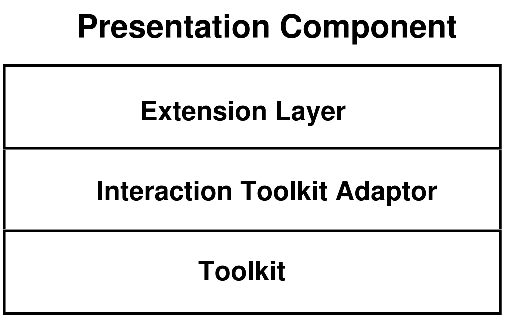

Figure 9: Details of the Presentation Component.

Presentation objects are generally constructed from entities made available in interaction toolkits. In general, interaction toolkits such as the X Intrinsics [OSF 89], provide an abstraction mechanism for defining new presentation objects. However, it is not always possible to build new presentation objects from the predefined building blocks of the toolkit. For example, in an earlier version of the X intrinsics, widgets (i.e. presentation objects) would occupy rectangular areas only. In such conditions, the notion of a wall in a floor plan drawing editor, could not be implemented as a diagonal line widget. Instead, a presentation object "wall" would be defined as a new abstraction outside the toolkit.

This example shows that the presentation component should be structured into three layers. Specific presentation objects which can be built from the building blocks of the toolkit belong to the toolkit. Those which cannot be built with the toolkit are part of a third layer, the extension layer. The interaction toolkit adaptor (ITA) such as XVT [Valdez 89] defines the boundary between these two layers.

An ITA defines a policy for the acquisition of user events. An event may or may not be automatically dispatched to the presentation object where the event occurred. If the tourist information system is to be implemented on top of an ITA with automatic dispatching, then Schematic receives tracker-ball events (as in section 4). At the opposite, if events are not dispatched, a special DC facet should be implemented for this purpose. In general, the top level DC of the system is a good candidate for event acquisition whether such events come from the FCA or from the ITA. In the example of the tourist information system, events would be acquired by the top level DC. This DC would then determine the new focus in the FCA, notify Detail to update its content as well as Schematic to move the rectangle at the appropriate location.

Semantic Enhancement
The notion of conceptual object in the functional core adaptor may be exploited to perform semantic enhancement. A conceptual object is supposed to match the mental representation that the user elaborates about a particular domain concept. It may be the case that the functional core, driven by software or hardware constraints, implements a domain concept in a way that is not adequate for the user. In the case of the tourist information system, the notion of “detail/schematic” was not modelled.

Semantic enhancement may be performed in the FCA by defining conceptual objects which reorganize the information of the functional core. Reorganizing may take the form of aggragating data structures of the functional core into a single conceptual object or, conversely, segmenting a concept into multiple conceptual objects. It may also take the form of an extension by adding attributes (such as “detail/schematic”) which can then be exploited by the UIS.

5.5. Performance

Performance may be enhanced through semantic delegation. The semantic quality of feedback may require frequent round trips with the functional core. This long chain of data transfer may be costly with respect to time. Therefore, it may be inconsistent with the expectation of the user. Semantic delegation, which consists of down-loading functional core knowledge into the user interface is a way to reduce transmission load. In particular, if the functional core is implemented as a distinct process running on a distinct processor, it may be judicious to use the FCA as a local cache of the functional core.

Semantic delegation may be performed in the dialogue controller as well. Functional core facets of agents provide a natural means for maintaining domain dependent information at multiple levels of abstraction. In the example of the tourist information system, semantic delegation could have been exploited with the FC facet of the Multiple-view agent.

5.6. Feedback

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 UIS 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 UIS 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 [Brown 89], do not support compound conceptual 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.

An explicit expression of accessibility in the FCA allows the UIS 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.

An explicit expression of responsiveness in the FCA can be used to control the granularity of semantic feedback. For example, consider the task of a customer filling up 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.

6. Conclusion

Despite the development of a large number of UIMSs and interaction toolkits, there is currently no satisfactory product. In particular, automatic generation of user interfaces may be applied in only a limited number of cases. As a result, software designers still have to face the difficult problem of implementing the missing pieces or overloading the inadequate portions produced with user interface tools. The architecture model and the engineering issues presented in this article address the shortcomings.

Acknowledgements

This paper was influenced by stimulating discussions with our colleagues and observers of the IFIP WG 2.7, and with members of the AMODEUS project: G. Abowd, Bala, L. Bass, M. Beaudouin-Lafon, M. Harrison, I. Newman, P. Dewan, J. Larson, C. Unger, H. Stiegler. A. Chabert and L. Nigay of the IHM group of LGI and V. Normand of Bull-Imag also deserve special thanks for their contribution to our model.

References

[Bass 91] L. Bass, J. Coutaz: Developing Software for the User Interface; Addison Wesley Publ., 1991.

[Brown 89] M.H. Brown, G. Avrahami, and K.P. Brooks: 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.

[Coutaz 90] J. Coutaz : Interface Homme-Ordinateur : Conception et Réalisation; Dunod Publ., 1990.

[Coutaz 91a] J. Coutaz, S. Balbo: Applications: A Dimension Space for UIMS's; Proceedings of the Computer Human Interaction Conference, ACM ed., May 1991, pp. 27-32.

[Coutaz 91b] J. Coutaz, L. Nigay: Software design rules for multi-agent architectures. Amodeus BRA 3066 Deliverable.To appear, August 1991.

[Hayes 85] P.J. Hayes, P. Szekely, R. Lerner : Design Alternatives for User Interface Management Systems Based on Experience with Cousin; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April, 1985, 169-175.

[Jacob 84]   R.J.K. Jacob : An Executable Specification Technique for Describing Human-Computer Interaction; Advances in Human Computer Interaction, H.R. Hartson, ed. Alex Publishing Co., 1984.

[Norman 86] D. A. Norman, S. W. Draper : User Centered System Design; Lawrence Erlbaum Associates Publ., 1986.

[Olsen 83] D.R. Olsen, E.P Dempsey : Syngraph : A Graphical User Interface Generator; Computer Graphics, July 1983, 43-50.

[Olsen 89] D.R. Olsen : A Programming Language Basis for User Interface Management; CHI'89 Conference proceedings, K. Bice, C. Lewis editors, ACM Press publish., April, 1989, 171-176.

[OSF 89] OSF : OSF/Motif, Programmer's Reference Manual, Revision 1.0; Open Software Foundation, Eleven Cambridge Center, Cambridge, MA 02142, 1989.

[Petoud 89] I. Petoud, Y. Pigneur : An Automatic and Visual Approach for User Interface Design; IFIP WG2.7 Working Conference Proceedings, Engineering for Human-Computer Interaction, Napa Valley, August, 1989.

[Pfaff 85] User Interface Management Systems; G.E. Pfaff ed., Eurographics Seminars, Springer Verlag, 1985.

[Robertson 89] G. Robertson, S. Card, J. Mackinlay: The Cognitive Coprocessor Architecture for Interactive User Interfaces; 2nd symposium on UIST, November, 1989, pp. 10-18.

[Schulert 85] A.J. Schulert, G.T. Rogers, J.A. Hamilton : ADM - A Dialog Manager; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April 1985, 177-183.

[Sturman 89] D.J. Sturman, D. Zeltzer, S. Pieper: Hands-on Interaction with Virtual Environments; 2nd symposium on UIST, Nov., 1989, pp. 19-24.

[Tanner 83] P. Tanner, W. Buxton: Some Issues in Future User Interface Management Systems (UIMS) Development. IFIP Working Group 5.2 Workshop on User Interface Management, Seeheim, November, 1983.

[Valdez 89] Valdez : XVT, a Virtual Toolkit; Byte 14(3), 1989.

[Wasserman 85] A. Wasserman : Extending State Transition Diagrams for the Specification of Human-Computer Interaction; IEEE Transactions on Software Engineering, 11(8), August, 1985.