A Making-Movies Metaphor for Structuring Software Components in Highly Interactive Application

Michelle Jacomi[1], Stéphane Chatty[1], Philippe Palanque[1,2]

[1]Centre d'Etudes de la Navigation Aérienne
7 avenue Edouard Belin, 31055 TOULOUSE Cedex, FRANCE
E-mail: {jacomi, chatty, palanque}@cena.dgac.fr

[2]LIS, University of Toulouse 1,
1 pl. Anatole France, 31042 Toulouse Cedex, FRANCE

ABSTRACT: Structuring full scale, highly interactive applications still involve complex design choices for programmers. This is because current techniques do not cover the issue of structuring applications at all scales. Programmers thus have to make choices without a good understanding of their consequences. We consider that this is similar to the problem encountered by a user who explores a user-driven application and has little guidance on actions that can be performed. In the same way as metaphors have been used to help users anticipate the consequences of their actions, we propose to use metaphors to help programmers make their choices. This article describes a making-movies metaphor that provides guidance for organising the interface of an application, but also its links with the objects of the functional core. We show how this approach can be merged with current software engineering techniques to specify and build full scale applications. This is exemplified with a graphical editor acting as an interface to optimisation algorithms, and used for splitting air space into air traffic control sectors.

KEYWORDS: Software architecture, user interface design, metaphors, formal specification.

1. INTRODUCTION

Structuring interactive applications is still by many aspects an open issue for software engineers and programmers. General architecture models explain how the large blocks of an application should be organised, with little help on how to implement them. At the other end of the spectrum, object oriented toolkits help programmers to structure and build the interactive parts of their applications, but provide little guidance on how to organise the functional core and weave all the resulting classes together. Even though novel approaches such as design patterns try to bridge the gap between those methods, today's programmers are still on their own for many design choices. This is especially painful for highly interactive applications, because the links between interaction objects and functional core objects are many, tight and complex. This explains why programmers often have trouble to scale up techniques presented in academic papers to full scale applications.

In addition to current research on software engineering techniques that would solve those issues, this paper proposes to use a metaphor as a way of guiding programmers. Metaphors have been successfully used for many years for structuring user interfaces. They have been proposed to prevent users from getting lost when the complexity of these systems increases. Indeed, with the advent of user-driven style of interaction, users are driving the interaction and thus no guidance (or as less as possible) is provided to them. In order to fill in this gap metaphors have appeared as one possible way to help users in understanding the use and the meaning of interactive objects. This help is given for example by "providing paths through the jungle of functionality" (Tscheligi & Vanaanen 1995).

We believe that the same help that is offered by metaphors to end users can be offered to software engineers and programmers, in addition to classical software structuring techniques. The next section of this paper is devoted to the presentation of those structuring techniques. It shows that merging some of them can solve most of the problems encountered while designing large scale applications. Section 3 introduces a metaphor we have defined and used for the building of interactive systems. It explains how this metaphor relates to software architectures and shows how formal specification techniques can be used to define both the inner behaviour of objects and their communication protocols. A large scale case study is presented in section 4. This case study comes from the Air Traffic Control (ATC) domain and is mainly characterised by the large amount of information that has to be displayed at the same time. In order to address this problem the application features a new "shelves-based" user interface with highly interactive interaction techniques such as movable filters in order to cope with information visualisation and handling.

2. STRUCTURING TECHNIQUES FOR INTERACTIVE SOFTWARE

Structuring interactive systems has been recognised for a long time as a challenging problem (Coutaz 96). The main problem to solve is to define and organise the various components and to state clearly their relationships. The solutions proposed in order to address this problem usually come from software engineering techniques customised for interactive systems. This research work can be subdivided in four categories according to the kind of approach they promote: abstraction first, implementation first, reuse first and global first.

2.1 Abstraction First: Models

The kind of research proposes general models more dedicated to the understanding of interactive systems than to their actual building. Among them are the Seeheim model (Pfaff 1985), the Arch/Slinky model (Arch 1992). The main point of these models is to cope with complexity by splitting the interactive application into abstract entities. They are helpful for high level management of applications but they do not go easily towards implementation. For example the Seeheim model must not be followed too closely if implementation is to be object oriented as the organisation in layers will be against object oriented principles that promote highly coherent and weakly coupled objects. For this reason more refined models such as the one presented in (Hudson 1987) refine the Seeheim model in order be closer to the implementation.

Another kind of approach that promotes abstraction first is the agent one that organise the application into collection of cooperating agents. Such models can be used recursively to define the application at different levels of abstraction. The communicative aspect of interactive systems provide a basis for further structuring the system as a network of interactors, each dealing with different subsets of the human computer dialogue. The most used of those models are MVC and PAC (Coutaz 1987), that is now extended to the PAC-Amodeus model (Nigay & Coutaz 93) which is a blending of the Arch/Slinky model and the agent model PAC.

The main drawbacks of these approaches are the lack of methodology in order to support the top-down process they promote and the fact that the number of elements in the model is too small in order to provide efficient classification of components in real size applications.

2.2 Implementation First: Toolkits

Toolkits provide functions and primitives for building interactive systems. Compared to models they address the problem of structuring interactive systems in the opposite way. Toolkits are usually based on conceptual models that impose a predefined way for organising the code of the applications. For example event-based environments such as Sassafras (Hill 1987) or more recently Visual Basic™ organise the code according to event-handlers. Some development environments are associated with other kind of architectural frameworks (InterViews, MacApp). They need a long time for designers to know how to use them and are heavily linked to the underlying toolkit used. Hence, they do not really provide reusable components and composition rules. The other problem is that they do not relate to abstract models and thus it is really hard for designers to have a global understanding of the application.

2.3 Reuse First: Design Patterns

A new tendency proposes design patterns as models to construct applications (Buschmann et al. 1996, Gamma et al. 1995). They provide useful information about both the structuring of a system and its actual implementation. These patterns can be seen as a "glue" between the abstract models and the implementation ones but they address more "low-level" aspects of the life-cycle of an application and are more suitable at implementation step than at conception level. In (Buschmann et al. 1996) two patterns (MVC and PAC) are addressing the problem of interactive systems, but they are only refinements of models used for a long time in the domain of interactive systems.

2.4 Insight First: Metaphors

A metaphor provides a method by which people can quickly learn and understand how to use a system. Through metaphors, users infer the meaning, the behaviour and the manipulation of objects by mapping their aspects in the system onto the associated one in the metaphor. For instance, with the desktop metaphor, users might infer that a trash can-like object is a container for objects that are to be discarded, and may infer that objects can be dropped into the trash without having to learn explicit instructions on how to throw away files: the knowledge for throwing away objects is contained in user's knowledge about trash cans in the real world (Lundell & Anderson 1995).

Metaphors have not only been used for organising the presentation of interactive systems. Several toolkits or development environments have been designed according to metaphors. A metaphor-based programming environment helps designers in organising software components. The need for a metaphor is even more important for interactive applications including advanced interaction tools since they are made up of a huge set of objects dedicated to the interface and interacting together. Several metaphors have already been proposed and used for structuring libraries see for example X_{TV} (Beaudouin-Lafon et al. 1990) and Whizz which is a library for building animated interactive applications based on a musical metaphor (Chatty 1992).

2.5 Discussion

Even though the different models presented before have proven their usefulness for designing interactive systems, they do not provide enough information for structuring the software components needed in a complex interactive application. Splitting such a system into agents means creating hundreds of user interface components. Those components, their roles and their relationships are difficult to identify for designers, and this can be even more difficult when it comes to maintain the system. The main problem with these approaches is that they only provide information at a generic level i.e. for all the interactive applications and no information for a given application that has to be built, and this particular point is addressed in the next section.

3. THE MAKING-MOVIES METAPHOR

We had to design an interactive application integrating new interaction techniques such as movable and specialised filters, according to the principles of the Magic-Lenses™. We used a metaphorical architecture framework for interactive application based on PAC model. Of course, as stated by (Lakoff & Johnson 1980) "metaphors do not imply a complete mapping of every concrete detail of one object or situation onto another". This section is thus devoted to present the metaphor and several aspects that can be found in making-movies will be emphasised and other will be suppressed.

3.1 Context

As a starting point we were using the X_{TV} -Whizz toolkit for the implementation of the application. The TV metaphor proposed in X_{TV} allows designers to structure most of the components of an application. However, this metaphor does not provide a basis for describing their behaviour and the relations between these components and those of the functional core. We thus extended the metaphor to address those problems. However, the resulting making-movies metaphor is not only dedicated to model components of the library but to describe precisely all the components that are to be part of the application.

3.2 The metaphor

All objects are articulated around the user and coordinated by a global object identified as the *Director*. Its role is to ensure the global coherence of the application according to its specification and its current state.

The director is helped in its tasks by assistants, that play as sub-directors for specific parts of the application. They are classified in two categories:

- Assistants linked to the Functional Core of the application, dealing with the communication and coherence between the Interface part and the Functional Core. The file manager in the desktop of the Macintosh™ could have been designed as an assistant linked to the functional core (the actual management of files).
- Assistants linked to the Interface, and dedicated to specialised interactions with users. They encapsulate a whole interactive functionality of the application. The help management in the desktop of the Macintosh™ could be designed as a assistant linked to the interface.

This distinction between assistants comes from the abstract models presented in section 2.1 and is really important as it preserves independence between the functional part and the presentation part of the application.

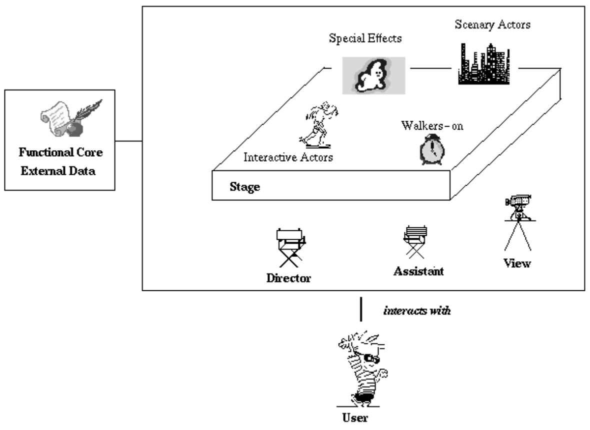

Figure 1: The Making-Movies metaphor

The different objects of the making-movies metaphor can be seen in Figure 1. The objects of the interface, named *Actors*, have predefined characters and behaviours, and may also react to user's actions or events emitted by other actors. Actors are organised on virtual surfaces named *Stages*, and can be viewed by users through objects named *Views* (i.e. windows).

Several stages may be displayed in a view. Their contents is organised consistently with the functional aspects of the application. In order to provide designers with more precise information the metaphor distinguishes four kinds of actors:

1. *Interactive Actors* are entities that can be manipulated by the user and usually correspond to application domain concepts displayed on the interface.
2. *Walkers-on* are actors whose appearance and/or behaviour evolve according to the current state of the application, but cannot be manipulated by users. Walkers-on are only dependent from the application, and not directly linked to users' interactions.
3. *Special effects* are temporary actors dynamically created used for providing feedback during interactions. The feedback is either visible or audible. For example, in the desktop metaphor, a temporary actor is created when the user drag a file on the screen.
4. *Scenery*, are non-interactive graphical objects defining the static appearance of the interface. Their appearance never changes during a session. They constitute the "background" of the display such as the pattern on the background of the desktop of the Macintosh™.

For each actor a *scenario* describes its behaviour according to its internal state and the current state of the application handled by the director.

However, neither the internal structure of the objects nor the communication mechanism among them are described by the Making-Movies Metaphor.

3.3 Decomposition of Actors

In order to describe the internal structure of the objects, we applied and refined the principles of the PAC model (Coutaz 1987). Figure 2 presents the graphical representation of this refinement.

• The Control describes the communication between objects. Two mechanisms are available. The first one, called distribution is used when the sender does not know the receiver (kind of broadcast). The other one called client-server, when the sender holds the reference of the receiver and directly invokes one of its services (public functions). Those communication mechanisms have been formally defined (Palanque & Bastide 1994).

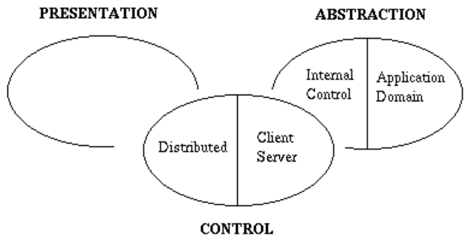

Figure 2: Refinement of the PAC model

• The Abstraction is made up of two parts called Application and Internal. The first one contains attributes and functions linked to the application domain. The second one handles the internal state of the object, and contains attributes and functions ensuring the inner consistency of the object.

• The Presentation holds the graphical attributes of the actors. Those attributes can only be manipulated by functions according to the communication mechanisms described above.

4. A FULL SCALE APPLICATION

Osmose is a tool developed at CENA for dividing airspace into control sectors. It combines optimisation techniques with a direct manipulation editor. The users of Osmose are flow management experts. They load traffic data extracted from flight plans, select a geographic area and ask Osmose to propose a set of sectors with balanced workloads. Then they use the interaction capabilities of Osmose to explore the results and adjust it if necessary.

The Figure 3 shows the interface of the Osmose application. The metaphor used for the presentation of Osmose is based on shelves, closets and boards with 3D effects.

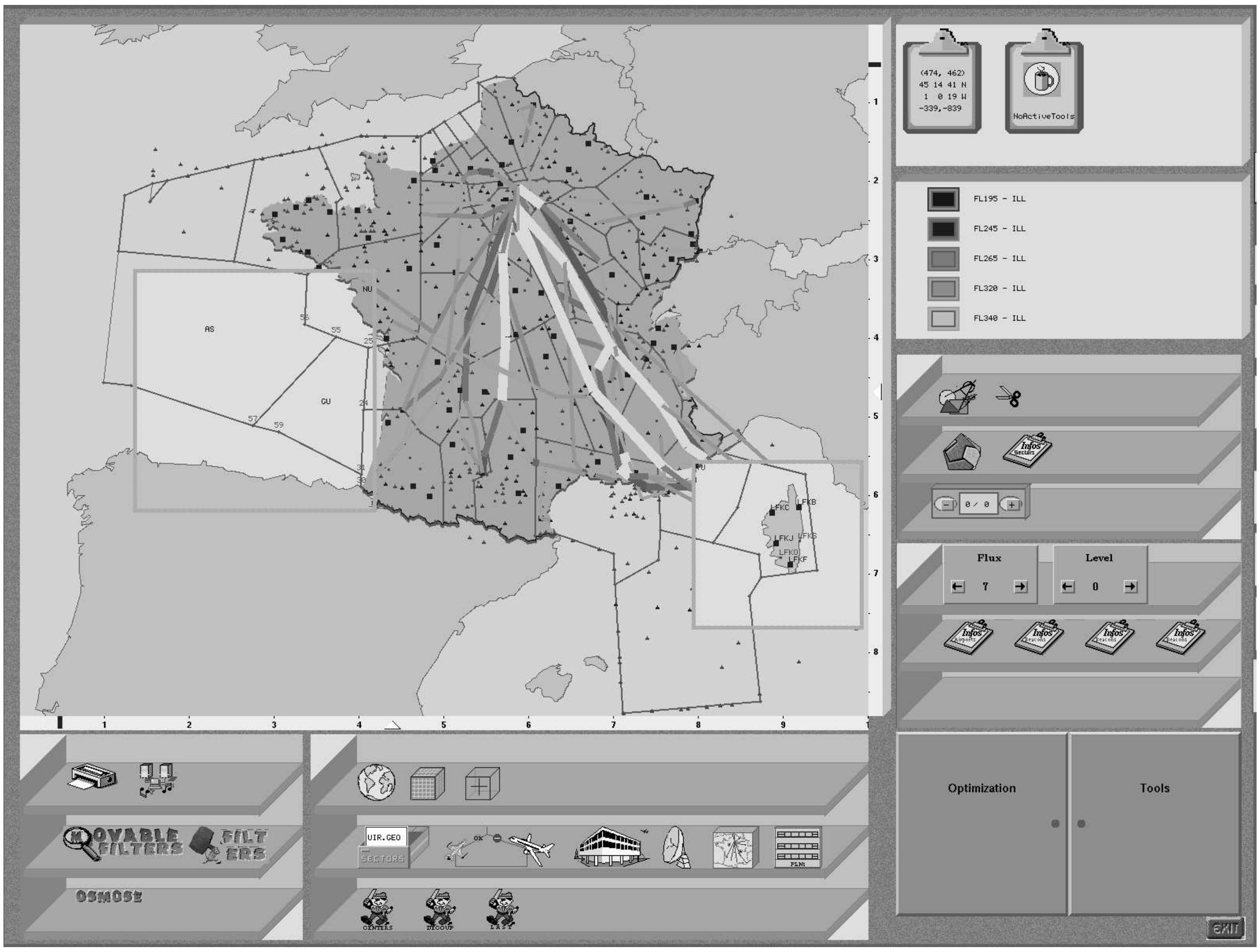

Figure 3 : The user interface of the Osmose application

The Osmose interface consists in three areas: the editing board, the post-it boards (status data), and the set of closets and shelves.

• The editing board is aimed at presenting ATC information to the users and allowing them to interact on the objects presented. Data such sectors, airports, or air-traffic flows can be modified by users by direct manipulation. In Figure 1, the black dots on the maps represent the airports the polygons the sectors and the thick lines the flows between airports.

• The post-it boards (upper right corner of Figure 1) contains several indicators on the application status and displayed information: current activated function, scale used, mouse position in latitude/longitude coordinates, etc.

• The graphical objects that trigger Osmose functions are displayed as objects standing on shelves, enclosed in closets. A closet is devoted to objects from the same "group": one for General Services such as print or save functions (the bottom left one), one for

Sectors Tools (the middle right one), one for ATC tools (the bottom middle one), etc.

Movable filters for instance, provide the ability to focus on a selected area by visualising detailed information while simultaneously keeping the global context displayed. In Figure 3, two movable filters are instantiated:

• one located above Corsica displays only the map, airports and airports names,

• one on the Atlantic only displays the map, the sectors borders, the vertices identifiers, and the sectors names.

This user interface presents several characteristics that fit the requirements of the Osmose application:

• Modularity is easily supported as adding or removing a functionality to the application will correspond to removing one element on a shelve. Group of functions can be handled as a whole by interacting directly with closets.

• Customisability is fully supported as users can dynamically change the appearance of the interface by moving the "graphical-objects" from one shelve to another one shelve according to their convenience (frequency of use, etc.). Using movable filters, this customisation is reinforced as not only the visualisation of functions but also data can be dynamically specified by the users.

5. IMPLEMENTATION

The making-movies metaphor presented above has been fully used for the structuring of the object classes used in Osmose. Thus any software component that has been built comes from the interpretation of the metaphor.

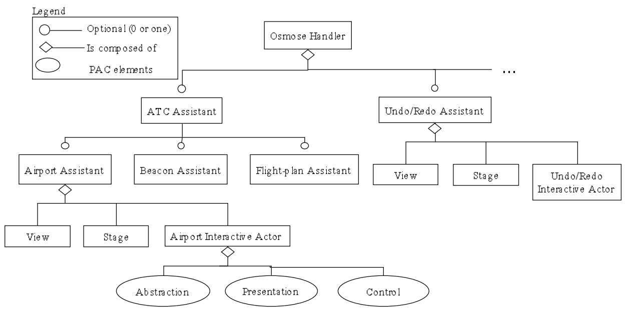

Figure 4: A subset of the class hierarchy in Osmose

The Figure 4 presents the hierarchy of classes of our application. The director *OsmoseHandler* is associated to Assistants such as *ATCAssistant* or *HistoryAssistant* inherating. An Assistant is made-up of: *views* that manage the display, *stages* that organise the displayed information, and *actors* that represent the interactive entities. The fact that each interactive actor is built according to the PAC model is represented by the composition relationship at the bottom of Figure 4. The right hand part of the Figure has been cut for space reason, but the same composition is applied to all the interactive actors.

6. CONCLUSION

In this paper, we have described the use of a metaphor to help programmers structure their highly interactive applications. We also have exemplified the use of that metaphor through the construction of a full scale application developed at CENA for air traffic experts.

Using a metaphor has shown to provide at least three kinds of help to programmers:

• as the metaphor is by definition object-oriented all the advantages of this approach such as reuse, reliability and encapsulation are fully supported by this approach.

• help in identifying presentation objects, functional core objects, and their relations. This is useful both at design time and at maintenance time, because it helps programmers navigate through the organisation of the application.

• help in distributing programming tasks in time or among programmers. Using the metaphor leads to easier integration.

However, though they have shown their usefulness in our developments, we are conscious that metaphors are only a help that cannot replace classical methods for structuring software. Our future research in that direction will focus at integrating such a metaphor with tools and methods for structuring interactive software, along several different lines. One of our approaches will consist in specifying more precisely the behaviour of objects (actors in our metaphor) with formal methods in order to perform static analysis of that behaviour (Palanque & Bastide 1995). Another approach will take the toolkit approach, and extend a graphical toolkit and its corresponding interactive editor (Esteban et al. 1995) to the description of the behaviour of objects and their relations.

7. REFERENCES

(Arch 1992) A metamodel for the runtime architecture of an interactive system. The UIMS Tools Developers Workshop, SIGCHI Bulletin vol. 24, n°1, pp 32-37.

(Beaudouin-lafon et al. 1990) Beaudouin-Lafon M. Berteaud Y. Chatty S. Creating direct manipulation interfaces with X_{TV}. EX'90, European conference on X Window.

(Buschmann et al. 1996) A system of patterns. Wiley Publ.

(Chatty 1992) Chatty S. Defining the behaviour of animated interfaces. Engineering for Human Computer Interaction Conference pp. 95-109, North Holland.

(Coutaz 1987) Coutaz J. PAC an implementation model for dialogue design. Proceedings of the Interact'87 conference. North Holland. pp. 431-437.

(Coutaz et al. 1996) Coutaz J., Nigay L. Salber D. Agent Based architecture for modelling Interactive Systems. Critical Issues in User Interface System Engineering, pp. 191-209, Benyon & Palanque (Eds.), Springer Verlag..

(Esteban et al. 1995) Esteban O., Chatty S. Palanque P. Whizz'Ed: a visual environment for building highly interactive software. Proceeding of Interact'95 conference Chapman et Hall. pp. 121-126.

(Fekete 1996) Fekete J.D. Un modèle multicouche pour la construction d'applications graphiques interactives, PhD thesis, Université Paris Sud.

(Gamma et al. 1995) Gamma E., Helm R., Johnson R. Vlissides J. Design patterns. Addison Wesley.

(Hill 1987) Hill R. Supporting concurrency, communication and synchronisation in Human-Computer Interaction. The Sassafras UIMS. Proceedings of ACM CHI'87 pp. 241-248.

(Hudson 1987) Hudson S.E. UIMS support for direct manipulation interfaces. Computer Graphics vol.21 n°2. pp120-124.

(Hussey 1996) Hussey A. & Carrington D. Comparing two user-interface architectures: MVC and PAC. FAHCI'96, Springer Verlag.

(Lakoff & Johnson 1980) Lakoff G. & Johnson M. Metaphors we live by. The University of Chicago Press.

(Lundell & Anderson 1995) Lundell J. & Anderson S. Designing a "Front Panel" for Unix: The Evolution of a Metaphor CHI95 ACM p.573-579

(Nigay & Coutaz 1993) Nigay L. & Coutaz J. A design space for multimodal systems: Concurrent processing and Data fusion. Proceedings of INTERCHI'93, ACM Press, pp. 172-178.

(Palanque & Bastide 1994), Palanque P & Bastide R. Petri net based design of User-Driven Interfaces using the Interactive Cooperative Objects Formalism. Proceedings of Design, Specification and Verification of Interactive Systems, Springer Verlag, pp. 383-400.

(Palanque & Bastide 1995), Palanque P & Bastide R. Verification of an interactive software by analysis of its formal specification. Proceeding of Interact'95 conference Chapman et Hall. pp. 191-196.

(Pfaff 1985) Pfaff G.E. et al. User Interface Management Systems, G.E. Pfaff Ed., Eurographics Seminars, Springer Verlag.

(Tscheligi & Vanaanen 1995) Tscheligi M. & Väänänen-Vainio-Mattila. Metaphors in User Interface Development: Methods and Requirements for Effective Support. Critical Issues in User Interface System Engineering, pp. 249-263, Benyon & Palanque (Eds.), Springer Verlag.