Engineering Multimedia User Interfaces with Objects and Patterns

Fernando D. Lyardet
LIFIA. Departamento de Informática, UNLP
fer@sol.info.unlp.edu.ar

Gustavo H. Rossi
LIFIA. Departamento de Informática, UNLP. Also at UNLM and Conicet
gustavo@sol.info.unlp.edu.ar

Daniel Schwabe
Departamento de Informática, PUC-Rio, Brazil
schwabe@inf.puc-rio.br

Abstract

In this paper we discuss the use of an object-oriented design model to describe multimedia user interfaces. Our approach is complemented with a set of design patterns aimed at capturing recurrent interface design problems together with good solutions to those problems. We first introduce the problem of designing usable multimedia interfaces; we then present the Object-Oriented Hypermedia Design Method to contextualize our work; then we discuss in depth the ADV object model and its feasibility for describing static and dynamic aspects of multimedia user interfaces; we finally introduce design patterns and present some simple patterns together with real examples taken from well-known multimedia applications, including web applications. Some further work is also discussed.

1. Introduction

Building Multimedia and Hypermedia applications is not easy. The combination of navigation through an information space with the inherent difficulties of dealing with multimedia data (in which complex transformations may occur) has been usually reported as the origin of many implementation and usability problems.

Electronic worlds provide both special challenges and opportunities in navigation. As electronic worlds become vast, distributed, and more integrated with daily activities, improved support for navigation is increasingly needed. Fortunately, good interface and information design provides such support and offer new ways of navigating the cyberspace [5].

Within a growing trend to the use of formal design methods in multimedia applications development, we can observe that:

- Human-computer interfaces of multimedia applications are often built in their entirety using implementation- and environment-dependent tools.
- Current hypermedia and multimedia design methods tend to emphasize conceptual and navigational design, but the design problems associated with the interface are usually neglected.

We claim that a formal design model should be used prior to implementation in order to maximize dialogue-independence and reuse-in-the-large of interface components. In addition, a design model can support improved communication between designers and implementors; when interface design decisions are well documented, they can be used both as a test to validate the implementation and as a reference during maintenance.

In this position paper we briefly describe our design approach for building multimedia user interfaces. It combines the use of some well-known principles of object-orientation together with design patterns as a way to document design decisions that are hard to specify using the method's primitives. As our approach has been developed in the context of the Object-Oriented Hypermedia Design Method (OOHDM) [29,30], we first present the most outstanding aspects of the method. We next present our approach to user interface design and present some simple patterns to support our claim. Some further work is finally discussed.

2. Overview of OOHDM

The Object-Oriented Hypermedia Design Method is a model-based approach for building hypermedia applications. It comprises four different activities namely conceptual design, navigational design, abstract interface design and implementation. They are performed in a mix of incremental, iterative and prototype-based development style. During each activity a set of object-oriented models describing particular design concerns are built or enriched from previous iterations. In Table 1 we summarize OOHDM activities, modeling primitives, abstraction and presentation mechanisms. We next describe each activity. (See [19] for a more detailed description).

0-8186-8925-0/98 $10.00

2.1. Conceptual Design

During Conceptual Design a model of the application domain is built using well known object-oriented modeling principles [26], augmented with some primitives such as attribute perspectives and sub-systems. Conceptual classes may be built using aggregation and generalization/specialization hierarchies. The main concern during this stage is to capture the domain semantics as "neutrally" as possible, with very little concern for the types of users and tasks. The product of this stage is a class and instance schema built out of Sub-Systems, Classes and Relationships.

Table 1: Summary of the OOHDM Methodology

Activities	Products	Design Concerns
Domain Analisis	Classes, sub-systems, relationships, attribute perspectives	Model the semantics of the application domain
Navigational Design	Nodes, links, access structures, navigational contexts, navigational transformations	Takes into account User profile and task. Emphasis on cognitive aspects. Build the navigational structure of the application
Abstract Interface Design	Abstract interface objects, responses to external events, interface transformations	Model perceptible objects, implementing chosen metaphors. Describe interface for navigational objects. Define lay-out of interface objects
Implementation	Running application	Performance, completeness

2.2. Navigational Design

In OOHDM, an application is conceived as a navigational view over the conceptual domain. This reflects the point of view that one of the key distinguishing features of hypermedia applications is the notion of navigation. It is in this stage that the designer takes into account the types of intended users, and the set of tasks they are to perform using the application.

2.3. Abstract Interface Design

Once the navigational structure has been defined, it must be made perceptible to the user through the application's interface; in this stage we describe the abstract interface model. This means defining which interface objects the user will perceive, and in particular the way in which different navigational objects will look like, which interface objects will activate navigation, the way in which multimedia interface objects will be synchronized and which interface transformations will take place

A clean separation between both concerns, navigational and abstract interface design, allows the building of different interfaces for the same navigational model, leading to a higher degree of independence from user-interface technology, and also allowing conformance with varying user needs or preferences.

3. Issues in Building the Abstract Interface Model.

In order to specify the abstract interface model we need to define interface metaphors and objects, and describe their static and dynamic properties and their relationships with the navigational model in an implementation-independent way [31]. We need to specify:

- The interface appearance of each navigational object seen by the user. The same navigational object may have different interface representations in different situations. For example, a node may have the representation of a picture of a monument when browsing the information and be an icon when used in the context of an access structure, such as an icon on a city map acting as an index to monuments.
- Other interface objects for providing access to other application functions such as menu bars, control buttons, and menus.
- The relationships among interface and navigational objects such as the way an external event such as the user clicking the mouse will affect navigation.
- Interface transformations occurring because of the effect of navigation or external events on the behavior of different interface objects.
- Finally, synchronization of some interface objects must be considered particularly when dynamic media such as audio and video are involved.

In the following sections we will show our design approach for defining the abstract interface model. Although the design method is discussed in the context of the OOHDM approach, it will become evident that Abstract Data Views can be easily adapted to other design approaches such as HDM [11, 12], EORM [15] or RMD [2]. Furthermore it can be used as a design front-end to advanced object-oriented hypermedia environments such as MacWeb [17], or DEVISE [13].

Our object-oriented approach for describing user interfaces can be also used with more general multimedia applications as interactive kiosks, web information systems, etc.

3.1. Overview of the Abstract Data View (ADV) Design Model

The ADV design model was originally devised to specify clearly and formally the separation of the user interface from the application components of a software system and to provide an environment-independent design method leading to higher degrees of reuse for both interface and application components [4, 7, 8]. ADVs are objects in that they have a state and an interface, where the interface can be exercised through regular functional or procedural calls or input and output events. ADVs are abstract in that they only represent the interface and the state, and not the implementation. ADVs are generally used to represent interfaces between two different media such as a user, a network or a device such as a timer, or as an interface between two or more Abstract Data Objects (ADOs) where ADOs are ADVs that do not support events.

In a typical application using ADVs, we have a set of ADOs managing data structures and control within the application and a set of interface objects (instances of ADVs) managing interface aspects of the application such as user input and system output to the user.

Thus, ADVs can be viewed as media transformers or transformers between ADOs. Thus ADVs are active objects while ADOs are passive and must be activated through one or more ADVs. Because of the split between active and passive objects, ADVs and ADOs can be used in client-server pairs to represent a software system.

Typically an ADO "owns" one or more ADVs which represent some aspect of its state to the "external" world. Each ADV must be consistent with its owning ADO and all ADVs owned by an ADO must be consistent with each other. These two types of consistency are called vertical and horizontal consistency respectively, and are illustrated in Figure 3.

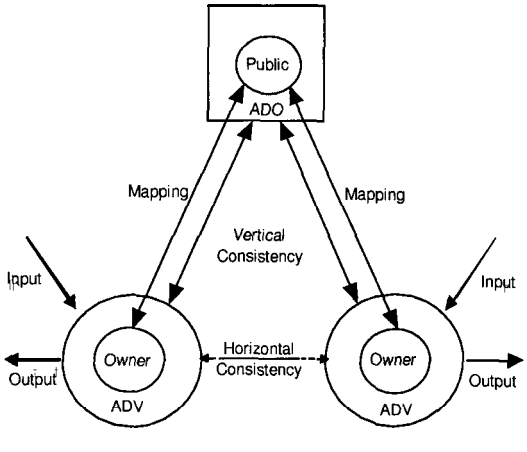

Figure 1: ADVs as user interfaces

A standard textual schema exists that can be completed to define an ADV or ADO. The expressions in the schema are usually written in some form of temporal logic or other formal design language (such as VDM or one of its object-oriented extensions).

An ADV when used in the design of hypermedia applications can be viewed as an interface object comprising a set of attributes which define its perception properties, and the set of events it can handle such as user-generated events. Attribute values may be defined as constants thus defining a particular style of appearance such as position, color, or sound. A reserved variable, "perceptionContext", is used to indicate modifications to the perception space. When we want to make some object perceivable we add it to perceptionContext, and elements removed from perceptionContext are no longer perceivable.

In the context of OOHDM, navigational objects such as nodes, links or access structures will act as ADOs, and their associated ADV will be used for specifying their appearance to the user.

Different abstraction and composition mechanisms are used in the ADV design approach; first ADVs may be composed by aggregation or composition of lower-level ADVs, thus allowing the construction of user-interfaces with nested perceivable objects.

3.2. Configuration Diagrams

Configuration diagrams have been originally defined in the context of ObjectCharts [6]. They are useful for expressing patterns of communication between objects in terms of provided and required services. In the ADV design approach they are used to represent external (user-initiated) events that an ADV handles; the services provided by the ADV (such as display) and the communication among ADVs and ADOs. A configuration diagram may also show the nesting structure of composite ADVs. In our context, we are interested in defining the way in which the user will interact with the hypermedia application and in particular which interface objects will cause navigation. Each Node Class will define a public interface with the services provided by its objects. In particular all nodes will react to the message: anchorSelected (A) by asking anchor A to initiate navigation across its associated link. In a composite ADV with different interface objects such as buttons associated with node anchors, we annotate the name of the anchor in the configuration diagram.

Using Configuration Diagrams it is possible to specify the abstract interface of a hypermedia application, that is, interface objects, their structural relationships and the relationships among interface objects (ADVs) and navigational objects (ADO). We next explain ADVcharts, a visual formalism for specifying the dynamic aspects of a hypermedia application.

3.3. ADV charts

ADVcharts are a generalization of Statecharts [14] and ObjectCharts [5] supporting nesting of states and ADVs and allowing the expression of the association among external events (typical of hypermedia applications) with ADVs.

As ADVs may be composed from lower-level ADVs, expressing the behavior of an ADV using an ADVchart will usually involve describing nested states and ADVs. Nesting states are the expression of behavioral nesting while nesting of ADVs expresses structural nesting.

ADVcharts are equivalent to StateCharts although the notation of ADVcharts allows the compact expression of the behavior of composite objects through both behavioral and structural nesting.

3.4. Hypermedia interface specification

To make this presentation concrete we will show some examples using a real hypermedia application: Microsoft's Art Gallery. In Figure 9 we show an outline of the navigational scheme of Art Gallery as described using OODHM. As previously explained, defining the abstract interface model of a hypermedia application implies:

- Defining ADVs for each navigational object. In fact, defining at least one ADV type for each navigational class.
- Specifying the configuration diagram showing static relationships among ADVs and ADOs (which are navigational objects);
- Specifying the ADVchart for each ADV showing the dynamics of the hypermedia application.

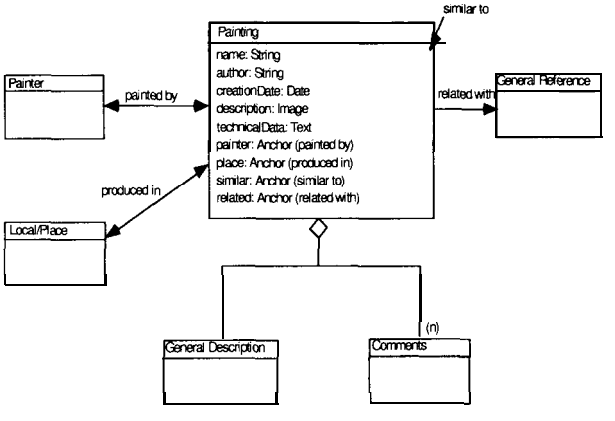

Figure 2: Navigational Scheme of Art Gallery (focusing Paintings)

3.4.1. Defining ADVs and Configuration Diagrams.

We need to define the way in which each navigational object will be perceived by the final user, as well as to specify other interface objects that we intend to make available. This process is usually performed in hypermedia and other interactive applications by defining the interface metaphor, which determines the overall interface structure of the hypermedia application. This metaphor determines whether there will be some regular interface structures omnipresent throughout the application.

For each navigational class, we need to define its corresponding ADV. In the case of Nodes the ADV structure should be easily determined by the node's structure, i.e. there is an enclosing ADV that stands for the whole node, and nested ADVs for each node attribute. Usually there will be some kind of "active" ADVs for anchors such as buttons, In Figure 3 we show an outline of ADV Art Gallery, using the ADVchart notation.

In Figure 3, ADV Art Gallery is a composition of several ADVs including Painting, Painter, Help, Options, and GoBack. The XOR composition of Painting, Painter and Place indicates that they represent exclusive states and they will not usually be perceived at the same time by the user. Meanwhile ADVs Help, Options and GoBack (AND composed in the ADV Art Gallery) will be always perceivable. Short Reference and Reference are different ADVs for Node Class General Reference. A Short Reference may appear as a "pop-up" ADV in the context of another ADV such as Painting and so it is modelled with AND composition.

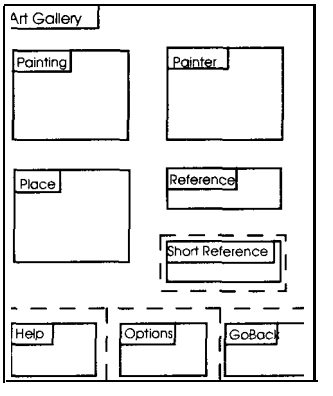

Figure 3: ADVs in Art Gallery

In many hypermedia applications, standard ADVs like Text, Graphic, Image and Button (with the semantics of their counterparts in current hypermedia environments) will be used. Sometimes we will need to define "special purpose" ADVs such as EditableText or AnchoredText as sub-types of existing ADVs. As shown in Figure 4 an AnchoredText will specify a set of anchors (acting as "hot words") and will refine events such as MouseClicked or MouseOn to support the expected behavior.

Although modem hypermedia environments provide similar functionality to those just described, the ADV design formalism allows us to describe the interface

structure and behavior in an abstract way, thus avoiding ambiguities and helping to record critical design decisions.

Usually we will not specify interface aspects of one-to-one links because their presence will be indicated by ADVs corresponding to their anchors; however we can define perceived aspects of one-to-many links by actions such as showing the end-points in a Menu-like style each time the link is activated. In this case we will specify ADV LinkInterface and corresponding sub-types when necessary and will express the desired functionality using ADVcharts.

The same is true for access structures like indexes and guided tours. An access structure may be perceived as a menu, a set of icons, or another more complex representation such as an orientation map. However in all cases we will specify it as a collection of ADVs whose owners will be the target nodes. In this case we could use different ADVs for the elements of an index. For example, we could show a text string for each target node, or an iconic tabletop, with an icon for each node.

In hypermedia applications, navigational operations are usually triggered by the user selecting an interface object that stands for a links anchor. We use Configuration Diagrams as the formalism to express behavioral relationships between interface objects and navigational objects. In Figure 4 we show the Configuration Diagram for ADV Painting.

For the sake of clarity we have omitted some required and provided services such as those for getting attributes' values such as name, creationDate, or technicalData, from Node Painting. In Figure 5 we present ADVs Art Gallery and Painting with part of the actual interface used in Microsoft's Art Gallery.

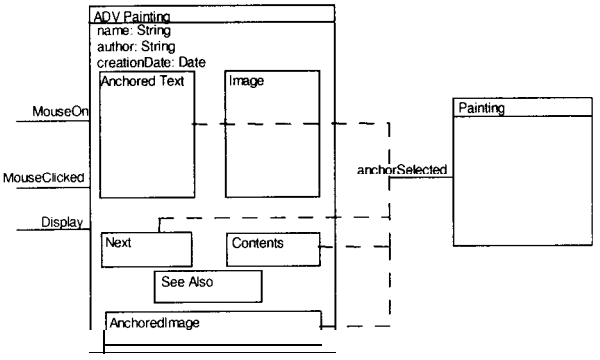

Figure 4: Configuration Diagram for Painting

3.4.2. Expressing Interface and Navigational Transformations. The overall application behavior is completely specified by defining the way in which external events affect both navigation and the interface appearance of the application.

The navigational semantics specify the "internal state" of the application, and how it changes during the navigation process. We must now show the effect of each external event in terms of the transformations occurring in the interface. We will use ADVcharts to show possible states and corresponding transitions of each ADV, in order to understand the way in which individual interface components, in this case ADVs associated with navigational objects behave when reacting to external events. We will suppose all ADVs in our example react to the "Display" event by displaying themselves - in the case of dynamic media "Display" may be understood as "Play".

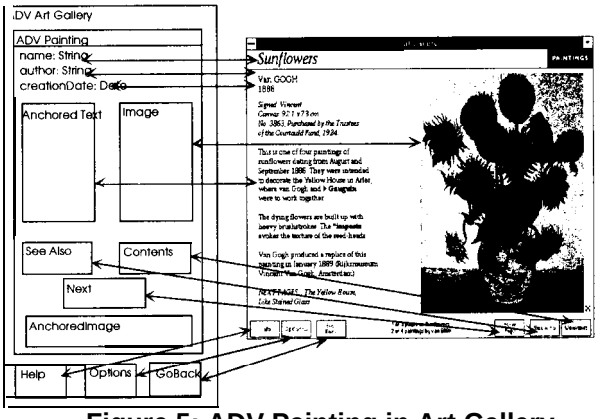

Figure 5: ADV Painting in Art Gallery

In this paper we will only focus on behavior that activates navigation. However, it should be indicated that we can also specify more complex behavior, such as the user selecting a menu option or clicking a button may cause the opening of a special-purpose editor, or create a new link.

4. Conveying design knowledge using Design Patterns

Design Methodologies provide the software designer with the means to express an idea with an appropriate language (usually a set of graphic primitives enriched with textual specifications). Nevertheless, extracting reusable knowledge information from specific, product-oriented designs is usually hard. Furthermore, even when such knowledge can be devised, there are no methodology constructs to help the designer in expressing rationales, trade-offs, etc. In the same way, user interfaces designers tend to reuse successful solutions they found in the past though these usually remain in their minds. This is where design patterns are useful.

Though originated in architecture [1] design patterns are being increasingly used in software design [9]. Design patterns are a good means for recording design experience as they systematically name, explain and evaluate important and recurrent designs in software systems. They describe problems that occur repeatedly, and describe the core of the solution to that problem, in such a way that we

can use this solution many times in different contexts and applications. Looking at known uses of a particular pattern, we can see how successful designers solve recurrent problems.

In some cases, it is possible to give structure to simple patterns to develop a pattern language: a partially ordered set of related patterns that work together in the context of certain application domain. Design Patterns complement methodologies in that they address problems at a higher level of abstraction. Many design decisions that cannot be recorded through the uses of the primitives of a method can be described using patterns. In our work with OOHDM we have found many different kinds of patterns: those describing general architectural constructs in complex hypermedia applications, those showing solutions to recurrent navigational design problems [18] and those that describe usual problems and their solutions in the interface domain **[IO].**

We next present a set of simple design patterns that addresses usual aspects concerning the organization of the interface. They constitute the basis of a pattern system for designing user interfaces. These patterns should not be seen as ultimate solutions to interface problems but rather well known and documented solutions; they have worked in successful applications and can be adapted to others. We describe each pattern using a simple template including the problem that originates the pattern, a short motivation and the solution. In all cases we include an example of the use of each pattern in real applications.

4.1. Information-Interaction Decoupling

Problem: How do you differentiate contents and various types of controls in the interface?

Motivation: A page of a complex application display different contents, and is related to many other pages, thus providing many anchors. Moreover, if the page supplies means of control activation other than navigation (such as triggering some query), the user may experience cognitive overhead. It is well known that when too many anchors are provided in a text, the reader is distracted and cannot take profit of all of them.

Solution: Separate the input communication channels from the output channels, by grouping both sets separately. Allow the "input interaction group" to remain fixed while "the output group" reacts dynamically to the control activation. Within the output group, it is also convenient to differentiate the "substantive information" (i.e. content) from the "status information". This solution not only improves the perception of a node's interface, but also the efficiency of the implementation.

Example: In figure 6, all links to related information on the current topic are displayed on the left. The graphics and video relevant to the current topic are displayed in the middle. Notice there are no links in the text itself.

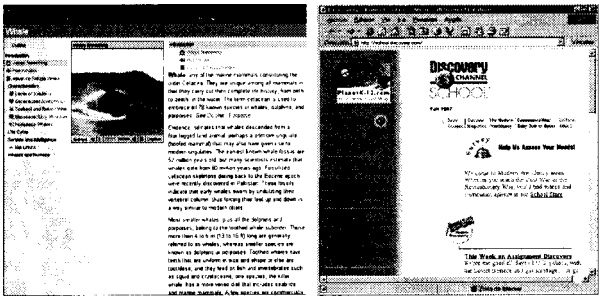

Figure 6: "Information-Interaction Decoupling" from the MS-Encarta97 CD-ROM and the Discovery Channel's educational web page.

4.2. Behavioral Grouping

Problem: How to recognize the different types of controls in the interface so that the user can easily understand?

Motivation: A problem we usually face when building the interface of multimedia applications is how to organize control Objects (such as anchors, buttons, etc.) to produce a meaningful interface. In a typical application there are different kinds of active interface objects: those that provide "general" navigation, such as "back' button, or anchors for returning to indexes, objects that provide navigation inside a context; objects that control the interface, etc. Even when applying Information-Interaction decoupling, there may be a lot of different kinds of control objects.

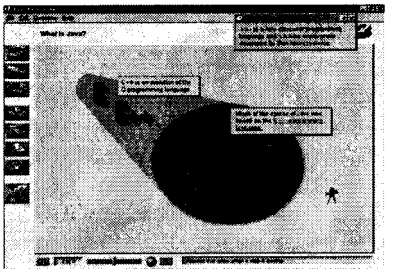

Figure 7: "Behavioral Grouping" from the MindQ's CD-ROM.

Solution: Group control interface objects according to their functionality in global, contextual, structural and application objects, and make each group enhance comprehension.

Example: In figure 7, the first picture is taken from MindQ's CD-Rom "An introduction to programming Java Applets" groups the navigation controls on the left and the current topic playing controls at the bottom.

4.3. Information on Demand

Problem: How to organize the interface in such a way that we can make perceivable all the information in a node, taking into account both aesthetic and cognitive aspects?

Motivation: We usually find ourselves struggling to decide how to show the attributes and anchors ina node.

Unfortunately, the screen is usually smaller than what we need and many times we cannot make use of other media (such as simultaneously playing an audio tape and showing an image) either for technological or cognitive reasons. For example, many times it does not make sense to play an audio recording while the user is watching a video or animation.

Solution: Present only a sub set of the most important ones, and let the user control which further information is presented on the screen, by providing him with active interface objects (e.g. buttons). The activation of those buttons does not produce navigation; they just cause different information of the same node to be shown. This just follows the "What you see is what you need" principle.

Known Uses: Information on Demand is used for example, in the Microsoft Atlas Encarta97 combined with "Node as a Single Unit". Instead of splitting the information unit in several pieces, the user selects which information about the current topic he is interested in.

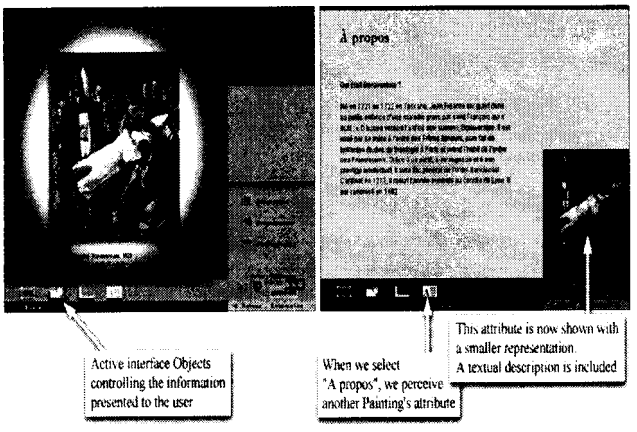

Figure 8: An example of "Information on Demand"

Example: In figure **8,** the first picture taken from the *"Le Louvre"* CD-ROM shows the painting as the main interface object. On the second, we have Information on demand: a textual attribute is perceived.

4.4. Behavior Anticipation

Problem: How do you tell the user the effect or consequence of activating an interface object?

Motivation: Many times, when building an interface, it is necessary to combine different interface elements such as buttons, hotwords, media controls or even custom-designed controls. It is usual to find readers wondering what has happened after activating a control, and the exact consequence of the action performed.

Solution: Provide feedback about the effect of activating each interface element. Choose the kind of feedback to be non-ambiguous and complete: different cursor shapes, highlighting, small text-based explanations called "tool tips". Also, these elements can be combined with sound and animations.

If we are using the behavioral Grouping interface pattern we can select different kinds of feed-back according to the kind of behavior provided; when the interface controls refer to a particular media such as animation, we could use a small status field for that family.

Example: In figure 9, there is an example from the Microsoft Atlas Encarta97. Each time the user positions the cursor over an interface element, a tool tip pops up with an explanation about the effect of activating the control.

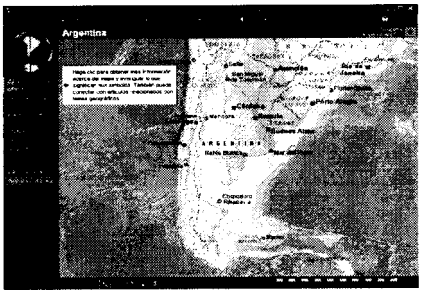

Figure 9: Example of *"Behavior Anticipation".*

5. Concluding Remarks and Further work

We have briefly presented a design approach for engineering multimedia user interfaces; it is based on the use of well-known object oriented principles, via the Abstract Data Views design model plus the systematic application of ideas coming from the design patterns community [9]. We believe that carefully documenting design aspects of multimedia interfaces is a step towards achieving a higher level of reuse besides ensuring correct and predictable behavior. As we have discussed in this paper, many critical design decisions remain hidden in the designers' mind either because design primitives do not allow him to express what he needs to (see for example "Behavioral Grouping) or because the solution involves many complex relationships among interface objects (see "Information on Demand"). Design Patterns are a powerful conceptual tool for enriching the designer's vocabulary and for allowing those designers to share their

expertise. We have presented some simple patterns addressing recurrent problems in multimedia interface design; our aim is not to present them as ultimate solutions but to illustrate the power of patterns as the vehicle to record and reuse design knowledge. Discovering design patterns in existing successful applications is a rewarding task and this paper is a way to stimulate the multimedia design community to share its collective knowledge in the form of design patterns; we aimed at building catalogues or pattern systems (as the one in [9]) describing recurrent problems and their solutions in an abstract way so they can be reused by others later.

We are now working to discover some more patterns and to incorporate them into our design vocabulary to obtain more abstract design descriptions. In this way, a designer may, for example, use the ADV approach and use the components of the Interface on Demand pattern without needing to express all the dynamic relationships existing among active interface objects (buttons for example) and the transformations they cause. Incorporating patterns into the development cycle is not easy and requires the designer to be aware of the existence of those patterns thus leveraging its design experience.

6. References

[1] Alexander, S.Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel: A Pattern Language". Oxford University Press, New York 1977.

[2] P. 2, T. Isakowitz and E. Stohr: "Designing Hypermedia Applications", Proceedings of the 27th. Hawaii International Conference on System Sciences, Hawaii, Jan. 1994.

[3] L.M.F. Cameiro, M.H. Coffin, D. D. Cowan and C.J.P. Lucena: "ADVcharts: a Visual Formalism for Highly Interactive Systems". In M.D. Harrison and C. Johnson editors, Software Engineering in Human-Computer Interaction. Cambridge University Press, 1994.

[4] Susanne Jul and George W, Fumas: "Navigation in electronic Worlds: a CHI 97 Workshop", ACM CHI Conference Workshop, 1997

[5] D. Coleman, F. Hayes and S. Bear: "Introducing Objectcharts or how to use Statecharts in Object-Oriented Design". IEEE Transactions on Software Engineering, 18(1): 9-18, January 1992.

[6] D. D. Cowan R. Ierusalimschy, C.J.P. Lucena and T.M. Stepien: "Abstract Data Views". Structured Programming, 14(1):1-13, January 1993.

[7] D. D. Cowan, ; C. J. P.Lucena, ; "Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse". IEEE Transactions on Software Engineering, Vo1.21, No.3, March 1995.

[8] Gamma, R. Helm, R. Johnson and J. Vlissides: Design Patterns : elements of reusable object-oriented software: , Adisson Wesley, 1995.

[9] A. Garrido, G. Rossi and D. Schwabe. "Pattern Systems for Hypermedia" In Proceedings of PloP'97, Pattern Language of Programming, 1997.

[10] F. Garzotto, P. Paolini and D. Schwabe: "HDM- A Model for the Design of Hypertext Applications", Proceedings ofHypertext' , ACM Press. pp. 313.

[12] K. Gronbaek: "Composites in a Dexter-Based Hypermedia Framework", Proceedings of the ACM European Conference on Hypermedia Technology, Edinburgh, 1994.

[13] J. Hannemann, M. Thuring: "What matters in developing interfaces for hyperdocument presentation?" Workshop in Methodological Issues on the Design of Hypertext-based User Interfaces, Darmstadt, Germany, July 1993.

[14] D. Harel, A. Pnueli, J.P. Schmidt, R. Sherman: "On the formal semantics of statecharts". Proc. 2nd. IEEE Symposium on Logic in Computer Science, Ithaca, N.Y., June 1987.

[15] D. Lange: "An Object-Oriented design method for hypermedia information systems", Proceedings of the 27th. Annual Hawaii International Conference on System Science, January 1994.

[16] Michael D. Levi and Frederik Conrad: "Usability Testing of World Wide Web Sites: A CH197 Workshop". ACM CHI'97 Workshop

[17] J. Nanard and M. Nanard. "Using Structured Types to Incorporate Knowledge in Hypertext, Third ACM Conferences on Hypertext Proceedings, Hypertext'9 1 ed. ACM Press. pp.. 329.

[18] G. Rossi, D. Shwabe and A. Garrido: "Design Reuse in Hypermedia Design Applications Development "Proceedings of ACM International Conference on Hypertext (Hypertext'97), Southampton, April 7- 11, 1997, ACM Press.

[19] D. Schwabe, G. Rossi and S. D. J. Barbosa. "Systematic Hypermedia Application Design with OOHDM".Proceedings o f Hypertext'96 (HT96). Washington, March 1996.