An Object-Oriented Model for Designing the Human-Computer Interface Of Hypermedia Applications

G. Rossi [1]*, D. Schwabe [1], C.J.P. Lucena [1] and D.D. Cowan [2]

[1] Departamento de Informatica Pontificia Universidade Católica. R. M. de S. Vicente, 225 Rio de Janeiro, RJ 22453-900, Brazil.
Fax: +55-21-511 5645. Telephone: +55-21-529 9544.
E-mail: [lucena, schwabe,rossi]@inf.puc-rio.br
[2] Computer Science Department. University of Waterloo, Waterloo, Ontario, Canada.
E-mail: dcowan@csg.uwaterloo.ca
*also at LIFIA, Universidad Nacional de La Plata, Argentina, and Conicet.

ABSTRACT

This paper presents an object-oriented approach to specifying the user interface of a hypermedia application using the Abstract Data Views (ADVs) concept. We discuss ADVs in the context of an object-oriented hypermedia design method (OOHDM) showing how to specify the interface aspects of hypermedia objects including nodes, links and access structures such as indexes and guided tours using high-level abstraction and composition mechanisms such as aggregation and generalization/specialization. We state that (ADVs) make it possible to describe, in an abstract, implementation-independent way, a number of relationships including the media objects perceived by the user of the hypermedia application, the mode of interaction with these objects, and the interface transformations that occur while navigating through the hypermedia. In addition, we present ADVcharts and Configuration Diagrams as a design tool to specify both the interface and the static relationships between interface objects and navigational objects in the hypermedia application. In the discussion of ADVcharts, a notation that combines concepts from Statecharts, Objectcharts and Petri Nets, we show how to specify the dynamic aspects of a hypermedia application, in particular the interface transformations that occur when certain media objects are "activated" during navigation. Finally we discuss some further issues such as the description of reusable interface objects and patterns of interaction styles, and compare our approach with other formal models used during hypermedia interface design.

1. INTRODUCTION

Building 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. In particular, it has been repeatedly stated (See [Schuler94] for example) that constructing a hypermedia interface is a critical aspect of the creation of a large hypermedia application.

Nevertheless, in spite of growing interest in hypermedia design methods [Balasubramaniam94, Garzotto91, Garzotto93], human-computer interfaces of hypermedia applications are usually built in their entirety using implementation- and environment-dependent tools. Current methods tend to emphasize conceptual and navigational design, but the design problems associated with the interface are usually neglected.

Modern GUI libraries do simplify the task of defining media objects and scripting the interface transformations, but 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; interface design decisions are documented and can be used both as a testbed to validate the implementation and as a reference during maintenance.

In this paper we present the Abstract Data View (ADV) approach. The method associated with ADVs includes an object-oriented approach for the design of the user interfaces which can be applied in hypermedia applications in the context of the Object-Oriented Hypermedia Design Model (OOHDM, [Schwabe94, Schwabe95]).

In addition, we describe OOHDM to provide a contex for our presentation and follow this presentation with a brief overview of the Abstract Data View Design Model. Our approach for designing the user interface of hypermedia applications is then discussed. Further work on reuse-in-the-large of interface objects and interaction styles is then analyzed. In the last section we briefly discuss related work and present some concluding remarks.

2. OVERVIEW OF OOHDM

The Object-Oriented Hypermedia Design Method is a four-step process that is a mix of iterative, incremental and prototype-based styles of development. In each step a model is built or enriched; and after the last step, we have enough information to implement the hypermedia application.

2.1 Step 1: Conceptual Design

In this step a model of the application domain is built using well known object-oriented modeling principles [Rumbaugh 91] augmented with some primitives such as attribute perspectives and sub-systems. Conceptual classes may be built using aggregation and generalization/specialization hierarchies. There is no concern for the types of users and tasks, only for the application domain semantics. The product of this step is a class and instance schema constructed from Sub-Systems, Classes and Relationships.

2.2 Step 2: Navigational Design

One of the key distinguishing features of hypemedia applications is the notion of navigation. In OOHDM, an application is seen as a navigational view over the conceptual domain. This view is constructed from the viewpoint of the types of intended users, and the set of tasks they are to perform. Different navigational models may be built for the same conceptual schema thus expressing different views (applications) on the same domain. We describe the navigational structure of a hypermedia application by defining navigational classes that reflect the chosen view over the application domain. The navigational structure is defined in terms of navigational contexts, which are induced (in different ways depending on the type of class) from navigation classes such as Nodes, Links, Indices, and Guided Tours.

Nodes represent logical "views" on conceptual classes defined in the conceptual schema through a mapping that provides a "cut-and-paste" of conceptual attributes. Nodes may be atomic or composites [Gronbaek94], specifying which attributes and abstract anchors will be available in each navigational context. Nodes are described by attributes and anchors for outgoing links if they are atomic and as a set of component nodes if they are composites, with an attribute-anchor specification for each possible navigational context in which a node may appear [Schwabe95] .

Links are derived from conceptual relationships in the schema, and may be one-to-one or one-to-many. The result of traversing a link is expressed by defining the navigational semantics either procedurally as the link's behavior or using an object-oriented state transition machine. Transformations of the Navigational Space, that is the current set of accessible navigational objects, are specified by an object-oriented state-transition machine similar to a statechart. This allows reasoning about transformations in the Navigational Space that is independent of the domain model and the interface.

2.3 Step 3: Abstract Interface Design

In this step the abstract interface model is built and includes the user's perception of the user interface, the appearance of different navigational objects, the interface objects which activate navigation, the synchronization of multimedia interface objects, and the interface transformations which occur. A clean separation between both navigational aspects and abstract interface design, supports the concept of building different interfaces for the same navigational model thus conforming with varying user needs/preferences or user-interface technology. The remainder of this article will discuss this step in more detail.

2.4 Step 4: Implementation

By mapping the navigational and abstract interface models into concrete objects available in the chosen implementation environment, the author produces the actual operational hypermedia system. The model generated after performing steps 1 through 3 can be implemented in a straightforward manner using available hypermedia platforms such as Hypercard, Toolbook, MacWeb, KMS, Guide, and Microcosm. By using a set of uniform modeling constructs (objects and classes), our approach allows a smooth transition from domain modeling to navigational and interface design and finally implementation.

3. ISSUES IN BUILDING THE ABSTRACT INTERFACE MODEL

In order to specify the abstract interface model we need to define interface metaphors, and describe their static and dynamic properties and their relationships with the navigational model in an implementation-independent way [Vaanamen93]. 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 to access structures such as an icon on a city map acting as an index to monuments.
- other interface objects for providing the various 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 behaviour of different interface objects.
- and 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 [Garzotto91, Garzotto93], EORM

[Lange94] or RMD [Balasubramaniam94]. Furthermore it can be used as a design front-end to advanced object-oriented hypermedia environments such as MacWeb [Nanard91], or DEVISE [Gronbaek94].

4. OVERVIEW OF THE ABSTRACT DATA VIEW (ADV) DESIGN MODEL

The ADV design model was originally created 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 [Carneiro94a, Cowan93, Cowan95]. 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 usually 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. ADVs as media transformers can be viewed as 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. We believe this approach does not limit the applicability to the design of a broad range of software systems.

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 1.

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.

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.

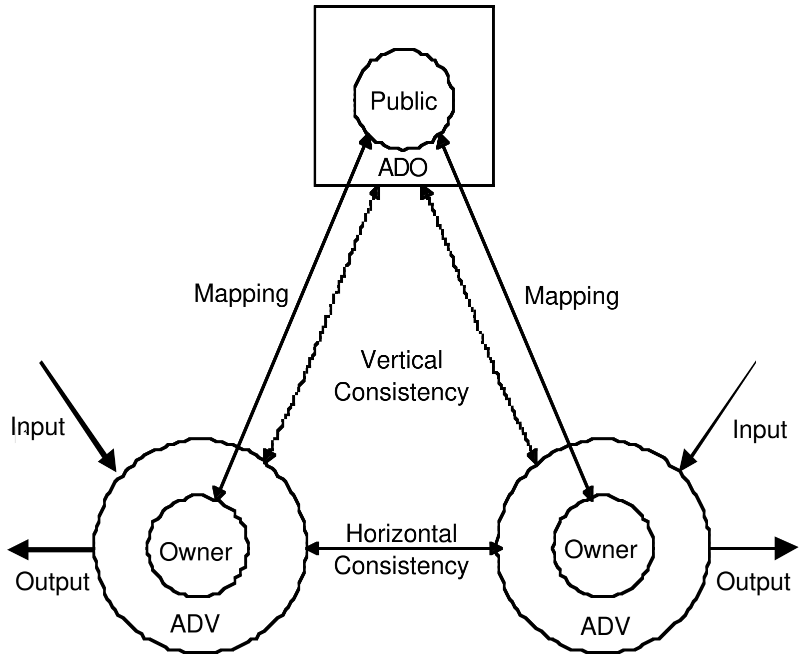


Figure 1: ADVs as user interfaces

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. For example, Figure 2 shows how an ADV describing a painting could be made out of 3 ADVs, containing an image, text, and a button. Furthermore, ADVs may be organized in generalization/specialization hierarchies that provide a powerful conceptual framework for defining hierarchies of interface objects.

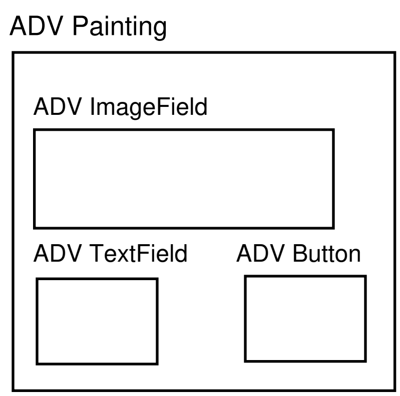


Figure 2: Composition of ADVs.

In this paper we use JASMINUM [Carneiro94b], a visual formalism based on state machines that supports the graphical specification of ADV-based designs. JASMINUM uses Configuration Diagrams and ADVcharts for specifying both static and dynamic relationships among ADVs and ADOs.

4.1 Configuration Diagrams

Configuration diagrams have been originally defined in the context of ObjectCharts [Coleman92]. 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. (See Figure 3).

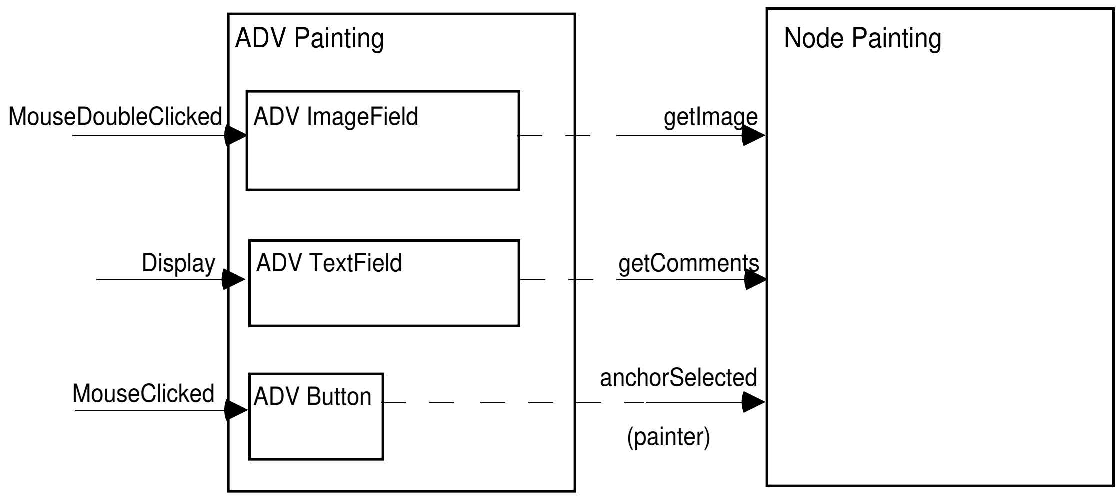


Figure 3: A Configuration Diagram

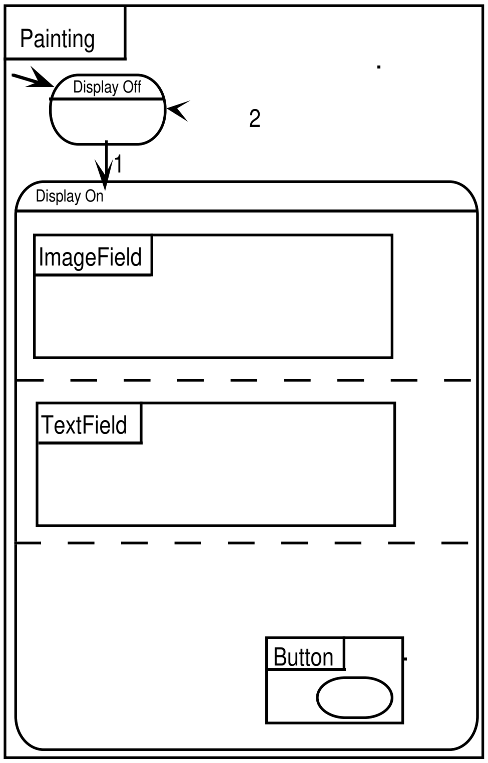
In Figure 3, ADV Painting can react to external events: MouseClicked, Display, and MouseDoubleClicked and communicates with its owner (an ADO that is a node Painting) by sending it the messages anchorSelected, getComments and getImage. In this example node Painting has only an anchor, painter.

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.

4.2 ADV Charts

ADVcharts are a generalisation of Statecharts [Harel87] and ObjectCharts [Coleman92] 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 as shown in Figure 4 Nesting states are the expression of behavioural nesting while nesting of ADVs expresses structural nesting.



Transitions

1:
Event: Display
Pre-Cond:
Post-Cond: perceptionContext = perceptionContext + Painting

2:
Event: MouseClicked
Pre-Cond: focus (Button)
Post-Cond: perceptionContext = perceptionContext - Painting; owner.anchorSelected (painter)

Figure 4: An ADV Chart

In Figure 4 ADV Painting may be either in state Display Off or Display On, and Transition 1 will be caused by the external event: Display. In the notation in Figure 4, state Display On is an AND of three sub-states, corresponding to ADVs ImageField, TextField and Button. Transition 2 from (an internal state of) ADV Button to DisplayOff is triggered when the event MouseClicked is generated by the user on ADV Button; this transition is expressed using the function focus (meaning the cursor is located on the corresponding ADV). This transition causes the current painting to be no longer perceivable (perceptionContext = perceptionContext - Painting) and the operation anchorSelected (painter) is applied to the ADV's owner. ADVchart in Figure 4 could be further refined by showing internal states of ADVs ImageField and TextField, by expressing the fact that when ImageField is MouseDoubleClicked (focus on Image) transition normal --> zoomed is triggered and ImageField is zoomed. When ImageField is in state zoomed, the event MouseDoubleClicked triggers transition zoomed --> normal.

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

As shown in Figure 4, transitions in ADVcharts are annotated with the event(s) that causes the transition, a list of pre-conditions that must be satisfied in order for the transition to fire and the list of actions to be executed each time the transition is fired, in the form of post-conditions. As previoulsy shown, ADVcharts allow the expression of common interface transformations in hypermedia applications. Figure 5 shows the definition of the ADV Application as the aggregation of three ADVs: AnchoredText, Button1 and Button2, while Figure 6 illustrates the ADVchart of the Application.

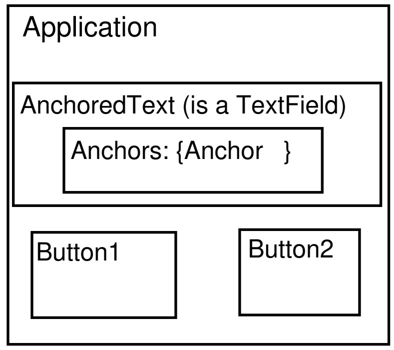


Figure 5 Definition of ADV Application

In Figure 5 we show the way in which we specialize an ADV; AnchoredText is a specialization of TextField; it inherits the functionality of TextField and enriches TextField with a set of Anchors. Anchors will react to event MouseClicked by sending a message to the application's owner.

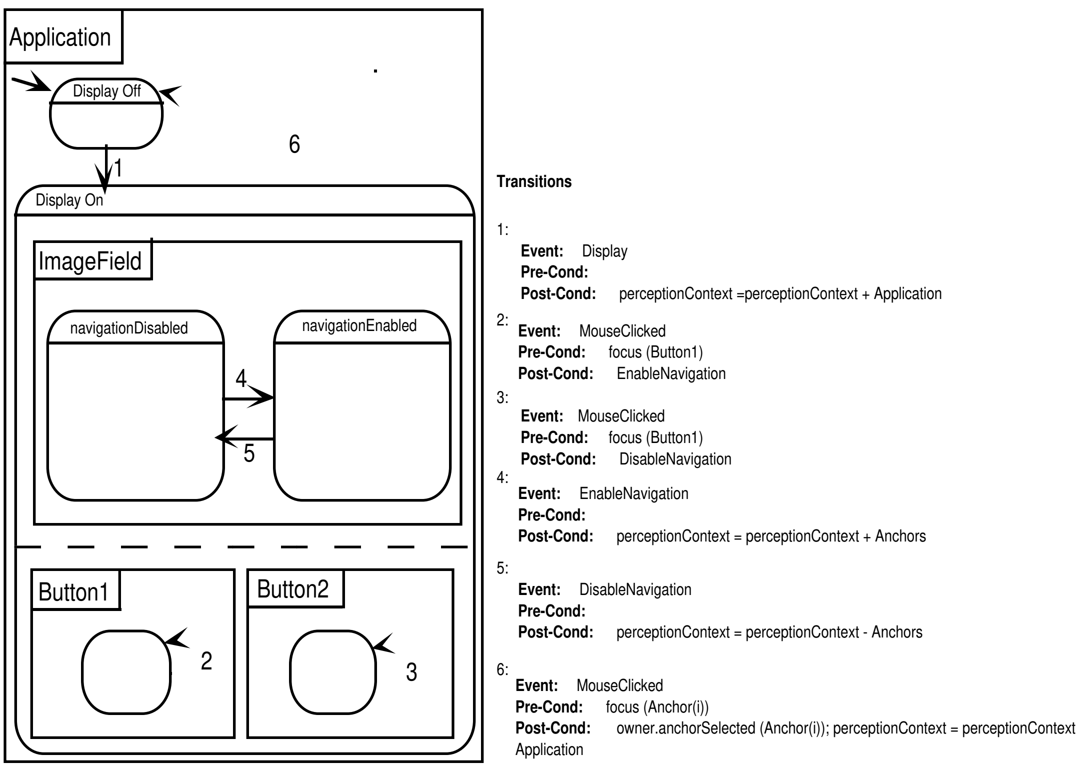


Figure 6: ADV Chart for Application

In the ADVchart of Figure 6 the ADV AnchoredText is specified as having two states: navigationDisabled and navigationEnabled. Button1 and Button2 serve as controls for enabling and disabling navigation in the ADV AnchoredText. When the event MouseClicked is produced and focus is on Button1, transition 2 is triggered and event enableNavigation is generated; this is called broadcast communication in StateCharts. Transition 4 fires when AnchoredText is in state navigationDisabled and event enableNavigation occurs. In this case, ADV Anchors is added to the perceptionContext and anchors are perceivable. In state navigationEnabled when the user clicks the mouse on some anchor, that is event MouseClicked occurs and focus is on some member of the collection Anchors, transition 6 is fired, sending the message anchorSelected to the owner and removing ADV Application from the perceptionContext. When the user clicks on Button2, event disableNavigation is produced (broadcast communication) and transition 5 is fired; ADV Anchor is removed from the perceptionContext, and anchors are no longer perceivable.

ADV Button1 and Button2 could have been refined if necessary; for example transition 2 and 3 could have caused the highlighting of corresponding buttons. An alternative design of this abstract interface could have been defining only one button enable/disable with two Xor states expressing different behavior when reacting to the event MouseClicked.

5. HYPERMEDIA INTERFACE SPECIFICATION

To make this presentation concrete we will show some examples using a real hypermedia application: Microsoft's Art Gallery. In order to be concise we do not analyze navigational contexts in Art Gallery as in [Schwabe95b], and will concentrate on describing the abstract interface model of the application.

In Figure 7 we show an outline of the navigational scheme of Art Gallery as described using OODHM.

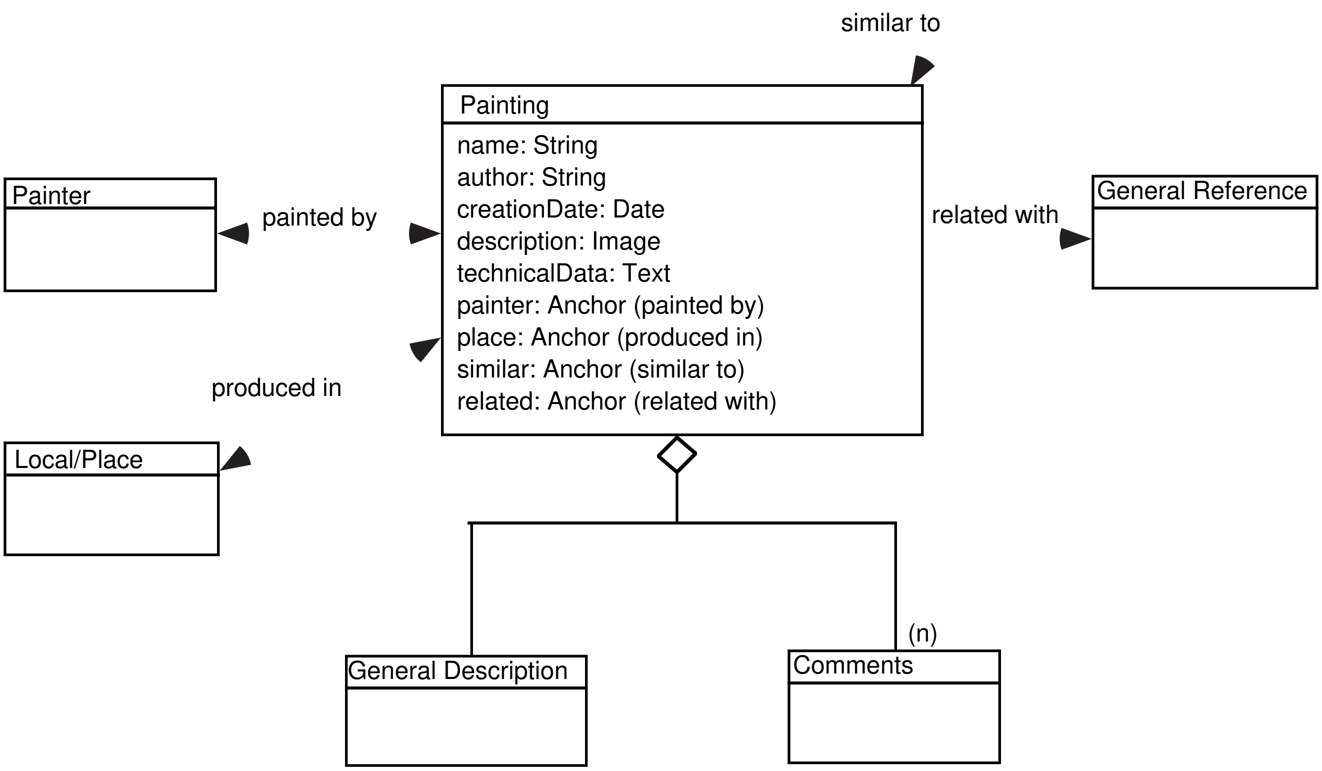


Figure 7: Navigational Scheme of Art Gallery (focusing Paintings)

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.

5.1 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 8 we show an outline of ADV Art Gallery, using the ADVchart notation.

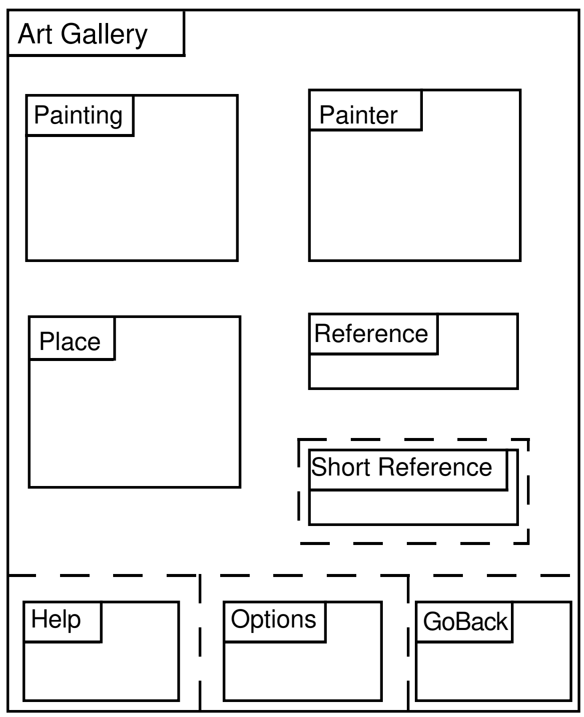


Figure 8: ADVs in Art Gallery

In Figure 8, 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.

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 5 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 modern hypermedia environments provide similar functionality to those just described, the ADV design formalism allows us to describe the interface structure and behaviour 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 indices 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 table-top, 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 link's anchor. We use Configuration Diagrams as the formalism to express behavioral relationships between interface objects and navigational objects. In Figure 9 we show the Configuration Diagram for ADV Painting.

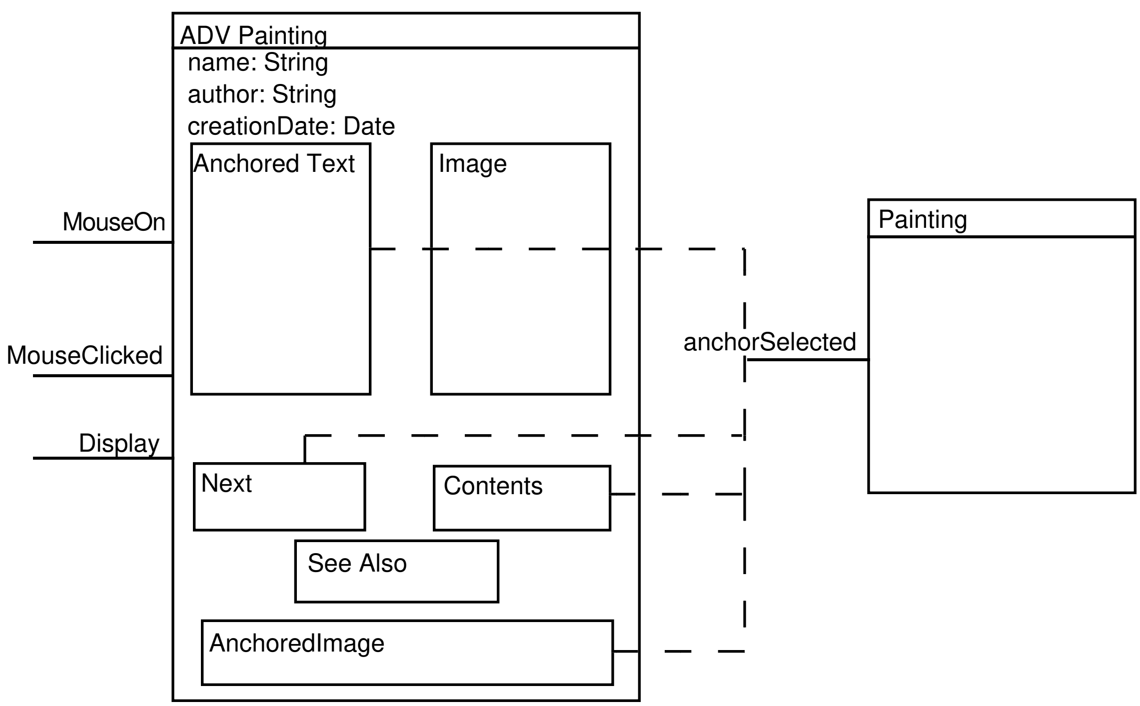


Figure 9: Configuration Diagram for ADV Painting

In Figure 9 ADV Painting is composed of several lower-level ADVs; for example Next and Contents are Buttons and when clicked they initiate navigation; AnchoredText is derived by sub-typing from ADV Text as in Figure 5, while AnchoredImage is derived from Image in the same way. "See Also" is derived from Button by sub-typing and its response to MouseClicked is refined: when it is clicked by making AnchoredImage perceivable as shown in Figure 10 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 10 we present ADVs

Art Gallery and Painting with part of the actual interface used in Microsoft's Art Gallery.

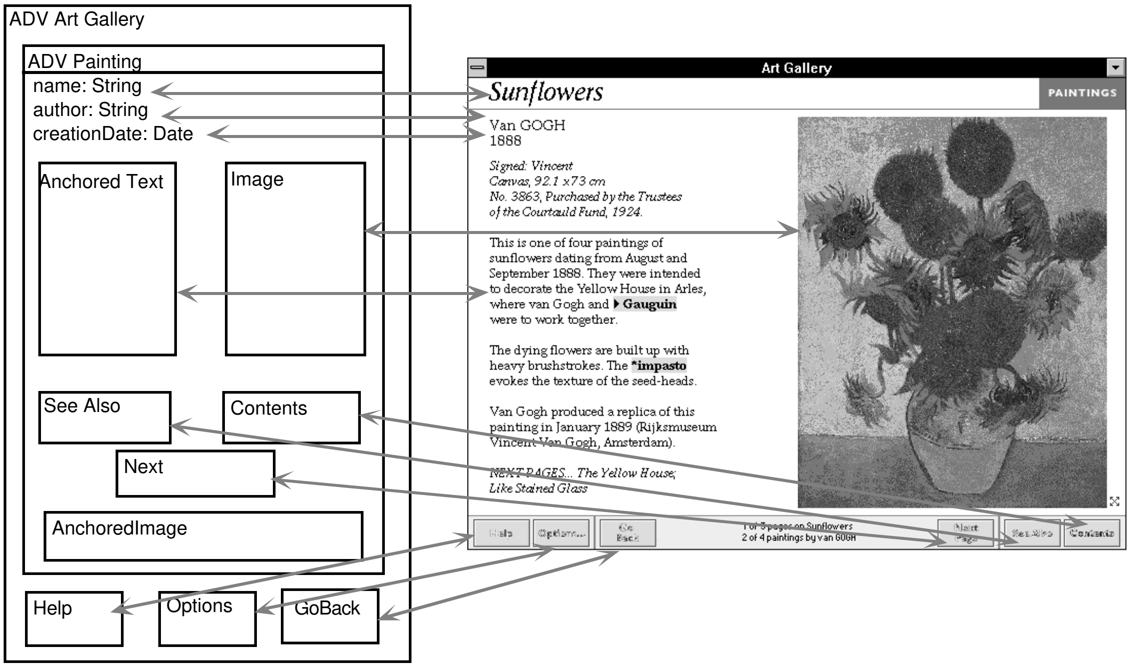


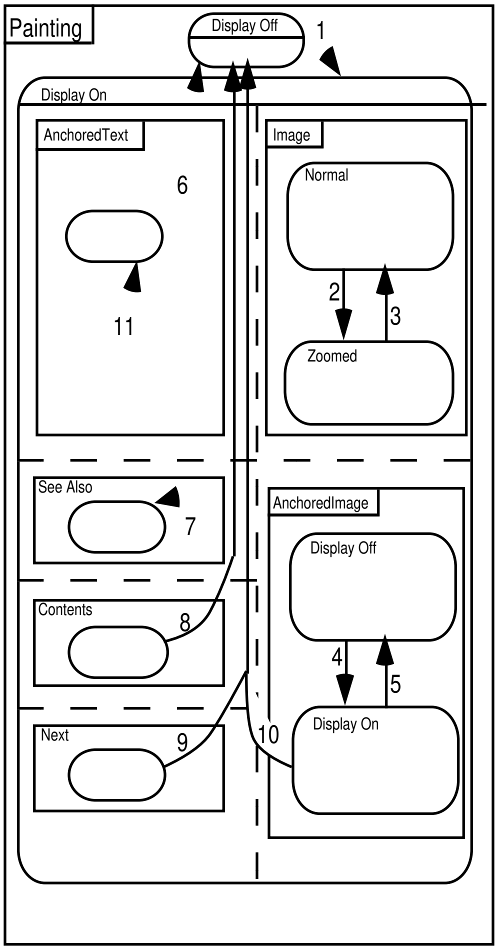
Figure 10: ADV Painting in Art Gallery

5.2 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".

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 that arising in a software engineering environment where external events 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. The ADVchart in Figure 11 shows the main aspects of the interface behavior for the ADV Painting. The other ADVs in Art Gallery are specified in a similar fashion, and the overall application behavior is defined as the combination of the individual behaviors of each part.



Transitions

1:
Event: Display
Pre-Cond:
Post-Cond: perceptionContext =perceptionContext + Painting

2:
Event: MouseClicked
Pre-Cond: focus (Image)
Post-Cond: self.DisplayZoomed

3:
Event: MouseClicked
Pre-Cond: focus (Image)
Post-Cond: self.DisplayNormal

4:
Event: ShowAnchors
Pre-Cond:
Post-Cond: perceptionContext = perceptionContext + AnchoredImage

5:
Event: MouseClicked
Pre-Cond: not (focus (AnchoredImage))
Post-Cond: perceptionContext = perceptionContext - AnchoredImage

6:
Event: MouseClicked
Pre-Cond: focus (Anchors (i))
Post-Cond: owner.anchorSelected (Anchors (i)) ; perceptionContext = perceptionContext - Painting

7:
Event: MouseClicked
Pre-Cond: focus (SeeAlso)
Post-Cond: ShowAnchors

8:
Event:: MouseClicked
Pre-Cond: focus (Contents)
Post-Cond: owner.anchorSelected (Contents); perceptionContext = perceptionContext - Painting

9:
Event:: MouseClicked
Pre-Cond: focus (Next)
Post-Cond: owner.anchorSelected (Next); perceptionContext = perceptionContext - Painting

10:
Event: MouseClicked
Pre-Cond: focus (Anchors(i))
Post-Cond: owner.anchorSelected (Anchors (i)); perceptionContext = perceptionContext - Painting

Figure 11: ADV Chart for Painting

Note that ADV AnchoredImage is only displayed (Transition 4) when ADV SeeAlso is "MouseClicked" (Transition 7) and event "ShowAnchors" is produced. In this way we can model "local scroll" effects like the one shown in Figure 10.

Transition 11 in Figure 11 above also deserves a careful comment. When it is triggered, the event "PopUpReference" is generated and ADV Short Reference (Figure 8) is displayed. As mentioned before, both Short Reference and Reference are different ADVs for Node Class General Reference and they are shown in different contexts and with different navigation metaphors. In the example Short Reference remains perceivable until event "MouseClicked" occurs and focus is not in Short Reference. Though we have not shown ADV Chart for ADV Art Gallery, the specification of transitions is similar to those in Figure 11.

5.3 Engineering the process of Interface Design

As we have shown, different aspects of hypermedia interfaces can be specified in a straightforward way using the ADV design approach. Using ADVs, Configuration Diagrams and ADVcharts, the hypermedia interface designer can map navigational objects to interface objects describing different interaction patterns that may occur in the application.

One of the most interesting consequences of our approach is that it can be the seed of a theory on the reuse of hypermedia interfaces and interaction styles. In current approaches the kind of reuse that may be achieved while building hypermedia applications is usually limited to reuse of interface widgets provided by a particular hypermedia environment. For example, in Toolbook we (re)use a library of interface objects such as buttons, menus, or text fields. In some cases we can also use templates that when completed yield simple navigational structures. It is clear however that critical design decisions taken while deciding the interaction and navigational style of an application usually remain hidden in code or are poorly documented.

Expert hypermedia designers do make good designs, and typically they don't solve every problem from scratch, but reuse solutions that they have used in the past. Such reuse of design experience is the most valuable kind of reuse. In the field of hypermedia interface design we could begin by reusing individual, low-level ADVs, their configuration diagrams and ADVcharts. The next step could be reusing a particular configuration of ADVs, for example a composite ADV implementing some standard hypermedia interface functions which embody some interface metaphor. We will see, however, that if we want to go a step further we will end with domain-specific ADVs that will be difficult to reuse in contexts that differ from the application for which they were originally devised. What we need to express and record are general solutions for recurrent problems that arise in different situations; we call them hypermedia interaction patterns.

Quoting the architect Christopher Alexander [Alexander77]: "each pattern describes a problem which occurs over and over again in our environment, and then describes the core of a solution to that problem in such a way that you can use this solution a million times over, without ever doing it the same way twice". Design Patterns are being increasingly used in software design [Gamma94] but remain unexplored in the hypermedia field and in particular for describing hypermedia interface and navigational styles. Though not the main focus of our paper we will next show a simple navigational and interaction pattern and indicate that it can be easily described using the ADV design approach. (See [Gamma94] for a comprehensive description of design patterns).

A common situation found in many hypermedia applications is the requirement to record, in a user perceptible manner, the state of the navigation. In addition, as the navigation progresses, this record is automatically updated. For example, suppose we have a hypermedia application showing European cities. We are navigating through them using different access structures and relationships. We may reach the same city through different navigation paths and we want to know which cities we have already visited. This can be achieved by having a map of Europe always visible in some portion of the screen. Whenever we visit a city, it is highlighted in this map, so we can instantly know which cities we have already visited during navigation.

To capture this situation, we must define a pattern that may be used whenever we want to have a record of the nodes we are visiting while navigating the hypermedia space. We want this record to be independent of the nodes, in that we want to change its style according to the user's needs and preferences but so that the same design style can be used in different applications.

This pattern may be defined as follows. For each Node Class in the Navigational Model we will have a corresponding ADV in the interface model. Define an abstract ADV: Abstract Record; it must be able to react (at least) to events Display, Highlight, and Reset. For each type of concrete Record, refine Abstract Record by defining its appearance and implementing the corresponding (refined) ADVcharts.

Define ADV MonitoredHypermedia as a composite with the ADV for Application and the Concrete Record as shown in Figure 12. Application is in turn a composite of MonitoredADV and perhaps other interface objects.

Note that structural details of ADVs internal to Application are not relevant in this discussion. In particular, if we instantiate this pattern for the previously mentioned application, we will have ConcreteRecord = EuropeanMap, and Application will contain ADVs for Cities (MonitoredADV) and other interface objects.

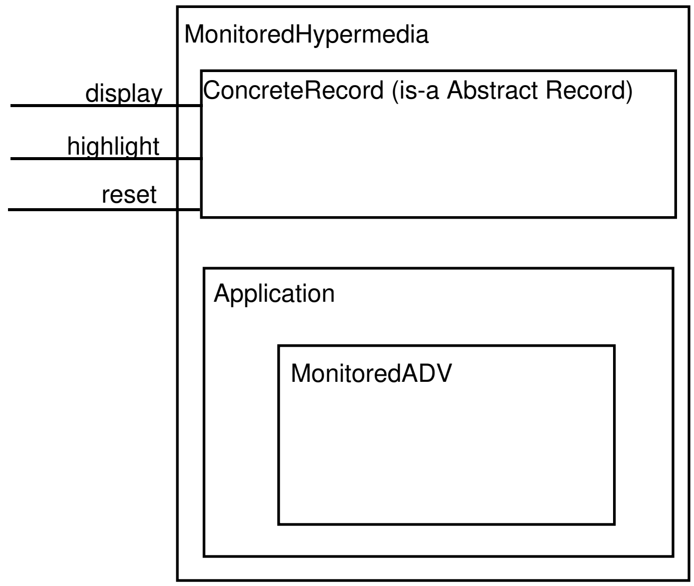


Figure 12 . Structure of ADV MonitoredHypermedia

The behavior of MonitoredHypermedia is as follows: each time a Node in the hypermedia application is activated, that is the corresponding MonitoredADV is displayed, the event "Highlight" is generated with the node-id as the argument. ADV ConcreteRecord treats this event by showing that node-id has been reached. Note that different implementations of ConcreteRecord may implement the handler for "Highlight" in different ways. Furthermore, the representations of nodes in the Record may be entities such as small icons, strings, or points in a map each of which is one possible ADV for a Node. In the same way, sub-types of MonitoredADV may be built if we want to monitor different types of nodes, such as cities and tourist attractions. In Figure 13 we show the ADV chart for MonitoredHypermedia.

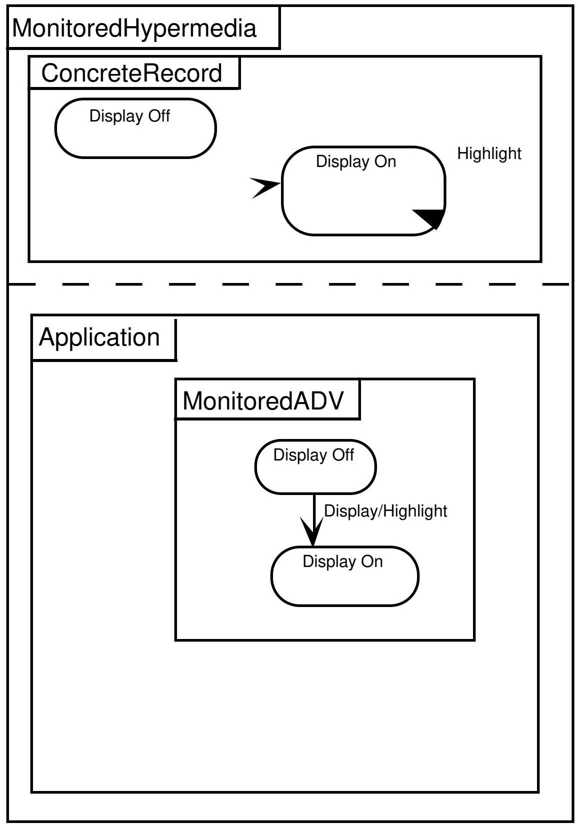


Figure 13. ADV Chart for Pattern MonitoredHypermedia

In Figure 13 a scheme of collaborations between ADVs in MonitoredHypermedia is shown. In ADVcharts notation as well as StateCharts notation, the transition from the inside of state DisplayOn to itself in ConcreteRecord represents the fact that when event "Highlight" occurs (broadcast by MonitoredADV), a transition from some internal state in DisplayOn to another is triggered, implementing the desired functionality. In the same way, each time a MonitoredADV such as an ADV for cities is displayed, that is transition Display Off -> Display On is triggered, we have specified that event "Highlight" is broadcast. Note that we could have specified many Records each one implementing its own handler for "Highlight"

When building a concrete application we use the previously described pattern by matching our concrete ADVs to abstract ones in the pattern and implementing the schematic behaviour in Figure 13 in the context of our application by following the naming conventions of the application.

Understanding and applying hypermedia interaction patterns is useful even when re-engineering existing applications. In Art Gallery for example we could enrich the interface by mantaining a record of the pictures the user has explored. Instantiating the MonitoredHypermedia interaction pattern will yield a configuration diagram similar to the one presented in Figure 9 including the ExploredPictures ADV. We would also modify the ADVchart in Figure 11 by including as a post-condition in transition 1:"Highlight (owner)" . ExploredPictures in turn will react to "Highlight" by adding the corresponding picture to a list of visited master pieces and making the list visible.

6. CONCLUSIONS

We have presented an object-oriented design model for specifying the interface of hypermedia applications. The model uses Abstract Data Views, Configuration Diagrams and ADVcharts for expressing in an abstract, implementation-independent way different aspects of the user interface of a hypermedia application. These aspects include the appearance of navigational objects, the relationships between "active" interface objects and abstract anchors and the interface transformations occurring when the user interacts with the application. Though being based on formal constructs, our approach yields abstract interface schemes that are easily mapped to conventional hypermedia environments because they model the interface in a manner similar to the approach currently used in modern GUI systems. Moreover it is well suited for design environments emphasizing incremental and opportunistic design of the hypertext structure [Nanard94].

Other authors have proposed the use of formal models for specifying hypertext semantics. In [Zheng92] for example, StateCharts are used for modeling different browsing semantics in hypertext. Our work is similar in that it uses ADVcharts (a derivative of StateCharts) for expressing the dynamics of the application. However it is different because it clearly separates navigational behavior (a concern of navigational design) from interface specification; moreover as ADVs may be composed to form higher level ADVs, nesting in ADVcharts not only indicate composition by behavior as in StateCharts, but also structural composition.

In [Vaananen93] a formal model identifying the relevant processing and descriptive elements required for specifying the interaction behavior of hypertext structures is presented. The model is based on the use of production rules and as such it uses an approach that is different from ours. We believe that Configuration Diagrams and ADVcharts present a more natural and intuitive representation model though they are still formal.

We are now working on the definition of a set of standard ADVs with their associated Configuration Diagrams and ADVcharts to be used as a design library. We are also analyzing hypermedia applications looking for recurrent navigational, interface and interaction patterns in order to define a catalog of hypermedia patterns. Building new hypermedia applications by combining well-known patterns should yield more robust and easy to maintain applications.

7. REFERENCES

[Alexander77] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King and S. Angel: "A Pattern Language". Oxford University Press, New York 1977.

[Balasubramaniam94] P. Balasubramaniam, T. Isakowitz and E. Stohr: "Designing Hypermedia Applications", Proceedings of the 27th. Hawaii International Conference on System Sciences, Hawaii, Jan. 1994.

[Carneiro94a] L.M.F. Carneiro, 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.

[Carneiro94b] L.M.F. Carneiro-Coffin: "Jasminum: Joining ADVs and State Machines in a Notation for User-Interface Modeling". PHD Thesis, University of Waterloo, 1994.

[Coleman92] 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.

[Cowan93] D. D. Cowan R. Ierusalimschy, C.J.P. Lucena and T.M. Stepien: "Abstract Data Views". Structured Programming, 14(1):1-13, January 1993.

[Cowan95] D. D. Cowan, ; C. J. P.Lucena, ; "Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse". IEEE Transactions on Software Engineering, Vol.21, No.3, March 1995.

[Gamma94] E. Gamma, R. Helm, R. Johnson and J. Vlissides: "Design Patterns: Elements of reusable object-oriented software", Addison Wesley, 1994

[Garzotto91] F. Garzotto, P. Paolini and D. Schwabe: "HDM- A Model for the Design of Hypertext Applications", Proceedings of Hypertext'1991 , ACM Press. pp. 313.

[Garzotto93] F. Garzotto, D. Schwabe, P. Paolini: "HDM- A Model Based Approach to Hypermedia Application Design", ACM Transaction on Information Systems, Vol. 11, #1, Jan. 1993, pp. 1-26.

[Gronbaek94] K. Gronbaek: "Composites in a Dexter-Based Hypermedia Framework", Proceedings of the ACM European Conference on Hypermedia Technology, Edinburgh, 1994.

[Hannemann93] 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.

[Harel87] 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.

[Kirste93] T. Kirste: "Some issues of defining a user interface with general purpose hypermedia toolkits". Workshop in Methodological Issues on the Design of Hypertext-based User Interfaces, Darmstadt, Germany, July 1993.

[Lange94] D. Lange: "An Object-Oriented design method for hypermedia information systems", Proceedings of the 27th. Annual Hawaii International Conference on System Science, January 1994.

[Nanard93] J. Nanard and M. Nanard. "Using Structured Types to Incorporate Knowledge in Hypertext, Third ACM Conferences on Hypertext Proceedings, Hypertext'91 ed. ACM Press. pp.. 329.

[Nanard94] J. Nanard and M. Nanard: "Key features in a Hypertext Workshop on Methodologies for Designing and developing Hypermedia Applications , Edinburgh, September 1994.

[Oliveira95] Simone Barbosa de Oliveira. "Modeling and Specification of Navigation in Hypermedia Applications", MSc Thesis, Dept. of Informatics, PUC-Rio, Feb. 1995 (in Portuguese).

[Rumbauh91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W.Lorensen: "Object Oriented Modeling and Design", Prentice Hall Inc. 1991.

[Schuler94] W. Schuler: "A Design Space for Hypermedia Interface". Workshop on Methodologies for Designing and developing Hypermedia Applications , Edinburgh, September 1994.

[Schwabe94] D. Schwabe and G. Rossi: "From Domain Models to Hypermedia Applications. An Object-Oriented Approach", International Workshop on Methodologies for Designing and developing Hypermedia Applications , Edinburgh, September 1994.

[Schwabe95] D. Schwabe and G. Rossi: "Building Hypermedia Applications as Navigational Views of Information Models", Proceedings of the Hawaii International Conference on System Sciences, Hawaii, January 1995.

[Vaananen93] K. Vaananen: "Metaphor-Based user interfaces for Hyperspaces". Workshop in Methodological Issues on the Design of Hypertext-based User Interfaces, Darmstadt, Germany, July 1993.

[Zheng92] Y. Zheng, M-C. Pong: "Using Statecharts to Model Hypertext". Proceedings of the ACM European Conference on Hypertext, Milano, December 1992.