Design Patterns for Object-Oriented Hypermedia Applications

Gustavo Rossi*, Alejandra Garrido and Sergio Carvalho**

LIFIA. Laboratorio de Investigación y Formación en Informática Avanzada.

Dpto. de Informática, Fac. de Cs. Exactas, Universidad Nacional de La Plata.

C.C. 11, (1900) La Plata, Buenos Aires, Argentina. Fax/Tel: +54-21-22 8252

E-mail: [grossi, garrido]@ada.info.unlp.edu.ar

* also CONICET-Argentina and Dto. Informatica, PUC-Rio, Brazil.

** Dto. Informatica, PUC-Rio, Brazil.

E-mail: sergio@inf.puc-rio.br

Abstract

This chapter proposes design patterns for object-oriented applications enhanced with hypermedia functionality. We briefly discuss our problem: building a software architecture for seamlessly extending object-oriented applications with hypermedia interface and navigational styles. Two new design patterns named 'Navigation Strategy' and 'Navigation Observer' are presented, showing how they are used to design flexible and extensible navigational structures.

1 Introduction

Hypermedia applications are characterized by the representation of unstructured information, chunked in a collection of nodes that are related through links. These links are navigated by the user with or without a predefined order, drawing the desired path. Besides, many of the features of hypermedia applications may be included in information systems to increase their utility and usability, leading to a new concept in the hypermedia arena called "hypermedia functionality" [Oinas-Kukkonen94]. Some of these features are:

-the representation of unstructured information, allowing to relate any piece of information with any other, or to define annotations;

-the ability to link large number of information units to be shared by collaborative groups [Balasubramanian94];

-the enhancement of Graphical User Interfaces, providing each user with information that suits his/her needs when navigating, using maps and browsers for viewing and exploring the application's database from different viewpoints and abstraction levels.

Hypermedia functionality is particularly helpful in software engineering environments or decision support systems [Bigelow88], [IEEE92], where it is necessary to combine formal with informal knowledge (e.g. relating the formal definition of a software component with its design rationale).

Some commercial applications such as Microsoft's Windows standard Help System and Lotus self-Learning Guide make use of hypermedia facilities by giving the user the chance of exploring an information base, built as a stand-alone hypermedia subsystem of the application that retains the control. Link-servers like Microcosm [Davis92] are a different approach that allows inter-application

linking. In our approach, however, we are interested in a richer use of hypermedia: as a way of improving access to information resources available in an object-oriented application, combining that information with the application's specific behavior.

A key problem in designing these hybrid applications is how to factor out the hypermedia structure from the application specific behavior, in such a way to allow different views of application objects (representing those views as hypermedia nodes) and to navigate among them in a passive mode, without affecting the objects' semantics, or in an active mode, triggering the objects' methods. For example in CASE environments, we could explore relationships among design documents and may or may not be able to affect them (depending on our role in the project).

In this context we decided to build an object-oriented framework that allows combining navigational features with the functionality of an object-oriented application. While constructing the framework, different design patterns were applied and some others were created; we are now aimed at designing a pattern language in the hypermedia domain.

The hypermedia framework has been used in different application domains such as learning environments [Leonardi94], sophisticated CASE tools [Alvarez95] and a design environment for the Object-Oriented Hypermedia Design Model [Schwabe95a], [Schwabe95b].

Section 2 of this chapter briefly outlines the architecture and use of the hypermedia framework. Section 3 and 4 present two design patterns that were found while trying to apply existing patterns and needing some variations. Finally, some concluding remarks are presented. The patterns referenced along the chapter can be found in [Gamma94] unless explicitly mentioned.

2 An Object-Oriented Framework for Hypermedia

We have designed and implemented an object-oriented framework defining the abstract design of applications combining rich navigational and interface styles with usual objects' behavior and semantic [Rossi94]. The user of this framework will be able to add hypermedia functionality to an object-oriented application, or may create a hypermedia application from scratch.

The application's architecture as induced by the framework is divided in three levels: the Object Level, the Hypermedia Level and the Interface Level. Different implementations may further divide the Interface component (using for example an extension of the Model-View-Controller framework [Krasner88]). As we mentioned above, the hypermedia framework allows the mapping of different views of application level objects to hypermedia level nodes. An object's view in this context can be regarded as the ‘face’ that the object shows with a subset of its data and behavior, for a specific user role or profile. Moreover, hypermedia nodes may be built by just “observing” single application objects or as sophisticated views on sets of related objects. Thus, the user of the framework will have to define a node class for each different application class (or set of classes) such that its instances are going to be displayed in hypermedia nodes. Navigation is supported by links that may be statically defined or may be computed on line during navigation.

Design patterns played a significant role in the framework architecture, because they helped us to improve the design and to communicate with a common vocabulary of design artifacts. Many patterns of the [Gamma94] catalog were applied, such as Observer, to define the connection among objects and nodes and among nodes and their interfaces; Composite, in order to aggregate nodes, and Mediator and Adapter while implementing the framework using Tool [Carvalho92], a strong typed object-oriented language. We are now working on the definition of a pattern language in the hypermedia domain; such language will not only deal with aspects related with the association of objects with nodes and links, but

mainly with the navigational structure of the application. In this context, new and specific design patterns will arise.

The following sections show an outline of two design patterns that, though being a specialization of existing patterns, have their own motivation and applicability. The first one, ‘Navigation Strategy’, addresses the computation of link endpoints while navigating. The second, ‘Navigation Observer’, deals with the process of recording visited nodes and links in a history of navigation. It also defines a separated hierarchy of viewers for the history.

Though these are patterns specific to the hypermedia domain, we found that they can be easily generalized to be used in other domains, as it is mentioned in the ‘Applicability’ section of each pattern.

3 Navigation Strategy

3.1 Intent

Define a family of algorithms that decouples the activation of hypermedia links from the computation of their endpoints, thus allowing different means of obtaining the endpoints, and their lazy creation.

3.2 Motivation

In conventional hypermedia applications as for example Microsoft's Art Gallery, links are hard-coded from the source node to the target. However, when the endpoint of a link depends not only on the target node but also on context information, or if it must be computed dynamically, it is necessary to perform some testing in the source node (or anchor). This situation is more complex in applications such as CASE tools or decision support environments allowing navigation across design documents, because the target of the link might be created on demand or just remain unspecified for later definition. Suppose for example that we want to navigate from the CRC card of a class to the Class Browser showing its current version. It may be necessary to query the version manager to obtain the link's target. Moreover, if there is no implementation for that class, traversing that link may mean creating the target node by opening the Browser in editing mode.

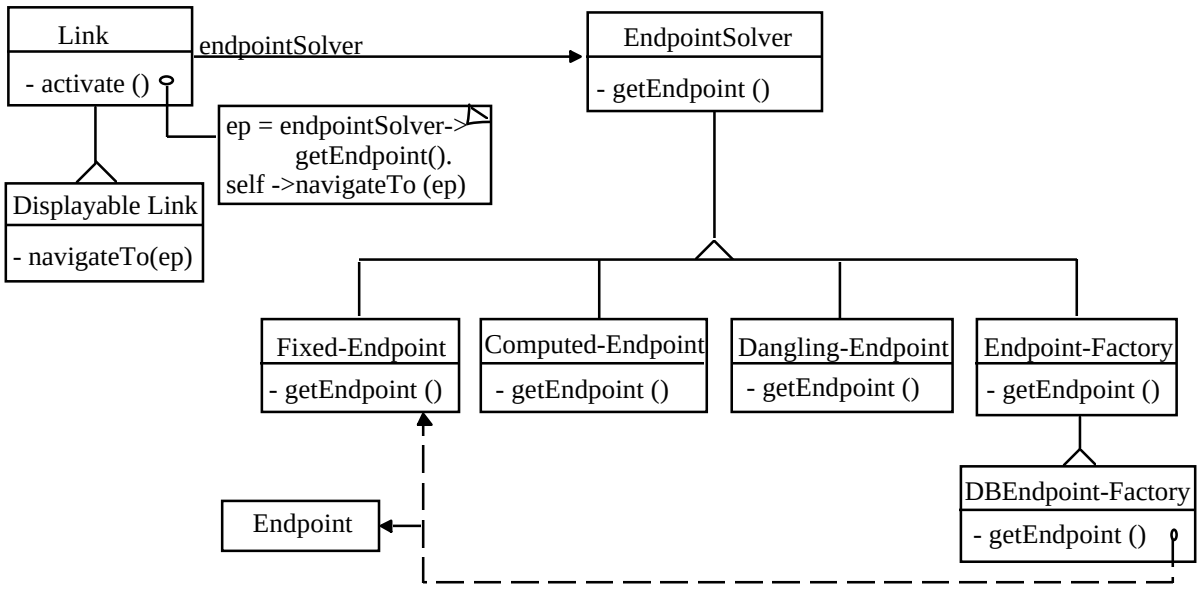

Figure 1: EndpointSolver hierarchy and its relation with Link class

In a first approach to solve this problem we intended to use the Strategy pattern, which models the encapsulation of different algorithms in a separated hierarchy, letting them vary independently from clients that use them. In this way, each link would be configured with the needed algorithm. But as we also wanted to support lazy creation of link endpoints, we have extended the pattern with a subhierarchy of Abstract Factories. In this way, the strategy algorithm encapsulated in each factory class becomes a FactoryMethod. The Concrete Factory classes will be associated with each different Endpoint class (node class) and with another Concrete Strategy class that will hold the endpoint computation once it is created. This new pattern was called Navigation Strategy.

Figure 1 shows an example of the Navigation Strategy pattern. The diagram shows the different Concrete Navigation Strategy classes (subclasses of EndpointSolver) that were defined in the hypermedia framework. The only concrete Endpoint-Factory subclass showed for simplicity is related to the creation of endpoints for those nodes stored in a database. The dashed lines mean that the Factory Method (getEndpoint) will create an endpoint and a Fixed-Endpoint solver that will contain it once created. The Link class is also subclassified with the class of links that display some information when they are navigated.

3.3 Applicability

Use the Navigation Strategy pattern in the hypermedia domain when

- you need different variants of the algorithm that computes the endpoint of a link;

- you need lazy creation of endpoints, e.g., in order to improve memory requirements;

In a general context, the Navigation Strategy pattern may be used when there is a need to establish a relation between two or more objects at different times, i.e., statically or dynamically, where the later can also lead to the lazy creation of the related object. It may be also useful when some objects are stored in a database and you want to retrieve them in memory only when a related object needs them.

3.4 Structure

The structure of Navigation Strategy is shown below in Figure 2.

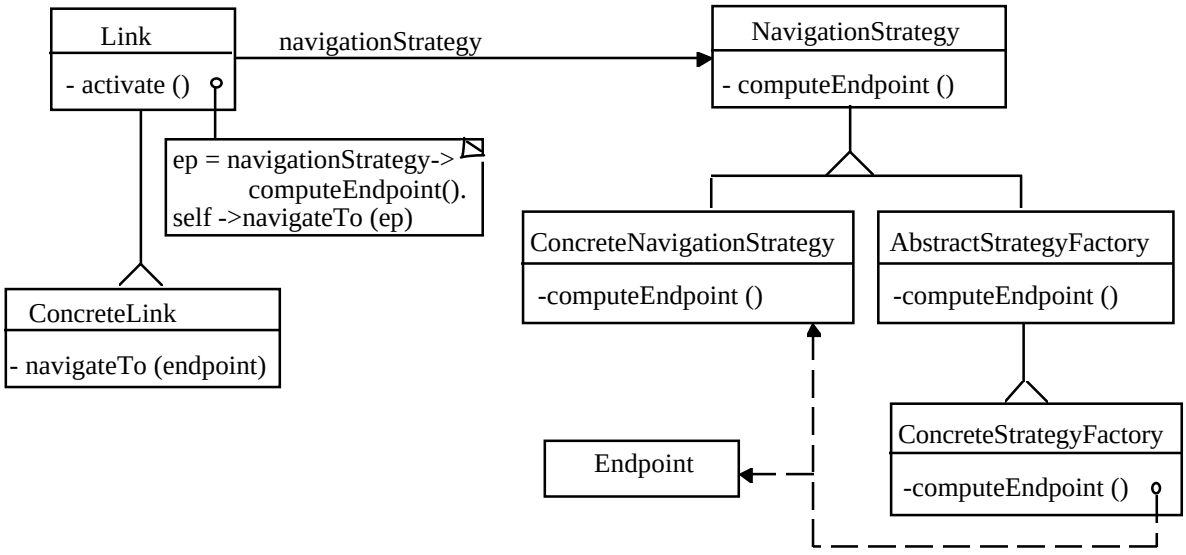

Figure 2: Navigation Strategy pattern

3.5 Participants

- **Link**

  - Maintains a reference to a navigationStrategy object.

  - Defines a Template Method to perform its activation, which will get the endpoint from its navigationStrategy object, and then navigate to it.

- **ConcreteLink**

  - Implements the navigateTo method. It may define different ways of performing navigation (e.g., displaying link attributes).

- **NavigationStrategy**

  - Declares an interface common to all supported algorithms that compute the link endpoint.

- **ConcreteNavigationStrategy**

  - Implements the algorithm that obtains the link endpoint. (For example it may be a fixed or computed endpoint).

- **AbstractStrategyFactory**

  - The algorithm that obtains the link endpoint acts as a Factory Method that creates the endpoint and an instance of another ConcreteNavigationStrategy.

- **ConcreteStrategyFactory**

  - Is the factory of endpoints for nodes of certain type and instances of a ConcreteNavigationStrategy, with which it is related. This relation product-factory is shown in Figure 2 with a dashed arrowhead from the factory method.

3.6 Collaborations

- Link and NavigationStrategy interact to implement the navigation through the web of nodes. When activated, the link asks its navigationStrategy to get its endpoint, by passing the necessary arguments, and later performs the navigation process with the obtained endpoint.
- Links' anchors (in source nodes) are the clients that activate links. However, they are not responsible of the ConcreteNavigationStrategy instance creation or selection. They are just a means of accessing and activating the link.
- Concrete Strategy Factories create both the endpoint and an instance of another Concrete Navigation Strategy for later access.

3.7 Consequences

- NavigationStrategy offers the same benefits as the Strategy pattern, as an alternative to conditional statements or subclassing, but also allows the dynamic creation and linking of nodes in an active hypermedia environment.
- This separation in the navigation process allows us to define different kinds of links, such as those that display themselves while navigating, and different types of endpoints, as single or multiple.

- The NavigationStrategyFactory may also improve memory requirements by deferring the retrieval of the target node for example when it is stored in a database, creating the associated endpoint in memory only when needed.
- Finally, the same drawbacks of Strategy can be found here, as an increased number of objects and communication overhead between Link and NavigationStrategy (in the case that the anchor is not needed). Also the subhierarchy of AbstractStrategyFactory can lead to class proliferation, as it is discussed in the Abstract Factory pattern, but this could be solved by using the Prototype pattern instead of Abstract Factory.

3.8 Implementation

The implementation issues of the Strategy pattern in [Gamma94] apply also to this one, but other considerations must be stated:

1-*Data exchange between Link and Strategy:* The only information that is normally exchanged between the link and its strategy is the link's anchor (from link) and the endpoint (from navigation strategy). This information is necessary when a link server is used or when the node has been dynamically computed or created.

2-*Endpoint and Node classes:* The Endpoint class may be further subclassified in SingleEndpoint (for one target node) and MultipleEndpoint (for a set of target nodes), so each link will be associated by way of navigationStrategy with only one endpoint. Moreover, each endpoint may contain some context and representation information in which the target node will be reached.

Node class will be further subclassified by the user of the framework for each different application class that he/she decides to observe in the hypermedia, as expressed in Section 2. This will imply a further subclassification of the AbstractStrategyFactory, associating each different ConcreteStrategyFactory with a different Node class. As discussed earlier, a different approach could be using prototypes for node creation, following the Prototype pattern.

3- In the case that nodes are extracted from a database, other considerations must be taken into account, as storage and update of nodes, but are out of the scope of this chapter.

3.9 Known uses

NavigationStrategy is widely used in modern hypermedia environments. For example in Microcosm [Davis92], an open hypermedia system that provides linking mechanisms among third party applications (like Microsoft's Word, Assimetrix's Toolbook, etc.), links are always dynamically extracted from a link database that contains information about the anchors' offsets and types.

In some proposed extensions to Netscape, for example [Hill95], it is possible to define new (private) links by using a similar mechanism as the one presented here. Moreover, the separation of links from documents allows the implementation of ‘generic’ links, i.e. those defined in terms of content rather than location, which can greatly ease the authoring of common links and reader-led navigation.

Some implementations of the Dexter Hypermedia Model [Halasz94], as for example [Grønbæk94], propose different alternatives for obtaining link endpoints, similar to the ones presented here.

Navigation Strategy may be used to improve the design of existing hypermedia applications. For example, in WWW viewers, the process of locating the endpoint of a link is often a non-atomic transaction (that may be even unsuccessful) and must be clearly separated from the activation process.

In the CASE environment presented in [Alvarez95], NavigationStrategy allows linking design documents with dynamically computed animations of those documents. The link endpoint may be fixed to another document, trigger the creation of a document or be created on demand.

3.10 Related Patterns

Navigation Strategy is similar to Strategy in that it allows defining a family of algorithms for computing endpoints, making them interchangeable and allowing the Link class to be extended independently of those algorithms. However, it differs from Strategy in that the former includes a subhierarchy of factories, in which the strategy algorithm acts as a Factory Method, thus allowing the lazy creation of endpoints and navigationStrategies. AbstractStrategyFactory is also similar to the Acceptor pattern [Schmidt95] in that both use lazy connection establishment; however, the former is similar to an Abstract Factory whereas Acceptor is not.

It also uses Template Method in Link for defining the abstract algorithm for performing navigation.

4 Navigation Observer

4.1 Intent

Decouple the navigation process from the perceivable record of the process. Navigation Observer simplifies the construction of navigation history viewers by separating the hypermedia components (nodes and links) from the objects that implement both the record of navigation and its appearance.

4.2 Motivation

Hypermedia applications should record in a user perceptible way the state of the navigation. As navigation progresses, this record must be updated automatically. For example suppose that the hypermedia application shows European cities and we are navigating through them using different indices and links. We may reach the same city through different navigation paths and may want to know the cities we have already visited. This can be achieved by having a map of Europe always visible in some part of the screen. Whenever we visit a city, it is highlighted in this map, so we can instantly know which cities have been already visited during the navigation. We could visualize which paths have been followed by displaying not only nodes (cities) but also links (roads). In some commercial hypermedia environments there is a pre-defined way for showing the history of navigation. For example Hypercard* provides a list of small pictures representing each card that has been accessed.

We could implement this behavior by requiring that objects representing cities (nodes in the hypermedia application) communicate with the object representing the map with the message *highlight(self)*. However this solution would make difficult the existence of different perceivable records by adding a strong dependence among nodes and viewers and would require modifying the hypermedia components for each new type of viewer that is defined.

The most convenient way to implement these history viewers is to use the Navigation Observer pattern. Navigation Observer is useful to have a perceivable record of the nodes and links visited while navigating the hypermedia space, and to make this record independent of them, changing its style accordingly to the user needs and preferences so that the same design and interface style can be reused in different applications.

4.3 Applicability

Use the Navigation Observer pattern in the hypermedia domain when

- you need to maintain the history of navigation;

- you want that not only nodes but also links or access structures could be recorded in the history;

- you need different viewers for the history;

- you need to support backtracking in the path made while navigating.

In a general context, the Navigation Observer pattern may be applied when you need to register the occurrence of some event in a 'Log', and you also want to configure the Log with one of different viewers.

4.4 Structure

In Figure 3, we show the structure of the Navigation Observer pattern using OMT notation [Rumbaugh91].

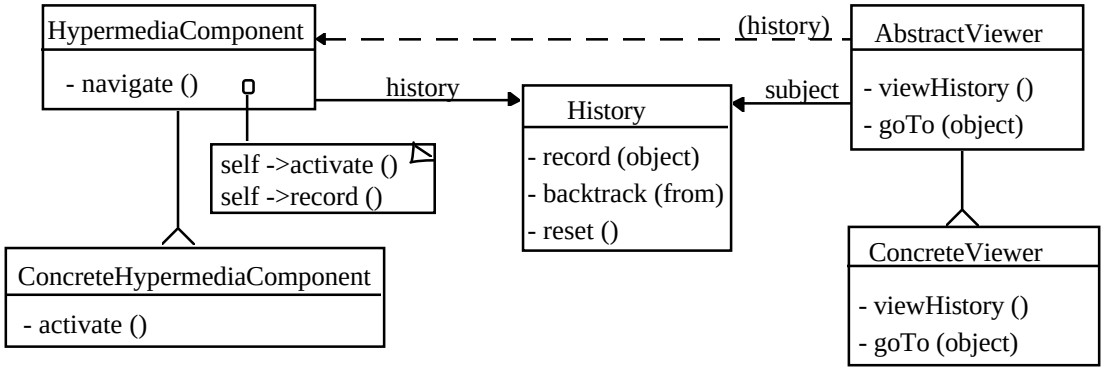

Figure 3: Navigation Observer pattern

4.5 Participants

- **HypermediaComponent**

  - Uses a Template Method such that each time an instance of a sub-class is visited, it will notify the history object.

- **ConcreteHypermediaComponent (Node, Link, Index)**

  - Implements the activate method.

- **History**

  - Records the history of navigation.

  - Is configured with the ConcreteHypermediaComponent(s) to be record, or with predicates to be tested against.

  - Implements the *backtrack* method.

- **AbstractViewer**

  - Defines an abstract interface for clients that need to display the history of navigation and perform navigational operations such as *goTo* (a hypermedia component in the history). This is shown with a dashed line in Figure 3, meaning that the viewer can activate the hypermedia component getting it from the history.

- **ConcreteViewer (List, Map, Graph)**

  - Implements the *viewHistory* and *goTo* methods.

4.6 Collaborations

- Each time a hypermedia component is accessed, it sends the message *record* to the *history*, sending itself as the argument, and the history registers it when corresponding.

- When the user wants a perceivable view of navigation, the message *viewHistory* is sent to an instance of the corresponding *ConcreteViewer*, which in turn makes the current history perceivable.

- *Concrete viewers* interact with hypermedia components for performing *goTo* operations.
- The message *reset* is sent to the history each time the user wishes to "reset" the navigation history.

4.7 Consequences

Navigation Observer offers the following benefits:

- It decouples navigation from its history and the history from a particular way of displaying it.

- This decoupling allows the separation of the application-specific styles of viewing the history from the application-independent functionality (that of activating nodes and traversing links).
- Some overhead could arise when the viewer is only interested in certain type of nodes (e.g., a map of visited cities in a hypermedia that also includes tourist attractions such as monuments, or museums). The viewer will have to interpret the history to filter out the cities, or the type of nodes in which it is interested.

4.8 Implementation

Several issues related with the implementation of Navigation Observer are discussed below.

1- *History configuration*. The history object may be configured with a predicate to be tested against hypermedia components, if it is found necessary to filter some of them. However, if the predicate turns too complicated, a subclassification of the History class may be more appropriate.

2-*Different algorithms for defining the history*. Though histories may be regarded as simple stacks recording each visit to a hypermedia component, sometimes it could be necessary to provide viewers with some "condensed" history. This may happen either when the same object is visited more than once or when complex backtracking in a multiple windows environment occurs. One possible approach is to define a different method in History that returns the record of navigation discarding duplicates and cycles. Another approach is to add this responsibility to viewer objects that may analyze the stack when needed.

3-*Mapping viewers to a history*. This issue is similar to the one in the Observer pattern [Gamma94], as viewers observe the history. It could be implemented either by using an associative look-up to maintain the history-to-viewer mapping so as not to incur storage overhead but increasing the cost of accessing viewers, or by storing references to viewers in each history object, what may be too expensive when there are many histories and few observers.

4-*When to update viewers*. Viewers may request information from a history each time they need to be displayed. Also one may want to update the view each time a new hypermedia component is visited. These two options may co-exist as discussed in [Gamma94].

5-*Having two or more histories*. If multiple users may access the hypermedia application concurrently, different history objects are needed to record the different navigation sessions (this is common in some hypermedia environments like Trellis [Stotts89]). This possibility complicates the implementation because the same hypermedia component may be visited in different contexts. In such case hypermedia components must be aware of the "current" history object or we may use a Mediator among hypermedia components and the history.

4.9 Known uses

Different hypermedia products provide a way of visualizing navigation history. For example Microsoft's Windows Help shows the history of a help session as a list of visited topics. Similarly, in some World Wide Web viewers like Netscape it is also possible to select a location from the navigation history. Though in Netscape only a text-based list is presented, the underlying structure of the application (running as a client of the WWW server) allows using Navigation Observer for building new types of viewers as those in [DasNeves94], [Wood94], [Chi94].

Another example not related with hypermedia applications arises in object-oriented environments providing ways to access and manipulate the execution history. In Smalltalk for example, different types of debuggers (Concrete Viewers) can be implemented by accessing the execution stack (History). In [Alvarez95] it is discussed how to build animations showing the way objects interact with each other; in this example the history "filters" objects and methods accordingly to a user choice, and those objects are later animated. Decoupling objects, histories and viewers helped in achieving a more flexible and extensible architecture for building animations.

4.10 Related Patterns

The relationship among viewers and history resembles the Observer pattern. In turn History may be implemented as a Singleton, or Mediator may be used to decouple hypermedia components from a particular history when dealing with multiple browsing sessions.

5. Concluding Remarks

This paper analyzed the role of design patterns in the process of building object-oriented hypermedia applications, i.e., applications in which we can navigate through objects' views by following links that mimic relationships among objects. We have presented two new design patterns, Navigation Observer and Navigation Strategy, that solve recurrent design problems in hypermedia applications. Other similar patterns used for structuring, controlling, or monitoring navigation must be discovered once we gain understanding about the kind of navigation structures we can find in hypermedia applications. Designing a Pattern Language for building these kind of applications will then be a feasible and fruitful approach for simplifying the construction task in this domain.

6 Acknowledgments

We would like to thank John Vlissides for shepherding our work through the approval process, helping us improve our paper with his careful review. The writer's workshop participants also made many useful comments.

Hypercard* is a trademark of Apple Inc.

7 References

[Alvarez95] X. Alvarez, G. Dombiak, A. Garrido, M. Prieto and G. Rossi. "Objects on Stage. Animating and Visualizing Object-Oriented Architectures in CASE Environments" Proceedings of the International Workshop on Next Generation CASE Tools. CAiSE'95, Finland, June 1995.

[Balasubramanian94] V. Balasubramanian and M. Turoff. “Incorporating Hypertext Functionality into Software Systems”. Workshop on Incorporating Hypertext Functionality into Software Systems. ACM, ECHT'94, Edinburgh, September 1994.

[Bigelow88] J. Bigelow. "CASE and Hypertext". IEEE Software, March 1988.

[Carvalho92] S. Carvalho : "Tool. The Language". Technical Report, PUC-Rio, Brazil, 1992.

[Chi94] Ed. H. Chi. "Webspace Visualization". Available on WWW: http://www.geom.umn.edu/docs/ weboogl/ weboogl.html

[Das Neves94] F. Das Neves. "Pictures from an Exhibition: A Spatial Metaphor for the concrete Narrative in Hypermedia". Workshop on Spatial Metaphors in Information Systems", ECHT'94, Edinburgh, September 1994.

[Davis92] H. Davis, W. Hall, I. Heath, G. Hill & R. Wilkins. "Towards an Integrated Environment with Open Hypermedia Systems". Proceedings of the ACM Conference on Hypertext, ECHT' 92. Milan, Italy, December 1992.

[Gamma94] E. Gamma, R. Helm, R. Johnson and J. Vlissides. "Design Patterns: Elements of Reusable Object-Oriented Software", Addison Wesley, 1994.

[Grønbæk94] K.Grønbæk. "Composites in a Dexter-Based Hypermedia Framework". ECHT'94 Proceedings.

[Halasz94] F. Halasz and M. Schwartz. “The Dexter Hypertext Reference Model”. Communications of the ACM, 37(2), 1994.

[Hill95] G. Hill, W. Hall, D. De Roure, L. Carr. “Applying Open Hypertext Principles to the WWW”. Proceedings of the International Workshop on Hypermedia Design, Montpellier, France, June 1995. Springer Verlag, forthcoming.

[IEEE92] IEEE Software. "Special issue on Integrated CASE", March 1992.

[Krasner88] G. Krasner and S. Pope. "A Cookbook for Using the MVC interface paradigm in Smalltalk-80", Journal of Object-Oriented Programming, 1(3), 1988.

[Leonardi94] C. Leonardi, M. Prieto, G. Rossi and R. Gonzalez Maciel. "Microworlds: A Tool for learning Object Oriented Modeling and Problem solving" , Proceedings of the Educator's Symposium. ACM Conference on Object Oriented Programming Systems Languages and Applications. OOPSLA'94.

[Oinas-Kukkonen94] H. Oinas-Kukkonen. "Hypertext Functionality Approach Defined". Workshop on Incorporating Hypertext Functionality into Software Systems. ACM, ECHT'94, Edinburgh, September 1994.

[Rossi94] G. Rossi, A. Garrido, A. Amandi. "Extending object-oriented applications with hypermedia functionality". Workshop on Incorporating Hypertext Functionality into Software Systems. ACM, ECHT'94, Edinburgh, September 1994.

[Rumbaugh91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen: "Object-Oriented Modeling and Design". Englewood Cliffs, NJ: Prentice Hall, 1991.

[Schmidt95] D. C. Schmidt: “Acceptor and Connector: Design Patterns for Active and Passive Establishment of Network Connections”, in Workshop on Pattern Languages of Object-Oriented Programs at ECCOP’95 (Aarthus, Denmark), August 1995.

[Schwabe95a] D. Schwabe and G. Rossi. "Building Hypermedia applications as navigational views of an information base", Proceedings of the IEEE Hawaii International Conference on System Science 1995.

[Schwabe95b] D. Schwabe and G. Rossi. "The Object-Oriented Hypermedia Design Method", Communications of the ACM, August, 1995.

[Stotts89] P.D. Stotts and R. Furuta: "Petri-Net based hypertext: Document structure with browsing semantics", ACM Transactions on Information Systems, 7(1):3-29, 1989.

[Wood94] A. Wood, N. Drew, R. Beale, R. Hendley. "HyperSpace: Web Browsing with Visualisation". Available on WWW: http://www.cs.bham.ac.uk/~nsd/research.html

Gustavo Rossi and Alejandra Garrido can be reached at LIFIA: Laboratorio de Investigación y Formación en Informática Avanzada. Dpto. de Informática, Fac. de Cs. Exactas, Universidad Nacional de La Plata.
C.C. 11, (1900) La Plata, Buenos Aires, Argentina. Fax/Tel: +54-21-22 8252. E-mail: [grossi, garrido]@ada.info.unlp.edu.ar

Sergio Carvalho can be reached at Departamento de Informatica. Pontificia Universidade Catolica. 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: sergio@inf.puc-rio.br

Gustavo Rossi can be also reached at Dto. Informatica, PUC-Rio, Brazil, and CONICET-Argentina. E-mail: rossi@inf.puc-rio.br