Navigating between objects.
Lessons from an object-oriented framework perspective

Gustavo Rossi[1] ,Alejandra Garrido[2], Daniel Schwabe (*)
LIFIA-Facultad de Ciencias Exactas. UNLP. Argentina
E-mail: [gustavo, garrido]@sol.info.unlp.edu.ar
(*) Dto de Informatica. PUC-Rio, Brazil
E-mail: schwabe@inf.puc-rio.br

Abstract

The main goal of this paper is to present a general architecture for building computational hypermedia applications, i.e. those applications that combine the hypermedia navigational style with other kinds of computations in an object-oriented system. We first motivate our work discussing why these kind of applications need special attention. Then, we briefly present the architecture and components of an object-oriented framework that allows extending object-oriented applications with hypermedia features. Finally, and as the main contribution of this paper, we discuss the most important design decisions behind the framework, presenting them as a set of micro-architectural constructs that yield a general architecture for integrating object-oriented and hypermedia applications.

1-Motivation

The emergence of the World Wide Web has raised a new generation of computing applications: those combining hypermedia navigation through an heterogeneous and distributed information space with operations that query or modify such information.
Enhancing object-oriented applications with hypermedia technology should give the user two different, though seamlessly integrated, visions of the information universe; by using hypermedia, software designers can provide navigational access to an information domain by letting users browse through that domain using the “point and click” metaphor. However, while the object-oriented paradigm focuses on behavioral collaborations among objects, hypermedia is the science and practice of relationships encouraging the exploration of (semantic) relationships among information items. Moreover, applications providing features such as forward and backward navigation, history maintenance, annotations, etc. pose many new and different problems to designers as repeatedly reported in the literature [Nielsen95]; clearly, when designing this new type of applications we must decouple objects’ behavior from navigational operations.
Imagine, for example, a software engineering environment in which the user not only is able to create new design documents, check them for consistency, and generate code in a programming language, but he can also navigate those objects, add some comments to his documents, group design artifacts in navigable sets, provide guided tours to managers showing the relationships among use cases and business rules, etc. Building this kind of

[1] This work was also funded by CONICET and UNLM, Argentina
[2] Also in the Computer Science Department, UIUC, USA

applications (from now on Object-Hypermedia Applications) involve many interesting design decisions, even if we do not deal with distributed objects in the Web. In this paper we report some lessons learned while designing and implementing an object-oriented framework for building Object-Hypermedia applications.

2-A Framework for Object-Hypermedia Applications

Our approach is focused on enhancing object-oriented information systems, improving the access to their information resources by using a navigational front-end. A layered architecture is a straightforward solution; the architecture contains three main layers: the Object layer, the Hypermedia view and the Interface. Hypermedia components such as nodes and links, and hypermedia functionality (navigation, annotations, indexes, guided tours, etc.) are thus interleaved between the application objects and the user interface; the resulting interface is then enriched with multimedia and anchoring facilities to provide access to the hypermedia features (See Figure 1).

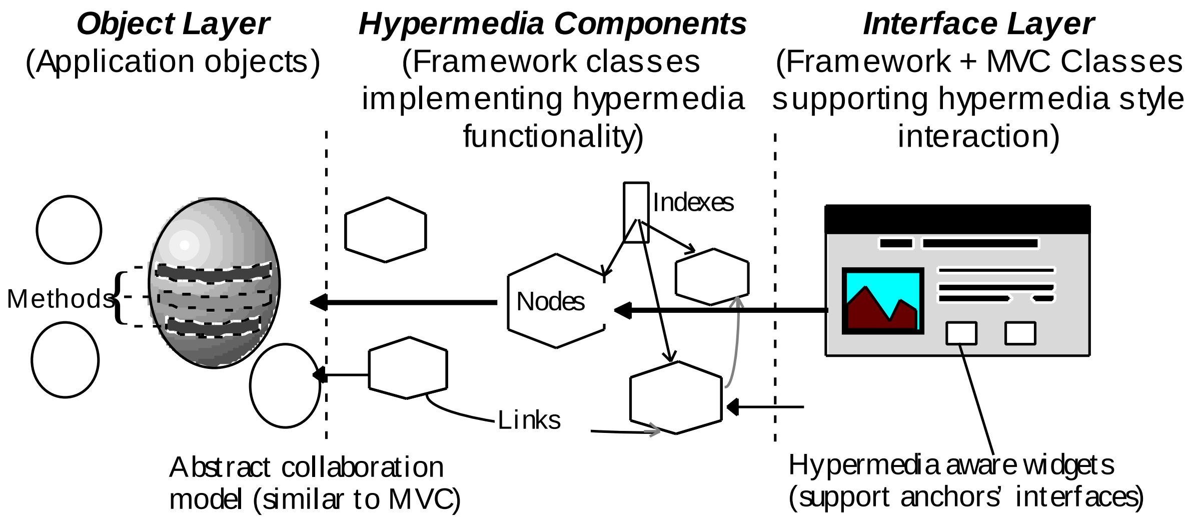
Object Layer
(Application objects)
Hypermedia Components
(Framework classes implementing hypermedia functionality)
Interface Layer
(Framework + MVC Classes supporting hypermedia style interaction)
Indexes
Methods
Nodes
Links
Abstract collaboration model (similar to MVC)
Hypermedia aware widgets (support anchors' interfaces)

Figure 1: The layered architecture implemented by OONavigator

An object-oriented framework (OONavigator) supporting the construction of applications with this architecture has been implemented [Garrido96] on top of the VisualWorks programming environment. It allows the designer to plug hypermedia nodes and links to application objects and to define different objects' views according to the user's profile; it helps in the creation of indexes and guided tours and in the implementation of a user interface providing anchors to links. OONavigator comprises a set of class hierarchies and a collaboration model among interface objects, hypermedia components and the application objects, following closely the philosophy of the Observer design pattern [Gamma 95]. Each object may have zero, one or more associated nodes acting as navigational views (observers) of that object. Views allow filtering objects' features and relationships for the corresponding

user’s profile or task. For each meaningful relationship in the object model, one can instantiate links connecting associated nodes. Links provide navigation paths among related application objects that can be traversed independently of the application’s behavior. Finally, interface objects are defined by using an extension of the MVC-based, VisualWorks interface framework that adds anchoring facilities such as sensible areas and hot-words. The framework has been used as the support environment for modern hypermedia and Web Information System design methods like OOHDM [Schwabe96] because it clearly separates conceptual from navigational and user interface design as proposed in those methods.
OONavigator has proved to be a solid framework; a set of toolkits have been implemented to allow visual plugging of nodes and links to objects, to simplify user interface definition and to adapt the interface to Web viewers [Maciel97]
However, the evolution of OONavigator from a white to a black box framework produced the most interesting result: a set of micro-architectural constructs that can be used to describe a reference architecture for applications combining navigation with most conventional computations. Many of these constructs resulted from applying well-known design patterns like Observer or Strategy [Gamma 95] or by extending the scope and applicability of those patterns to this particular domain. As a consequence, designers could reuse these solutions in different contexts; new applications could thus be implemented using even commercial tools for the Web, without using OONavigator but just applying the solutions expressed by the patterns on top of which the framework has been built.
We next summarize some architectural design decisions we took while designing the framework; they do not only show a sketch of the framework architecture but express more general design knowledge about this domain obtained from the evolution of the framework. A complete description can be found in [Garrido97].

3-From Patterns to a Reference Arquitecture

It is interesting to analyze the way in which the micro-architectural styles described below generate the architecture of an Object-Hypermedia application. While 3.1, 3.2 and 3.3 show two strategies for decoupling the object level from hypermedia components and thus influence the overall application’s architecture, 3.4 presents a design decision that deals with the organization of hypermedia components. Each design construct is described below using a simple template containing the problem, its solution and the pattern applied or adapted

3.1 Nodes as Objects Views

Problem: How to add navigation capabilities to the components of an object-oriented application? How to integrate objects behavior with navigation through related objects?

Solution: We have defined a navigational layer between the application to be enhanced and its graphical interface, built up of objects’ observers that are called *nodes*. While objects implement the application behavior, we implemented most navigational operations in nodes. Nodes in the framework “view” one or more objects in the application; they provide anchors for links (see 3.2) and forward all behavioral requests to observed objects. We have defined a class hierarchy of nodes, including simple and composite nodes and other kind of hypermedia components like indexes. This solution shows an interesting implementation of the Observer design pattern; nodes are not just object viewers because they implement additional (and independent) functionality. They are connected with other nodes by links, allowing navigation to proceed without the intervention of nodes' subjects (application objects). Besides, as discussed in 3.4, nodes are also typed.

3.2 Links as Relationships Views

Problem: How to map relationships among objects to links between nodes?

Solution: Links have a critical role in hypermedia applications. They have been thought as first-class citizens as well as nodes; each link represents a relationship of interest (in terms of navigation) between two or more objects, which must also have nodes as navigational views over them (see 3.1). Links are responsible for the navigation process described above. They are activated from anchors in nodes and their standard behavior is to open the link's target node. During this process, the destination of a link may be fixed or computed on demand by querying the objects involved in the relationship, and obtaining the node that "views" the target object. The different algorithms for computing the end-point of a link have been designed in a separate hierarchy, following the Strategy design pattern [Gamma95].

3.3 Wrapper Node

Problem: How can we add navigational behavior to existing GUIs?

Solution: Many times we deal with existing applications in which it is not possible or reasonable to re-design the application's structure following 3.1; this is usually the case with tools like class or file browsers, graphic editors, etc. In this situations we decorate the interface with a node wrapper. The node wrapper will capture those interface actions dealing with navigation and activate it, passing to the decorated interface all other actions. The wrapper node has an associated group of widget's wrappers for each widget in the decorated interface. Those widget's wrappers (for example text fields, images, etc.) must be hypermedia-aware, that is, it should be possible to define and activate anchors for links within their data. This design solution is subtly different from the one in 3.1. In this case, we have used a slight variant of the Decorator design pattern [Gamma95].

3.4 Node Class, Link Class

Problem: How can node (and link) types be defined, according to the classes of an object-oriented application that is extended with hypermedia functionality? Suppose we have application classes such as *Course* and *Professor*. We will surely need node types *Course* and *Professor*. How can we avoid this kind of duplication?

Solution: We defined a class NodeClass whose instances act as templates (types) for Nodes. Each instance of NodeClass represent a particular type of node and is related with corresponding instances of Node. Using this solution we decouple Node from NodeClass; Node contains basic hypermedia functionality (reacting to anchor activation for example) while instances of NodeClass act as the template for data and behavior that is common for all nodes of the same type (e.g. *Professors*). We also made instances of NodeClass responsible for the creation of nodes. In the same way, we defined LinkClass as a class whose instances act as Type-objects for instances of Link. This solution is an example of the use of the Type Object design pattern [Johnson 97].

4- Conclusions

Object-oriented frameworks represent the state of the art solution for implementing reusable software designs; however, one major impact of their implementation is the definition of micro-architectures addressing sub-problems within the framework's domain. In this paper we have shown some micro-architectural design constructs we used while implementing an object-oriented framework for hypermedia. Though described in the context of OONavigator, they represent the core of a more general architecture for building Object-Hypermedia applications, i.e. those that combine navigation with other computational behaviors. Ideas such as decoupling nodes from their behavioral counterparts (see 3.1) can be used successfully either in non "pure" object-oriented settings such as the WWW and implemented using conventional tools like HTML editors.

References

[Johnson97] R. Johnson and B. Woolf: "The Type-Object design pattern". In Pattern Languages of Program Design III,

[Gamma95] Gamma, E., Helm, R., Johnson R, and Vlissides, J. : "Design Patterns: Elements of reusable object-oriented software", Addison Wesley, 1995

[Garrido96] Garrido, A. , Rossi, G. : "A framework for extending object-oriented applications with hypermedia functionality". The New Review of Hypermedia and Multimedia, Taylor Graham, 1996, 25-42.

[Garrido97] A. Garrido. G Rossi and D. Schwabe: "Pattern Systems for Hypermedia". Preliminary Proceedings of PLoP'97, Allerton, USA, September 1997. Also in http://st-www.cs.uiuc.edu/~hanmer/PLoP-97/Proceedings/garrido.pdf.

[Maciel 97] R. Gonzalez Maciel: ""Integrating OO-Navigator with the Web - Visual and linking aspects". Proceedings of the V International Workshop on the Hypertext Functionality Approach, ICSE'98, KYOTO, April, 1998. Also in: http://www-lifia.info.unlp.edu.ar/~ramiro/htf5.

[Nielsen95]J. Nielsen: "Multimedia and Hypertext. The Internet and Beyond". Academic Press, 1995.

[Schwabe96] Schwabe, G. Rossi and S. Barbosa: "Systematic Hypermedia Design with OOHDM". Proceedings of the ACM International Conference on Hypertext

(Hypertext'96), Washington, March 1996. 116-128.