# A FRAMEWORK FOR EXTENDING OBJECT-ORIENTED APPLICATIONS WITH HYPERMEDIA FUNCTIONALITY

ALEJANDRA GARRIDO AND GUSTAVO ROSSI (*)

*LIFIA. Laboratorio de Investigación y Formación en Informática Avanzada.*
*Dto. de Informática, Fac. Cs. Exactas, Universidad Nacional de La Plata.*
*C.C. 11, (1900) La Plata, Buenos Aires, Argentina.*
*E-mail: [garrido, grossi]@info.unlp.edu.ar*
*(*) also CONICET and Dto. Informatica, PUC-Rio, Brazil.*

ABSTRACT

A core set of navigational aspects may be found by extracting the key features of hypermedia applications. Incorporated to an information system (IS), they may increase its utility and usability (1). This type of extension of an IS is called 'The Hypertext Functionality Approach' (2), and it may be done in different ways. In this paper we present a novel approach to add navigational features into object-oriented (OO) applications: by using the application model as the basis for the hypermedia model and placing the navigational features in a different layer, allowing to augment the application functionality without polluting the base model. The navigational features were included as components of an OO framework, and were defined by abstracting the major concepts of current hypermedia design models. The major goal of this approach is the seamless integration of the application's behaviour with main characteristics of hypermedia. This results in a hybrid application that may be considered either as an OO application enhanced with hypermedia functionality (HF) or, conversely, as a hypermedia application defined with the semantics of an OO model and enhanced with other than navigational computations.

## 1. INTRODUCTION

Adding hypermedia support functionality to an application may be addressed at two different levels, as defined by Oinas-Kukkonen (2): at the *information system (IS) level*, incorporating navigational features within the application environment, or at the *operating system level*, through a linking ability among different applications of an open environment (3, 4). This work is focused on the first approach, that is, enriching an IS by improving access to its information resources. For example, integrating hypertext in an intelligent tutoring system may provide the learner and educator with an additional communication tool besides allowing a more flexible organization for the concepts to be learned. In fact, current learning tools are usually hypermedia-based (5). However, when the application itself has not been conceived as a hypermedia application, as it is the case with most information systems (ISs), adding hypermedia support may be extremely complex or may be reduced only to trivial aspects (like interactive help systems, which let the user explore an information base, built as a stand-alone hypermedia application). On the other hand, merging the interface and navigational aspects of hypermedia (as it is usually the case in commercial packages) prevents building clean extensions either to the interface or the hypermedia model. Since hypermedia has become a well-known and mature software domain, it is possible to build good abstractions of its underlying concepts; therefore, they can be reused in different application areas, separating those abstractions from the application to be extended and from the interface level.

Object-oriented (OO) design has proved to be the best way of achieving *reusability*, because it allows defining by abstraction and composition the different components of an application, that are plugged in to fulfil the expected system behaviour. The application data is distributed among the components that become responsible of maintain it and achieve its assigned behaviour. This organisation of data resembles the non-linear organisation of hypermedia applications, in which chunked pieces of data are related in a graph-based way. Consequently, we claim that *OO applications may be easily extended with hypermedia functionality* (HF), by just extending their own model.

The state of the art solution for building complex and reusable architectures separating functionality in the OO field, is the construction of *frameworks* that model a specific domain. Following this idea we have constructed an OO framework for hypermedia applications that allows different instantiations in the hypermedia domain. An instantiation may be either an extension of some existent OO application, or may be just a hypermedia application constructed from scratch.

The framework allows extending applications with HF through a direct mapping of objects to nodes and links, thus viewing objects as nodes and accessing related ones by link traversal. Moreover, the framework allows preserving the behavioural semantics of the application, as specified during design, by providing communication paths for notifying objects about the occurrence of interface events. This does not imply that the application should be notified of the presence of link traversal operations. In contrast, the application remains unaware of navigation, allowing making reusable, maintainable and less complex extensions.

In order to design the framework's components over the basis of a *unified core model*, many hypermedia design methodologies have been surveyed and studied, as (6, 7, 8, 9, 10, 11). These design models, however, do not deal with the incorporation of behaviour inside nodes. Hence, we have enhanced the resulting model conceiving each node as an active agent that may dispatch certain computations by message passing with the related objects.

In the next sections we discuss the rationale for building an OO framework, and the entire design is presented with an example; we further analyse the different approaches for instantiating the framework and finally discuss some application areas, related work and concluding remarks.

## 2. AN OBJECT-ORIENTED FRAMEWORK FOR HYPERMEDIA. ISSUES.

A framework in the OO field is an abstract design of a particular type of application or domain (12). It consists of a set of abstract and concrete classes and the abstract specification of the collaborations between objects (13). Building OO frameworks is difficult because they must represent a concrete domain in an abstract way, i.e., allowing different instantiations of the entire framework while being flexible enough to be customised to a particular application. This requires a deep knowledge both of the target domain and the profile of applications that may use the framework. Its design implies not only describing a set of classes but also the thread of control that it must dictate over the domain, and defining a model of the relationships between the framework and the outside components. The benefits of using frameworks are considerable: faster application building and easier maintenance, because all applications using a particular framework will exhibit a common design structure.

One of the most widely known framework is the Model-View-Controller (MVC) (14), which allows implementing sophisticated graphical user interfaces by separating the application (Model), its views (View) and the interaction with the outside world (Controller). The MVC metaphor is prevalent in many OO languages and architectures. Its implementation in Smalltalk-80 allows defining new kinds of Views and Controllers thus easing the transition from ‘conventional’ interfaces, to others using different type of media for output and complex devices (data gloves, pens, etc.) for input.

Following a similar strategy, we have designed and implemented a *framework in the hypermedia domain* for enhancing OO applications with HF, and the other way around, for creating hypermedia applications enhanced with OO concepts. By using this framework, an application is benefited with the possibilities of:

- navigate among related items of interest,
- define different context for different users' profiles, providing each user with the information that suits their needs,
- enhance the graphical user interface with the possibility to define anchors for links over any type of data,

- enable rapid and easy access to the desired information by way of indexes and backtracking,
- guide in the search for information that fits the user profile with guided tours,
- add annotations and bookmarks,
- provide history lists.

Conversely, if we consider the hypermedia application point-of-view, it gains:

- a semantically rich definition of links,
- reuse of hypermedia concepts,
- the addition of computations, that related with the information inside nodes, as much as the one related with dynamic linking,
- the possibility to define classes of nodes and links, and automatically create all their instances.

Although navigational features may result helpfully, not all ISs will allow a full integration of hypermedia and application functionalities. We have identified three different subsets of applications in general, according to the *different needs of hypermedia support functionality* that may result *feasible* and *meaningful* for them. These subsets are identified as:

a) *Data-based applications:* these are passive applications, where hypermedia features are easily introduced by directly mapping each concept to nodes and its relations to links. In the OO field, this is translated to mapping classes to node classes and relations between them to link classes, allowing a meaningful definition of hypermedia components and the complete automatization of their instantiation. A full set of hypermedia features may be added to this type of static applications in which no conflicts can arise with the proper application functionality.

b) *Hybrid applications*: they have a behavioural aspect associated with each information item of their components. In this case, each node may be considered as an active agent that allows the dispatching of actions related with the data (or the class of object) that is currently mapping. All or most of the navigational features can be added to them, though the dynamism of the application must be taken into account. Our work is mostly oriented to this type of applications because they are predominant in the OO field. We will give a full example below.

c) *Behaviour-based applications:* these are applications with a browser-based interface for controlling the dispatching of the proper application functionality. Consider as an example an airline information system, with an interface where each sub-window allows some kind of "filtering" over the set of flights (such as airline, origin, destination, cost, etc.). Navigation through the application data seems marginal because the application is focused on its computations that make it highly dynamic, as opposed with the passiveness of hypermedia. Nevertheless, the framework allows the designer to preserve the "control tools" that are important for its transaction's management and to which the user is already accustom, and offers a navigational facility to the computational results or to each involved entity, plus the possibility for annotations.

## 3. LIBRARY SYSTEM: A HYBRID APPLICATION EXAMPLE.

Consider that we have a library system as our OO application. A simplified sketch of the model in the Object Modeling Technique (OMT) notation (15) is shown in figure 1. In that model, we have the following classes: Library, Book, Chapter (as parts of books), Author, Reader, Loan, Special-loan (as a subclass of Loan) and Reservation. For simplicity, the figure only shows the information assigned to each class, although they also have an associated behavioural aspect. There are other interface classes, such as LoanTool and ReservationTool.

**Fig. 1***: Model for the library application*

This is a hybrid application because, besides its valuable data base, it has an important behavioural aspect related with the loan and reservation of books. The possibility to navigate between the books and their authors, among chapters and referenced books, or among readers and their borrows or reservations, would be very helpful. We can think also in the addition of multimedia information, like author photographs, library maps displaying book's location, etc. Annotation facilities and access structures are evident great advantages. In addition, it would be useful to have a different presentation and manage of system data for distinct roles of users (like library director, librarians, and readers), in order to support their different understanding of the domain. We will further analyse the hypermedia extension that we propose over this application as we explain each framework concept.

## 4. FRAMEWORK SPECIFICATION

As we mention above, merging the interface and navigational aspects of hypermedia within the application could derive in obscure code, preventing to build maintainable extensions either to the application, the hypermedia model, or the interface. Accordingly, three levels of abstraction were defined:

1) *Object level*: the application level with its data and behaviour model;
2) *Hypermedia level*: where the framework components were defined;
3) *Visual level*: where the interface aspects are managed.

We have also defined a model for the communication between the first and second levels, and between the second and third one. In OOHDM methodology (6), a similar layered architecture is defined for hypermedia applications, despite it does not address the HF approach.

We will explain each level below, and next extend the second one, where we will outline the framework design.

### *Object Level*

This level conforms the underlying model for the resulting application. It consists of the classes of the application domain that will provide the data to be shown, the relations among data, and the behaviour that will be extracted by the hypermedia level.

If a semantically rich model is used, i.e., a model with explicit definition of relationships like that proposed by the OMT (15), links of the hypermedia level will be easily obtain by reflecting relationships among objects.

This level may be compared with the *model* of the MVC framework, or related to the 'Conceptual Model' of the OOHDM methodology.

### *Hypermedia level*

In this level we have defined the hypermedia framework's components, which the designer will be able to instantiate in order to define a navigational view over the first level. It may be related to the 'Navigational Design' of OOHDM.

The framework itself comprises a set of abstract and concrete classes and their behaviour specified as methods in each class (and corresponding sub-classes). We will describe the classes in section 4.1, relating them with the concepts defined in current hypermedia design methods.

*Visual level*

In the visual level, another interface tool or framework is used to specify nodes' appearance (several graphical representations may be defined for the same node). It may be compared with the 'Abstract Interface Design' of OOHDM. Our current implementation uses the MVC framework that has been extended for dealing with link anchors over any type of data. Though the framework can be used with other interface metaphors requiring minimal re-coding, using the MVC has several advantages: first, as said before, its use is prevalent in OO systems; besides, the interface 'look and feel' is well suited for building hypermedia applications. We will not discuss this level in detail; see for instance (14).

## 4.1. HYPERMEDIA LEVEL ARCHITECTURE

*Hypermedia, Nodes and Links*

In the hypermedia level, an instance of the class *Hypermedia* is created for a particular visualization of the application. Inside it, nodes are defined as objects' views and links stand for associative objects or simple relationships. This means that nodes and links are empty templates that only know how to be navigated, but that require to be "plugged" with application objects in order to be instantiated. Every node derived from the object model, instance of the class *ObjectNode*, is said to "depend on" or "observe" the correspondent object to which it is plugged. This is achieved by associating every aspect of the object that returns data with a node "slot", so the node will use that aspect as messages to the object whenever it needs to get or set the information. In order to reflect the object actions, the interface events that the node does not understand as link activations are directly delegated to the object. In this way we can preserve the original application semantics. In the Library example we will have an 'objectNode' for the Library instance, one for each Reader and one for each Author.

A node is not restricted to map only one object; it can be a composition of related objects, as would be preferred when one or more objects depend on another, and they are not relevant enough to be isolated in a node. Such would be the case if the library address were a complex object with some attributes (like street, number, post office, city), but you would like to see this data together with the other information of the Library. You must then map a library object together with his correspondent address object in the same node. In this sense, OOHDM apply the same approach when translating from 'Conceptual Model' to 'Navigational Design'.

The nodes described above are called *AtomicNodes*, i.e., those that contain a set of basic or multimedia data, a set of methods (both extracted from the associated object(s)), and a set of anchors. Furthermore, the framework gives the possibility to map a complex object, with its 'part-of' relationships, to a complex node, as complex entities of HDM (8). These aggregations are called *CompositeNodes*, and represent structures that capture hierarchy, nonlink-based organisations of information, extracted from the underlying application. Composite nodes provide one of the possible access-structure described below for accessing their components. Composites are further specialised in *DependentCompositeNodes* which existence depends on that of the parent node. (See figure 2 for the resulting node hierarchy). Returning to the example, the objects of class Book will be mapped to composite-nodes, which will display the data of a particular book, and an index-structure to reference its chapters.

Very close to our idea of composites is the idea of aggregation of nodes, that Dexter (7) and DHM (16) models have proposed, and a lot of systems have implemented (KMS, NoteCards, Intermedia). In spite that they are called "composites" also, they are different from ours, in that they not always show a hierarchical, part-of relation. Those other types of aggregation structures, that can be regarded as sets with a special navigational pattern (resembling more the idea of 'Collections' defined by Garzotto *et al.* (9)), are included in our model as instances of *CollectionNode*. Under this category we also have *VirtualNode* representing those nodes computed at run-time as the result of a query (in a similar way as in (16)) so we can make those results browsable and linkable.

The designer can also come out with nodes that are not object views; instead, they specifically belong to the hypermedia level, as could be the case of a presentation node. The framework supports them as instances of *HyperNode,* and its data is encapsulated in objects added to the application by the framework. Usually, they will display multimedia data (as the library maps or videos) because this type of information rather appeared at the object level.

Fig. 2: *Node hierarchy*.

For behaviour-based applications (type c), the designer is encouraged to maintain the interface tools and add navigation to them, if they permit a minimum of flexibility such as the selection of a piece of data. The metaphor is to introduce a special type of node that we called *Navigator*, which acts as a "transparent" over the node, with only anchors for links shown on demand when the user explicitly wants to navigate. The designer may define the anchors over items of a list (which will become an access structure of demand) or over a piece of text (*hotword*) or other kind of media. The user will be also able to create annotations dynamically, just selecting and defining the anchor to link it. This approach is similar to that introduced by Microcosm (17), with the difference that the links are not inter-application.

Our library application had two pre-defined interfaces for book loan and reservation. They are filling forms rather that browsers (as that of applications of type c), but we may use them under the underlying HF in the following way: once arrived by hypermedia access to the desired book node, the action of loan or reservation might be triggered through buttons or menu items, opening the respective form to be filled with the reader information.

The class *Link* represents the association between two or more nodes. Links are accessed through anchors contained in nodes or in access structures, and arrive at link-endpoints. A subclassification of the class *LinkEndpoint* allows single or multiple endpoints, and for the later, an intermediate Index allowing a particular target selection is provided. There are different ways provided for computing the endpoint, i.e. statically or dynamically, the later resolving the target when the link is activated, as exposed in (7). This leads to another hierarchy of algorithms that compute the link endpoint. (See Figure 3). For a detailed explanation of this strategy see (18).

Fig. 3: *Endpoint-Solver hierarchy*

Links may have different attributes (as OOHDM defines), like a meaning and a block of postconditions to be executed at link activation. Moreover, a link may derive from an associative object, and hence it will have a reference to that object, from which it can also extract its attributes and postconditions. The latter are called *ObjectLinks* (See figure 4 for the resulting hierarchy). We will later discuss the difference between *typed* and *untyped links*.

Fig. 4: *Link hierarchy*

*Node classes and Link classes*

Simulating the way classes factor out the structure and behaviour of its instances, a *NodeClass* defines the common characteristics of a set of 'ObjectNodes'. Furthermore, NodeClasses become "dependents" of application classes, and so they can observe the creation of each instance of the class. The concept of *NodeClass* can be related to Entity types of HDM, and mainly to node classes defined in OOHDM.

Furthermore, its implementation inside our framework allows the automatization of nodes' creation. A NodeClass automatically creates and associates each node with the objects of the related class(es). For this purpose it was necessary to make a subclassification of NodeClass parallel to the hierarchy of ObjectNode, with *AtomicNodeClass* and *CompositeNodeClass.* Bieber and Kacmar (19) have mentioned in their related work the weighty facility of automatic node and link creation.

Following the same idea, we defined the concept of *LinkClass* and *LinkDestinationClass*. An instance of the class LinkClass will define the common characteristics of a set of *TypedLinks*, such as the source nodeClass, the linkDestinationClass, its meaning and activation postconditions.

Haake and Streitz (20) have recognised the importance of typed nodes and links in order to communicate goals more effectively, by expressing more semantics. Nevertheless, they also considered the importance of flexibility in the organization of knowledge for readers. Agreeing with them, we support *UntypedLinks* (fig. 4) which are created by hand.

Using the library example, the hypermedia will have the following atomicNodeClasses: Library, Chapter, Author and Reader, and the compositeNodeClass: Book. LinkClasses will be: *Has-reader*, between the Library nodeClass and the Reader destinations; *Has-books*, between Library nodeClass and Book destinations; *Wrote*, between Book and Author nodeClasses; *References*, between Chapter and Book; *Loans* and *Reservations* between Reader and Book.

*Access structures*

Access structures are very useful in order to deal with the famous problem of information overhead that a hypermedia may produce. The framework implements three different types of them: Index, Guided-tour or

Iconic structure (which have iconic references to its components). Access structures are characterised by three attributes, in accordance with the model given in OOHDM: the set of target nodes, the set of selectors or target's data items to be displayed, and a logic predicate on target nodes that allows the definition of conditional access structures, as proposed in the RM methodology (10).

Indexes may be accessed each time we activate an anchor related to a multiple-destination link; CompositeNodes define an access structure for accessing its components; AtomicNodes may also have access structures inside them, in pursuit of a better organising of their links. Examples are the indexes that display the loans and reservations of a reader, allowing the navigation towards the involved books, or a guided tour to navigate through author's books from an 'Author node'.

### *Contexts, Views and Representations*

It is usually necessary to have different *contexts* in the hypermedia for distinct user roles or profiles, and therefore have different *node views* associated with each context. This provides different navigation alternatives (when different outcoming links have been defined in each context). The Nested Context Model (11) defines contexts as a way of grouping related nodes and as the basis for navigation. Our intention, however, was not to group nodes, but to provide different views of the same node. The idea is closer to 'Navigational Views' of OOHDM, but there, each possible user profile defines a different hypermedia application.

Graphically, we might clarify the idea thinking about a node as a prism where each face shows a particular node view (Figure 5).

The Library may have a context for the library director, another for librarians and one for readers.

Furthermore, many *representations* can be defined inside each node view, in order to change the appearance or the media of the same piece of information. It means that each node view defines the set of data to be displayed under a given context, and each representation inside the view will define how to display that data. In the example you could find useful to show the information describing the library as a text, or with the library images; you might read a book summary or just listen to it. The concept of Representation is the same as the idea of Perspective in HDM and OOHDM.

Fig. 5: *Node views and Representations*

## 5-USING AND EXTENDING THE FRAMEWORK

The essence of a framework is the model of interaction and control flow among its components (13). Ideally, a particular instantiation can be created entirely from classes in the framework. In practice, it could be necessary to derive new concrete subclasses of the abstract classes in the framework, in order to customise the application. This is one of the reasons for which the design of a framework is usually difficult: it must be *adaptable* and *extendible*.

Thus, instantiating the hypermedia framework implies deciding whether the concrete classes defined in it are adequate and complete. Examples of subclassification would be the addition of a new access structure, or a special kind of node, supporting for instance constraint checking.

The hypermedia framework also requires deciding which objects will be viewed as nodes, which conceptual relationships will be mapped to links and in which way nodes will be visualised, as explained above.

Building a navigational view of an OO information system can be done by coding or by using high-level tools as it is usual in interface design. Codifying the connections between the different system levels may be very extensive and cumbersome. Using high-level tools the connection between the object and hypermedia levels, as well as that between the hypermedia and interface levels, may be easily achieved. We have implemented a graphic tool as an extension of the VisualWorks$^{TM}$ canvas to simplify the application construction.

## 6-EXPERIENCE WITH THE FRAMEWORK

We are using the framework in two different areas: Software Engineering Environments and Information Systems. We have prototyped a CASE environment for an OO analysis and design method, OOSE (21), and have enriched it with HF using our framework (22). The hypermedia extension over this application was particularly interesting and fruitful. The CASE environment is highly dynamic, and the user is almost always creating anchors for links when the destination points are not even created, e.g., when defining a new document or component inside it. In this case we have used links with dynamic-computed destinations, created at link activation. Furthermore, we noted that hypermedia is the best way for achieving *traceability* along the entire software life cycle, because it can deal with all the information of document and program derivation. We also used hypermedia to capture the design rationale (23). In relation with this we found that there is a need to restrict the possible link types in order to improve navigation and data organization, and to structure the user mind map. Our future step will be to find the most convenient and meaningful restrictions over link types, and the way to improve consistency from the trace information that links may provide.

We have also re-engineered an information system for an academic department that provide functionality for registering students to courses, maintain the professor's information base, etc. In this case combining the "standard" behaviour with the navigational style of hypermedia has been straightforward.

While designing the framework we found usefully to discuss and communicate design alternatives in terms of *design patters* and their vocabulary (24). A design pattern systematically names, explains and evaluates an important and recurring design in OO systems. We are now working on the definition of a *pattern language* in the hypermedia domain (18).

## 7-RELATED WORK AND CONCLUDING REMARKS

Object-oriented frameworks are the state-of-the-art solution for reusing large scale abstract designs in a particular application domain. In some mature areas (like user interface design) they have been used for years. Using a framework allows a designer to separate his application from the framework's components, thus allowing easy *extension* and *maintenance*.

In this paper we have proposed using an OO framework for embedding hypermedia support features into an application. We provide a layered separation between the application and the hypermedia domains though preserving the application's behavioural semantics by allowing the definition of communication paths between nodes and links with application objects. We have made some experiences with the framework showing its feasibility either for building new OO applications or for re-engineering existing ones.

The key difference between our approach and other OO views of hypermedia applications building, like the ones proposed by Lange (25) or Marmann *et al.*(26), is that the hypermedia framework is only a component of a perhaps complex application that supposes the use of a collaboration model with the Object and Visual levels. In this way it is an excellent alternative for incorporating hypermedia features to traditional business applications.

Grønbæk and Trigg (27) and Arents and Bogaerts (28) proposed layered architectures similar to the one presented here, although they do not address the HF approach.

The nearest work is the one presented by Bieber and Kacmar (19), that also deals with HF added to computational applications, separating the computations from the hypermedia and interface modules. The differences arise from the fact that it is not for OO applications, so it lacks of hierarchies of abstract hypermedia classes and of the "dependency mechanism" that we apply between application and hypermedia components (available in OO environments). Instead, the work is based on rules for the definition of node and link types over an application data base.

We hope that the framework presented here can serve for identifying the *core model of HF features* in the OO field, and the extent to which each type of application allows the *integration of its functionality with the navigational one*. We have plenty of work for the near future: the definition of a pattern language validating the framework design, the improvement of the tool for instantiating the hypermedia components, the addition of intelligent browsers of nodes and links, and the incorporation of the framework in an open hypermedia environment.

## 8. ACKNOWLEDGEMENTS

The authors would like to thank Mr. Harri Oinas-Kukkonen for his valuable comments and suggestions.

## 9. REFERENCES

1. BALASUBRAMANIAN, V. *and* TUROFF, M. *Incorporating Hypertext Functionality into Software Systems*. New Jersey Institute of Technology: Workshop on Incorporating Hypertext Functionality into Software Systems I, 1994. (Institute for Integrated Systems Research Technical Report #95-10).
2. OINAS-KUKKONEN, H. Developing Hypermedia Systems - The Hypertext Functionality Approach. *In: Data Management Systems, Proceedings of the Basque International Workshop on Information Technology*, BIWIT'95 (San Sebastian, Spain, July 1995). Los Alamitos, North Carolina: IEEE Society Press, 1995, 2-8.
3. ARENTS, H. *and* BOGAERTS, W. *Link Services Badges: A Mechanism for Announcing Available Hypertext Functionality*. New Jersey Institute of Technology: Workshop on Incorporating Hypertext Functionality into Software Systems I, 1994. (Institute for Integrated Systems Research Technical Report #95-10).
4. ASHMAN, H. *and* VERBYLA, J. *Retrofitted Hypermedia for Third-Party Applications*. New Jersey Institute of Technology: Workshop on Incorporating Hypertext Functionality into Software Systems I, 1994. (Institute for Integrated Systems Research Technical Report #95-10).
5. AMANDI, A., ROSSI, G., PRIETO, M. *and* LEONARDI, C. Learning Object-oriented Concepts with Multimedia Technology. *In:* J. Archibald and M. Wilkes, *eds. Addendum to the Proceedings of the 8th ACM Conference on Object Oriented Programming*

*Systems Languages and Applications,* OOPSLA'93, (Washington, DC, USA, Sept. 1993). OOPS Messenger, 5 (2), April 1994, 13-16.

6. SCHWABE, D. *and* ROSSI, G. Building hypermedia applications as navigational views of information models. *In: Proceedings of the 28 th. Annual Hawaii International Conference on System Science*, (Maui, Hawaii, January 1995). Vol. III, 231-240.
7. HALASZ, F. *and* SCHWARTZ, M. The Dexter Hypertext Reference Model. *Communications ACM 37* (2), 1994, 30-39.
8. GARZOTTO, F., PAOLINI, P.*and* SCHWABE, D. HDM-A model-based approach to hypermedia application design. *ACM Trans. Info. Syst. 11* (1), 1993, 1-26.
9. GARZOTTO, F., MAINETTI, L. *and* PAOLINI, P. Adding Multimedia Collections to the Dexter Model. *In: Proceedings of the European Conference on Hypermedia Technology,* ECHT'94 (Edinburgh, Scotland, September 18-23). New York: ACM Press, 1994, 70-80.
10. BALASUBRAMANIAN, V., ISAKOWITZ, T. *and* STOHR, E. Designing Hypermedia Applications. *In:* R. H. Sprague *and* B. Shriver, *eds. Proceedings of the 27th. Hawaii International Conference on System Sciences*, HICSS'94 (Maui, Hawaii, January 1994). IEEE Computer Society Press, 354-365.
11. CASANOVA, M. *and* TUCHERMAN, L. The Nested Context Model for Hyperdocuments. *In: Hypertext'91 Proceedings* (San Antonio, Texas, Dec. 1991). New York: ACM Press, 1991, 193-201.
12. JOHNSON, R. *and* FOOTE, B. Designing Reusable Classes. *Journal of Object-Oriented Programming*, *1* (2) 1988, 22-35.
13. JOHNSON, R. *and* RUSSO, V. *Reusing Object-Oriented Design*. Champaing-Urbana, 1991 (University of Illinois Tech report UJUCDCS 91-1696).
14. KRASNER, G. *and* POPE, S. A cook-book for using the model-view-controller user interface paradigm in Smalltalk-80. *Journal of Object-Oriented Programming, 1* (3), 1988, 26-49.
15. RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F. *and* LORENSEN, W. *Object-Oriented Modeling and Design.* New York: Prentice Hall, 1991.
16. GRØNBÆK, K. Composites in a Dexter-Based Hypermedia Framework. *In: Proceedings of the European Conference on Hypermedia Technology,* ECHT'94 (Edinburgh, Scotland, September 18-23). New York: ACM Press, 1994, 59-69.
17. DAVIS, H., KNIGHT, S. *and* HALL, W. Light Hypermedia Link Services: A Study of Third Party Application Integration. *In: Proceedings of the European Conference on Hypermedia Technology,* ECHT'94 (Edinburgh, Scotland, September 18-23). New York: ACM Press, 1994, 41-50.
18. ROSSI, G., GARRIDO, A. *and* CARVALHO, S. Design Patterns for Object-Oriented Hypermedia Applications. *In:* J. Vlissides, J. Coplien, and N. Kerth, *eds. Pattern Languages of Program Design, Volume 2*, PLOP'95 (Monticello, Illinois, Sept. 5-8). Reading, Mass: Addison Wesley, 1996. *Forthcoming.*
19. BIEBER, M. *and* KACMAR, C. Designing Hypertext Support for Computational Applications. *Communications of the ACM*, *38* (8), 1995, 99-107.
20. HAAKE, J. *and* STREITZ, N. Coexistence and Transformation of Informal and Formal Structures: Requirements for More Flexible Hypermedia Systems. *In: Proceedings of the European Conference on Hypermedia Technology,* ECHT'94 (Edinburgh, Scotland, September 18-23). New York: ACM Press, 1994, 1-12.

21. JACOBSON, I., CHRISTERSON, M., JOHSSON, P. *and* OVERGAARD, G. *Object-Oriented Software Engineering.* Reading, Mass: Addison-Wesley, 1992.

22. ALVAREZ, X., DOMBIAK, G., GARRIDO, A., PRIETO, M. *and* ROSSI, G. Objects on Stage. Animating and Visualising Object-Oriented Architectures in a CASE environment. *In: Proceedings of the International Workshop on Next Generation of CASE Tools*, NGCT'95 (Jyväskylä, Finland, June 12-13). Paris: Université de Paris Press, 1995.

23. CONKLIN, J. *and* BEGEMAN, M. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions of Office Information Systems 6* (4), 1988, 303-331.

24. GAMMA, E., HELM, R., JOHNSON, R. *and* VLISSIDES, J. *Design patterns: elements of reusable object-oriented software*. Reading, Mass: Addison Wesley, 1994.

25. LANGE, D. Object-oriented hypermodeling of hypermedia-supported information systems. *In: Proceedings of the 26th. Annual Hawaii International Conference on System Science*, (Maui, Hawaii, January 1993). IEEE Computer Society Press, 1993, 380-389.

26. MARMANN, M. *and* SCHLAGETER, G. Towards a Better Support for Hypermedia Structuring: The HYDESIGN Model. *In:* D. Lucarella, J. Nanard, M. Nanard *and* P. Paolini, *eds. Proceedings of the ACM Conferences on Hypertext*, ECHT'92 (Dec. 1992). New York: ACM Press, pag 232-241.

27. GRØNBÆK, K. *and* TRIGG, R. Dessign Issues for a Dexter-Based Hypermedia System. *Communications of the ACM 37* (2), 1994, 40-49.

28. ARENTS, H. *and* BOGAERTS, W. Towards an Architecture for Third-Order Hypermedia Systems. *Hypermedia, 3* (2), 1991, 133-152.

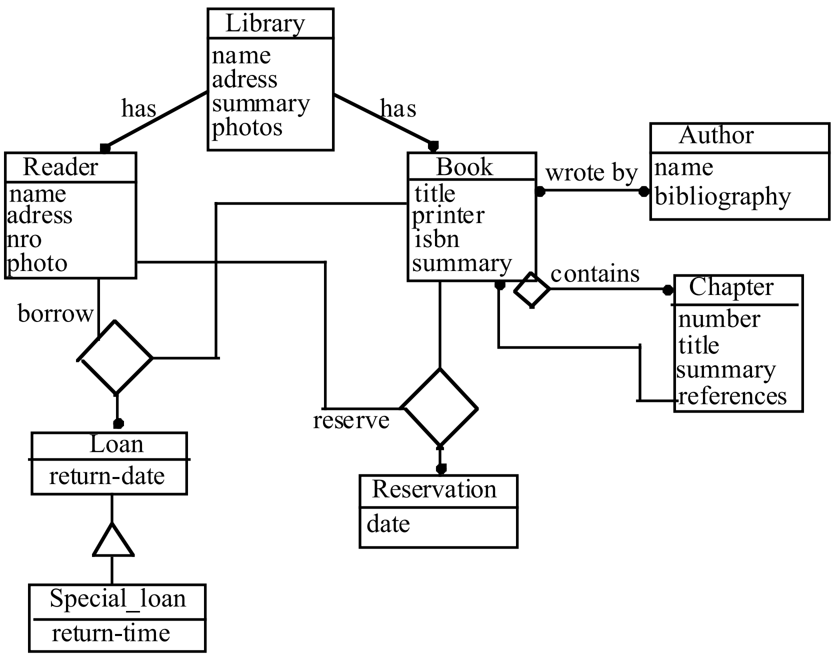

**Fig. 1*****: Model for the library application***

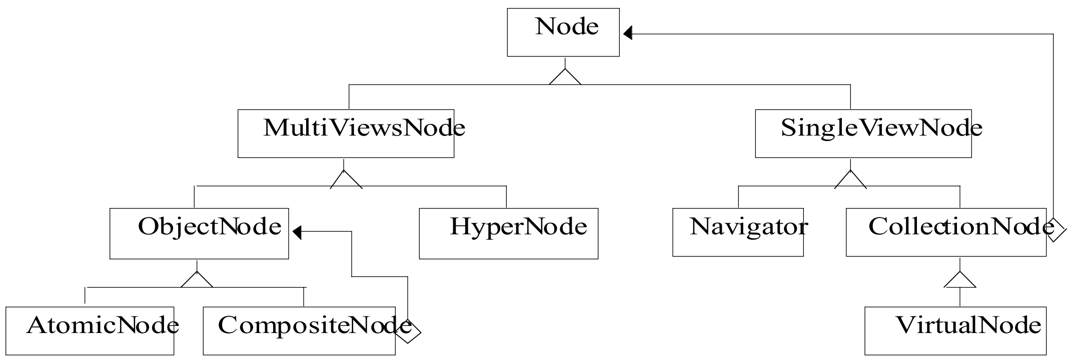

**Fig. 2:** ***Node hierarchy*****.**

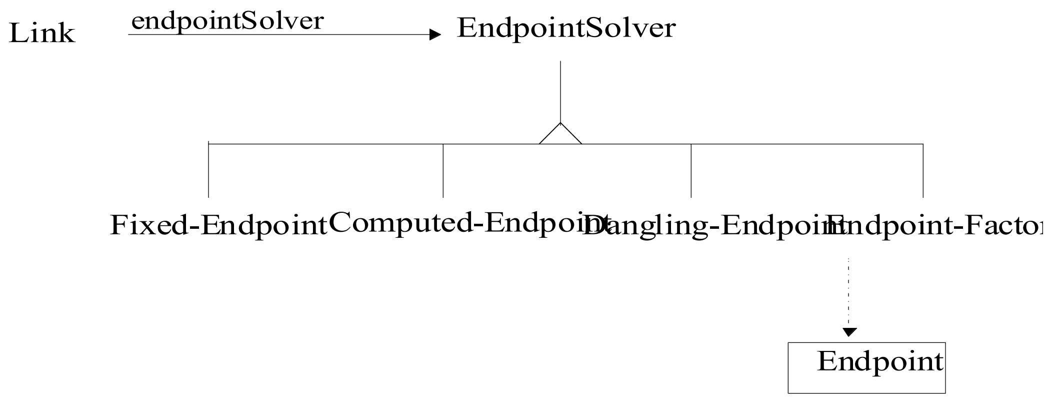

**Fig. 3:** ***Endpoint-Solver hierarchy***

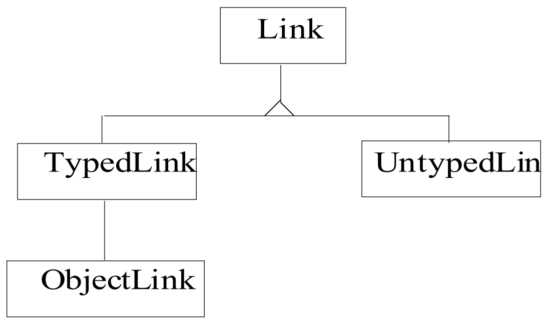

**Fig. 4:** ***Link hierarchy***

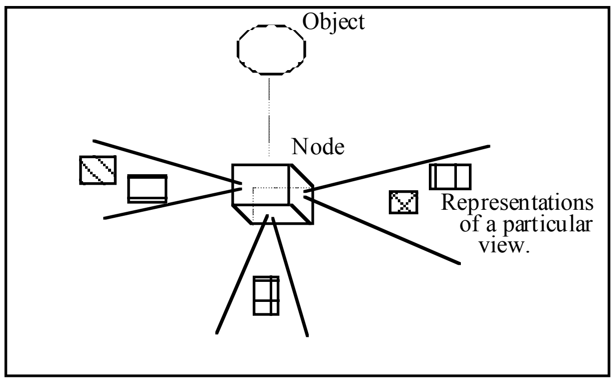

**Fig. 5:** ***Node views and Representations***