Building Hypermedia Applications as Navigational Views of Information Models

Daniel Schwabe and Gustavo Rossi*

Departamento de Informática

Pontificia Universidade Católica, Rio de Janeiro, Brazil

Presented at 28th Hawaii International Conference on System Sciences, Hawaii, Jan. 1995

Abstract: In this paper we present a novel approach for defining hypermedia applications as navigational views of an object-oriented hypermedia schwma.

We briefly describe OOHDM (Object-Oriented Hypermedia Design Model) using an academic information system as a concrete example to illustrate each modeling construct. We further analyze the whole process of hypermedia applications building focusing mainly on navigational design.

The approach we propose allows clean separation of content design, navigational design and abstract interface design. Such separation of concerns allows seamless evolution from abstract domain models to concrete implementation of hypermedia applications, especially those .in which there is a wide range of information to be handled .

Key-Words: Hypermedia, model-based hypermedia design, object-oriented models, navigational design, aggregation, composites

1-Introduction

In this paper we present a methodological and technical framework for building hypermedia applications. It allows building hypermedia applications as navigational views of an information model, represented as the conceptual schema of a hyperbase[1]. It also separates navigational views from interface presentation thus conforming with modern software engineering approaches. It combines well known object-oriented constructs (classes, objects) and abstraction mechanisms (aggregation, inheritance) with useful hypermedia concepts (hierarchical structures, perspectives, nodes and links). Though being based on an Object-Oriented Hypermedia Design Model (OOHDM [20]) our approach can be used with other similar models like [13, 11]. As pointed out in section 6, our approach emphasize the use of object-oriented concepts as modeling tools rather than as implementation artifacts, as is the case in most uses of object orientation in the hypermedia field.

Building hypermedia applications is oftentimes difficult. There are some reasons for this:

- the very nature of the hypermedia field where we must combine a rich variety of multimedia data types with navigational access [7,8, 10]; and
- the fact that in large organizations we need to handle a wide range of information systems ranging over many of its aspects and they must be integrated. These software systems usually present different views of the same shared database according to different contexts and user needs.

As it has been stated elsewhere [5, 13], hypermedia design models and in particular object-oriented models of hypermedia [14] allow the description of a hypermedia application using high level modeling constructs in an implementation independent way. Step-by-step methodologies can then be defined on top of existing design models [11].

However, in domains such as Management Information Systems (MIS), more powerful approaches are needed because applications are not standalone entities but rather logical portions of a (perhaps shared) information model. What is needed is an authoring approach, presented in this paper, that allows both describing the problem domain at a conceptual level, taking a global view independently from implementation concerns; and describing user and task oriented views, which are to be reflected in particular navigation patterns over the hyperbase. In addition, interface aspects should be specified separately of navigation patterns and of particular implementation environments. Such separation supports reuse of conceptual models (or sharing of this model between different applications), portability, and maintenance in the large as defined in [3].

The structure of the paper is as follows: we first present a concrete example, an academic MIS pointing out which aspects make it difficult to develop as a

*also LIFIA, Univ. Nacional de La Plata , Argentina, and Conicet

[1] By hiperbase we mean a collection of nodes containing (multimidia) information and a collection of links connecting these nodes.

"standard" hypermedia application. We then present our approach for the overall development process of hypermedia applications discussing which activities related with the example must be performed in each step. Next we summarize OOHDM using the example to show its modeling constructs and abstraction mechanisms. Then we show how to build navigational views as patterns of nodes and links derived from the conceptual schema. We next compare our proposal with existing work in the field, and some further issues are finally discussed.

2 - An example: the academic MIS

Consider a hypermedia-based software system for an academic department including: general department information, professors, courses, research projects, budgets, funding information, publications, etc. This software systems is to be composed of several applications[1] which support different types of users, each with different information needs. Possible types of users are visitors (to the department), students, and funding agencies. It should be clear that visitors accessing information about projects and courses will have a different understanding of the domain than a manager of a funding agency who wants to explore research projects to decide whether to give financial support or not to the department.

A more traditional development approach would treat each of these applications separately, even if they share common information. For example, a manager and a visitor will have different navigational patterns and possibly use different media types to view the information (e.g., a manager might prefer reading a brief summary of a project rather than listening to a narration of activities in a project). Clearly, maintenance efforts would be duplicated not only if there is need to update the information but mainly if there is need to modify each model because of changes in the domain.

Another approach is to build a more general conceptual model supporting both contexts (and perhaps others). However, such a model will be rather complex; first because it must integrate different points of view, then because it will include higher level abstractions to provide genericity and flexibility. To cope with the added complexity, a conceptual mechanism is necessary, that allows us to specify which aspects of the model are to be used in each context.

We propose building a shared conceptual model in which we focus on the domain objects and relationships. Later, different navigational views will be "derived" from this model taking into account the profiles of intended users. (See 7 for some comments about the development process). The conceptual model will act as a shared repository from which we will build different views (in fact navigational views) of the problem domain; each view will constitute a separate hypermedia application. (see Figure 1 for an example in the Academic MIS).

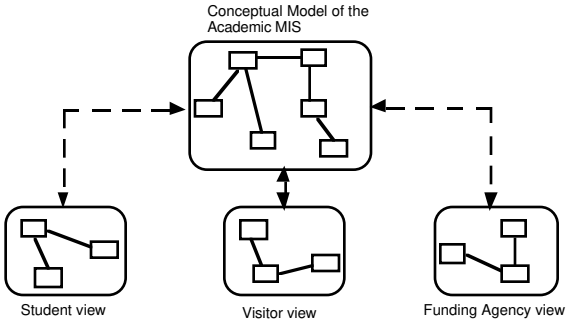

Figure 1: A shared conceptual model and its navigational views

In the following sections we will outline a methodological approach and a set of modeling constructs that allows solving the previously mentioned problems

3 - An overview of the whole process of hypermedia construction

We claim that building a hypermedia application is a four step process, where these steps are used in a mix of iterative, incremental and prototype-based styles of development. Though the focus of this paper is step 2 below, we will provide the reader with a brief description of each one.

3.1 - Step 1: Conceptual model design

During this step the shared conceptual model is described using the object-oriented hypermedia design model (OOHDM) primitives, namely: classes, relationships and sub-systems. Other hypermedia design models may be used in this step: HDM [Garzotto 91,93], EORM [13], etc.

In terms of OOHDM the following activities are performed: defining classes, sub-systems and relationships according to the domain semantics, building part-of and is-a hierarchies, assigning types to attributes, enriching relationships with cardinality information and adding instance-specific information to the schema. As we will show later, this modeling approach is similar to existing ones in the object-oriented area though enriched with some ideas (like perspectives) and applied with rather different criteria (emphasizing structure and relationships rather than behavior).

[1] it is irrelevant here whether these applications are separate or bundled into one large systems with several sub-subsystems

For the Academic MIS example, during this step we will define domain classes like: "Department," "Equipment," "Professor," "Research Project," sub-systems like "Research Sub-System" and "Courses Sub-System," relationships between classes, like "teaches," etc. Only the domain semantics are taken into account; peculiarities of applications (views) are not considered during this step.

3.2 - Step 2: Navigational design

During this step we describe hypermedia applications, by defining navigational structures that take into account the class (profile) of the intended users. Typical classes defined during this step are Nodes, Links, Indices, Guided Tours, etc.; we call them Navigational Classes. Nodes represent logical windows on classes defined in the conceptual schema and contain attributes and anchors. Attributes are information containers and anchors are link origins. Links implement previously defined relationships. Access Structures, though specified as classes, have the same meaning as in HDM, i.e. they provide ways for easy access to information.

In our example we define the "Visitor" application by specifying nodes that present information useful for a visitor, such as professors' names and areas of interest, the list of offered courses, etc., while the "Funding Agency" application will contain nodes showing more details of the domain entities including, for example, budget information. Nodes may or may not be directly derived from classes in the schema, though there is always a standard one-to-one mapping which directly translates conceptual classes and relationships into nodes and links in the navigational view. In some cases we will have nodes containing attributes of more than one conceptual class, and in others a node will basically filter information (attributes and relationships) from the corresponding conceptual class.

Other kinds of classes that will be defined during this step are those providing additional ways of accessing the hyperbase such as browsers, query interfaces, etc., or other artifacts that are used during navigation, like choosers. When the hypermedia application needs to be integrated with other kinds of "conventional" systems, navigational classes may also implement editors, report generators, etc. that though not being strictly navigational are necessary in the final system.

3.3 - Step 3: Abstract interface design

The goal in this step is to specify how the user will perceive the navigation objects through the interface; this specification is to be done at a higher level that of actual implementation environments. It is in this step that the interface metaphor and interaction styles are defined.

Hypermedia applications, like other interactive applications, require that the author must specify what are the perceptible objects the author intends to make available to the user, and how they behave in terms of the actions originating from the user (and from other perceptible objects). Perceptible objects will be in general built using primitive objects like buttons, text fields, graphic fields, etc. and will provide the interface for navigational objects as defined previously.

Some of the perceptible objects will be activated by the user, and such activation will cause transformations in the perception context (i.e., the set of perceptible objects), thus implementing the navigational style defined during Step 2. The dynamic behavior of the application can then be specified as the set of possible transformations to any given perception context. New objects appear in the perception context while others disappear[1].

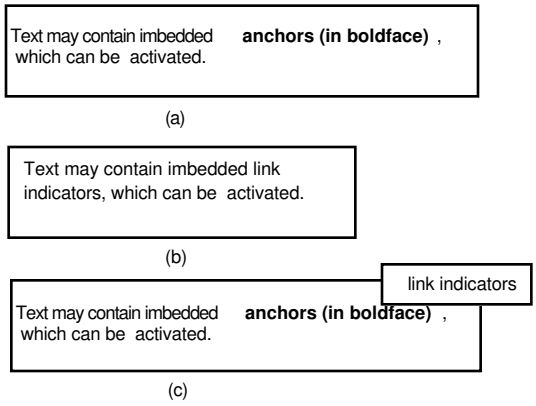

Figure 2: Example of possible interface behavior - (a) original perception context, including active perceptible object in bold; (b) possible perception context after activating perceptible object in (a); (c) alternative perception context after activating perceptible object in (a)

For example, consider the perception context depicted in Figure 2 (a), which includes two perceptible objects – a window containing the text, and an active object represented by the text "anchor (in boldface)" in bold. Two possible behaviors could be specified when the active object is activated: in (b), the active object is replaced "in-line" by another (active) object, the text "link indicators"; in (c), a new active object – a (pop-up) window with the text "link indicator" – is added to the perception context.

[1] It should be noted that this specification can be made for any kind of interactive application, and not just hypermedia applications.

During abstract interface design, the author has to map the navigational objects defined in step 2 above into perceptible objects. In particular, by mapping (abstract) anchors into active perceptible objects, it is possible to define the intended browsing semantics, i.e., how the application will behave when the user navigates in the hypermedia application. Navigational classes (such as nodes) provide information to be mapped (i.e., become contents of) perceivable objects. A clean separation between both concerns, navigational and abstract interface design, allows building different interfaces for the same model thus conforming with varying user preferences, user-interface technology or implementation restrictions. As a summary, while during Step 2 we specify what objects are going to be navigated, during Step 3 we specify in which way those objects will be perceived.

Many authors consider this step as being an implementation concern – it is true that modern Graphic User Interface libraries simplify the task of defining graphical interface objects. We claim, however, that this step must be faced in an implementation-independent way. We are now using Abstract Data Views (ADV) [2] as a design tool for specifying perceptible objects and their transformations. Abstract Data Views allows specifying the interface aspects of an object independently of the object itself. Using ADV's we specify interface aspects and behavior of Navigational Classes. Thus, we clearly separate the view and the user interface aspect of the application allowing different interfaces for the same navigational view.

After this step has been performed we have enough information to implement the application using a hypermedia system.

3.4 - Step 4 : Implementation

By mapping the abstract interface design – the perceptible objects and their transformations – into concrete interface objects, i.e., those available in the chosen implementation environment, the author produces the actual hypermedia system to be run. In particular, the model generated after performing Step 1 to 3 can be implemented on top of available hypermedia platforms such as Hypercard, Toolbook, KMS, Guide, Microcosm, etc., provided we define good heuristics for mapping the model to the platform of choice.

By virtue of being based on a set of uniform modeling constructs (objects and classes), our approach allows a smooth transition from domain modeling to navigational and interface design and into implementation. Its object-oriented nature provides a traceability model, thus allowing better understanding and maintenance of hypermedia applications. By the same token, it can also be the basis for a theory of reuse of hypermedia components as reuse of objects and classes. In the remainder of this paper we will describe Steps 1 and 2 in detail. For conciseness we shall no discuss Step 3 and 4.

4 - Building the conceptual schema for the academic MIS

4.1 - Classes, Relationships and Sub-Systems

In our model an Object Oriented Hypermedia Modeling Schema is built upon objects, classes, relationships and sub-systems. The schema consists of a set of objects and classes connected by relationships; objects are instances of classes, and thus, when a relationship holds between classes, it abstracts the object-to-object relationship. Classes may be related to sub-systems (abstractions of a whole hypermedia schema). Figure 3 shows part of the schema of the Academic MIS. The notation is similar to Rumbaugh's OMT ([19]) enriched with sub-system's information as proposed in [9] and [24]. Boxes represent Classes with name and attribute information inside; arcs represent relationships; ovals represent sub-systems (further described in what follows); and the small diamond at the end of an arc indicates aggregation.

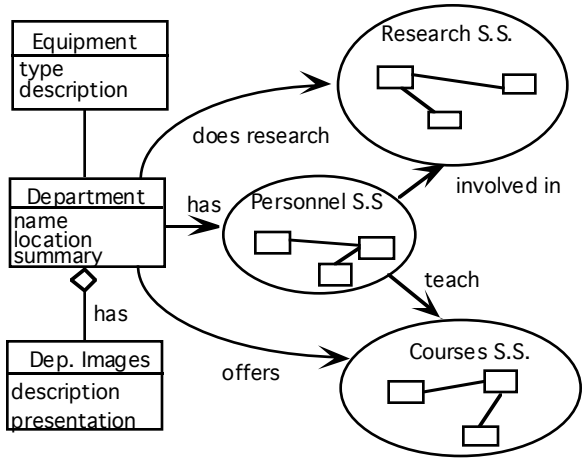

Figure 3: Conceptual Schema of the Academic MIS

In the example we have defined two main classes: "Department" and "Equipment" and three sub-systems: "Personnel," "Research" and "Courses". The rationale for defining Sub-Systems is not application-dependent but rather domain-dependent. Sub-Systems stand for complete hypermedia schemas that are used as part of another hypermedia schema. They may be used when another hypermedia schema or application already exists or may be defined opportunistically as a modularization strategy.

Usually sub-systems contain entry points, i.e. classes in the sub-system that may be accessible from other classes outside the sub-system. Entry points are defined while defining navigational semantics. Sub-systems may also be nested, i.e. a sub-system may contain others.

As in other object-oriented approaches, classes are described with typed attributes and behavior. For the sake of conciseness we will focus on structural aspects and omit the discussion about behavior.

Relationships in our model express relations between domain objects intended to be navigated by the final user, and will be mapped to links in the navigational view. Therefore, it is important to ensure that, when possible, relationships are not hidden in class attributes. For example, consider the class "Department" (as shown in Figure 3); one could conceivably specify "hardware facilities" as an attribute of class "Department", whose values are objects of class "Equipment". In this case, this attribute would be hiding a relationship between classes "Department" and "Equipment", as opposed to representing it explicitly as was done in Figure 3. As a consequence, attributes should be made, whenever possible, atomic.

It should be noted that the discussion above is independent of any particular navigational semantics; when navigating from a department to a piece of equipment, the information could be presented in the same window or not, and the same would be true had the equipment been specified as an attribute of department.

Relationships are also defined as classes thus including attributes and behavior and are further organized in hierarchies. Cardinality constraints may also be specified when defining relationships. In Figure 4 relationship "teaches" between "Professor" (in Sub-System "Personnel") and "Course" (in Sub-System "Courses") reflects the fact that the relationship is one to many and is further characterized by attribute: "semester".

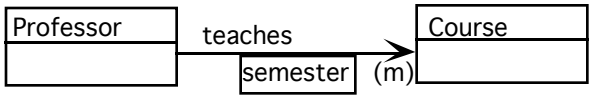

Figure 4: Relationship with an attribute

To document the schema, we propose using cards similar to CRC cards [24]. These are easy to manipulate and may contain, among other things, trace information, critical during maintenance. Backward trace information allows answering questions such as: where does this artifact come from? (a real world entity or relationship for example). Forward trace is important during system evolution for analyzing the impact of changes in the model. (What classes does this change impact?).

Finally it is worth saying that the use of high-level abstraction mechanisms makes it possible to obtain concise descriptions of complex domains, thus enhancing modularity.

4.2 - Attributes, Types and Perspectives

Class attributes are typed and represent intrinsic or conceptual properties of objects (a Department's name, its description). The type (or class) of an attribute will represent either an implicit relationship (when the type refers to other objects in the schema and as such it will be mapped to a hypermedia link – see discussion in the preceding subsection), the kind of media used to represent it, or the rhetorical appearance of the attribute in the final hypermedia application.

Each possible appearance of an attribute is called a perspective of the attribute. We use the same semantic for perspectives as HDM [5]; some examples of this feature are : the attribute "presentation" in a "Department Image" (Figure 2) might be viewed as a text , photograph, or video clip.

When multiple perspectives exist we use "[...]" and if one of them is the default one we mark it with a +. In the previous examples we would write: presentation:[Text, Photograph , Video+]. Only the default perspective must be present in all instances, while the others may or may not be implemented. Note that, as explained in [Garzotto 91,93], perspectives will originate a class of hypermedia links not explicitly specified in the schema, namely, those connecting different perspectives of the same attribute (perspective links in HDM).

4.3 - Abstraction mechanisms: Aggregation and Inheritance

In our modeling approach we provide two abstraction constructs for dealing with complexity: Aggregation and the pair Generalization/Specialization. The first one is useful for describing complex Classes as aggregates of more simple ones and the second for building Class Hierarchies and using inheritance as a sharing mechanism. In addition, the notion of sub-system may be viewed as a third high-level abstraction mechanism.

Part-of relationships (Books' chapters, Scenes of a film), are described using aggregation relationships. Aggregates in a class definition are similar to components in HDM. Implicit relationships exist between a complex object and its parts (and vice versa) and between the parts themselves. These are called structural links in HDM. Judicious definition of aggregate structures is important in hypermedia because their specification can be useful when defining navigational views. Understanding the exact nature of an aggregate, i.e. what kind of object composition it represents is important for building good navigational structures (see [18, 23]). Figure 3 shows an

aggregation (of Images) for Class "Department"; in Figure 5, "Syllabus" is defined as an aggregation of "Units".

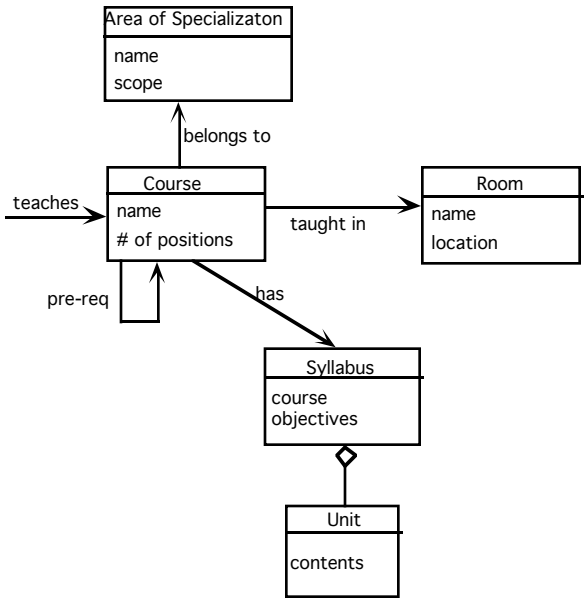

Figure 5: The Courses Sub-System

5 - Navigational design: defining the hypermedia application

As previously stated, large-scale MIS may involve several different contexts or views. Though we have already defined a conceptual schema, it may be rather abstract in terms of user needs. In our approach each possible navigational view is implemented by defining nodes, links, access structures, etc. that act as logical windows on the classes defined in the conceptual schema. In this paper we will not further elaborate on implementation issues such as building a shared hypermedia database and considering each hypermedia application as a (complex and navigational) database view. However, it is obvious that the conceptual framework presented in this paper naturally leads to this implementation scheme.

5.1 - Defining nodes

Nodes are the basic information containers in hypermedia applications. Though it may be easy to define translation rules from a conceptual model to a nodes and links structure (and such rules exist for HDM), the fact is that the structure of nodes depends on the application semantics, i.e., the particular interests of intended application users.

A node class is characterized by specifying the conceptual class(es) from which it is derived. We will call these conceptual classes the node's subjects. Node's attributes must be single-typed, i.e., when multiple perspectives exist in the conceptual class, different attributes or different nodes must be defined (one for each perspective). It should be kept in mind, however, that having different nodes does not necessarily imply that the user will actually perceive different objects, as multiple nodes may be mapped into a single perceptible object in Step 3.

In Figure 6 we show node classes "Department" and "Professor" as defined for the navigational view "Visitor":

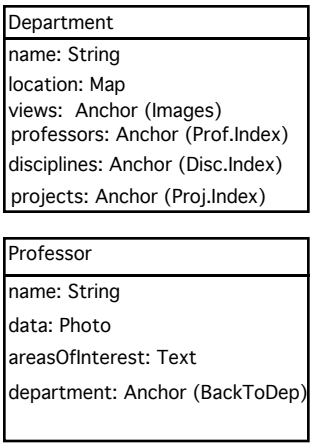

Figure 6: Node Classes for "Visitor" view

In Figure 6 the structure of nodes is derived almost directly from their corresponding conceptual classes; the node's subjects are the conceptual classes with the same name. Node attributes are originate from some of the corresponding conceptual attributes and relationships (See Figures 3 and 5). Anchors allow accessing links (like Images) or Access Structures (like Prof. Index). We have omitted defining anchors for links to "Equipment" in "Department" and to "Vitae" in "Professor" to illustrate that the author may elect not to show this to the general public. Links and Access Structures will be shown later.

An interesting issue arises when defining nodes based on aggregate structures, such as "Department Images" defined in Figure 3 for "Department". In Figure 7 we show node class "Department Electronic Videowall", a node acting as the electronic equivalent of a "videowall" (an installation with multiple TV monitors, each showing a possibly different image) with videos of the Department.

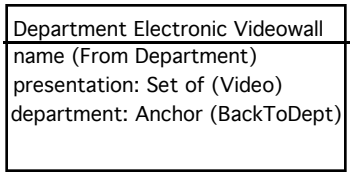

Figure 7: Node class Department Electronic Videowall

Attribute "presentation" above originates from the corresponding attribute in conceptual class "Department Images," also grouping many presentations (of "Department Images") in the same node; attribute "name" is "inherited" from node class "Department". This kind of inheritance of values is common in composite objects and allows us to clearly specify the way in which we will navigate composites.

In Figure 8 we show node classes "Department" and "Professor" defined for the navigational view "Funding Agency". Note that in this case we have filtered some attributes and relationships and added others.

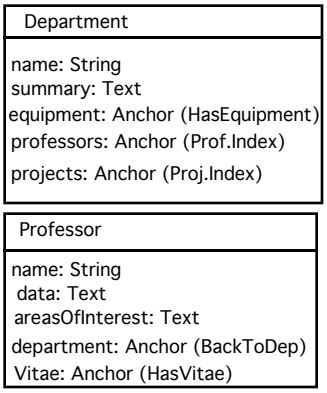

Figure 8: Node classes for "Funding Agency" view

Nodes may be formed out of a combination of features from different conceptual classes or from single nodes combined among them or with access structures and are called "composition nodes." They are usually found when defining hypermedia catalogues or when mapping aggregate structures into navigational classes (See below).

5.2 - Defining links and access structures

In our model, links implement relationships defined in the conceptual schema. In other words, links are the navigational realization of relationships. Link classes are defined by specifying link attributes and behavior, source and target objects and cardinality. Link attributes express properties of the link itself and may be useful when defining n-ary links or links with cardinality greater than one. In such cases the link may behave as a node (similar to the web's center in HDM), acting as an intermediate object (between the link source and destination) during navigation. Taking into account link behavior patterns as discussed in [13], link classes may be organized into hierarchies where abstract classes provide common behavior, and concrete ones add structure and eventually specialize behavior.

Implementing one-to-many relationships, like "teaches" (Figure 3) requires some design decisions. In Figure 9 we show link class "Teaches" using a Link card (we have omitted trace information). Such link may be defined as one-to-one, which requires multiple anchors (one for each instance) or as one-to-many, which requires a chooser or an Index as an intermediate object. The same holds for "structural" links like "Images" (mentioned in Figure 7). Anchors for bi-directional links may be derived directly from the link definition.

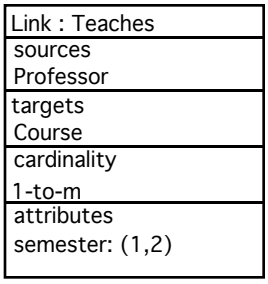

Figure 9 Definition o f Link class "Teaches"

Access structures act as indexes or dictionaries and are useful for helping the final user find the desired information (the list of Professors, a Research Projects Catalogue, a Time-line of Project Milestones, a Guided Tour through some set of disciplines, etc.). Access structures are also modeled as classes and further characterized by a set of selectors, a set of target objects (usually objects in the schema) and a predicate on target objects. The predicate expresses which objects will be accessible in terms of their properties. Selectors usually stand for some of the attributes of the target objects and are organized according to a pre-defined data structure (an ordered list, a set of icons, etc.). In either case they must be explicitly mentioned in the definition of the access structure.

Some access structures may be defined in the conceptual schema level, and may be used in any navigational view; for example the "Professor's Index" or the "List of Publications" are useful in all previously defined views. Note that this is almost always true for access structures allowing access to all instances of a class. In Figure 10 we show the Access Structure "Professor's Index" using the card formalism; we have not included a predicate on target objects because we

want the whole set of professors to be reached. A predicate could have expressed the fact that we wanted to access those professors that have some particular interest area. Note that this definition resembles conditional indexes as discussed in [11].

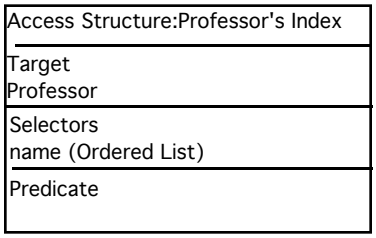

Figure 10: Professor's Index

As discussed previously, access structures may be accessed from nodes, using specified anchors or may themselves be part of a composition node – for example, a node consisting of domain information plus access structures. In Figure 11, we can see the definition of a "Professor's Catalogue" as a composition node, where the value of the attribute "index" (an access structure) is fixed for a particular catalogue and that the value of attribute "prof" varies dynamically according to the selection within the index. We have encountered this kind of node in many situations; it seems to be a recurrent pattern in hypermedia applications with a clear navigational semantics and easy to implement in hypermedia platforms.

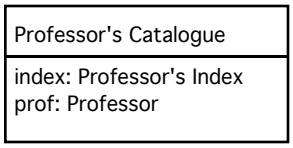

Figure 11: A Composition Node – Professors Catalogue

Finally, in Figure 12 we show part of the navigational schema for the student's view. Small boxes represent access structures and dashed lines represent links that are automatically derived from node and access structure definitions (i.e., links connecting nodes and access structures). The navigational schema gives a snapshot of the navigational structure as previously defined.

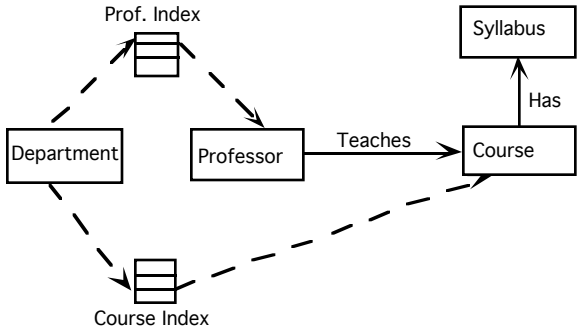

Figure 12 A Navigational schema

6 - Related work

Our work is similar to other model-based approaches to hypermedia design like HDM [5] and its descendants [6], in that it recognizes the importance of specifying a conceptual schema prior to implementation. It is based on an object-oriented model that builds hierarchical structures as aggregations of simpler ones and encourages the use of perspectives for presenting the same conceptual entity in different ways. The underlying model enriches HDM in that it provides higher level abstraction constructs (classes and objects) and mechanisms (generalization/ specialization). Our approach for building navigational views generalizes the use of Derived Entity types in HDM, which can play a similar role. Furthermore, HDM does not deal with abstract interface design.

A step-by-step methodology for designing hypermedia applications, named RMD has been recently proposed [11]. RMD is built on top of HDM and HDM2 [8]. It enhances HDM2 concepts with additional access structures (conditional indexes and guided tours) and proposes a seven step process for building hypermedia applications. Our approach is similar in that we identify several steps prior to implementation (like Navigational and Abstract Interface Design), though it is different in that it uses classes and objects during the whole process. It also formalizes navigational design as the process in which different views of the same domain are built.

Object-Oriented ideas have been used in the hypermedia field for some time now. However, with the exception of [13], objects have been mainly used as implementation artifacts (See for instance [15] or [17]). Our modeling approach differs from those mentioned in that it addresses design aspects rather than implementation ones. Though being based on well known Object-Oriented modeling approaches it includes some novel features like the use of attributes' perspectives.

Recently, [13], has proposed extending an object oriented modeling technique (OMT, [19]) with

hypermedia links semantics. The resulting design model (EORM) can be compared with the underlying model in our approach (OOHDM) in that it uses well known object-oriented concepts and mechanisms for representing application domains.

However, some key differences must be highlighted: First, the overall process is different because we divide the task of hypermedia construction so that during the task of domain modeling we only focus on objects, attributes and relationships in the domain without considering navigational semantics. Navigation aspects are addressed while constructing hypermedia applications, considering them as views on the conceptual schema. Second, the underlying O-O model provides an additional high level modularization construct, the sub-system. We also use attributes' perspectives (the object-oriented equivalent of HDM perspectives) and consider aggregation as a "first-class" relationship, which provides a natural context for defining navigational semantics (similarly to entity structures in HDM). Finally, though not shown in this paper we use instance-diagrams to enrich the conceptual schema with instance peculiarities.

Our proposal is similar to current work in the object-oriented design field, aimed at separating the application (and presentation) views from the conceptual model. [4] describes a technique for supporting specification of user interfaces and reports that allows the many and varied users of MIS to access common data in a way tailored to them. Though navigational views, as described in this paper, support only hypermedia-based navigation, the correct use of nodes and links behavior can yield a flexible and powerful architecture for building "conventional" MIS.

It is worth saying that several object-oriented database management systems (like O2 [1]) provide navigational access to the object base. In this way they could be used as a natural implementation environment of hypermedia applications designed using our model.

7 - Concluding remarks

We have presented a conceptual framework for building hypermedia applications as navigational views of an object-oriented schema. We have briefly outlined the underlying design model (OOHDM). Using this model it is possible to build a schema for a domain of hypermedia applications, using known object oriented concepts such as structure, behavior, and abstraction mechanisms such as aggregation and generalization/specialization. The model may be further enriched with instance-specific information when certain instances of a class exhibit exceptional features specific only to those instances.

The four steps outlined in section 3 provide a smooth path from high level domain modeling to implementation. Using navigational classes (nodes and links) it is possible to provide a smooth transition from domain and application modeling to concrete hypermedia design. Using abstract interface specifications, it is possible to map the hypermedia objects defined as navigational classes into perceptible objects, which can in turn be mapped into concrete implementation constructs.

This approach takes into account the very nature of MIS where different users need to access shared data in a way tailored for them. A hypermedia application is built as a set of navigational objects that act as logical windows on a shared conceptual model.

We have used our approach to model many of the applications that had been already modeled using HDM, resulting in more flexible schemata (i.e., easier to extend and to reuse). We have also modeled more complex applications, such as a hypermedia-based software engineering environment, in particular a complete CASE environment for an object oriented software engineering methodology, and another CASE environment for our model and the associated methodology. Both examples are particularly interesting since they are evolving applications, i.e., ones in which nodes are added by the readers as they use the application. We noticed that using our modeling constructs and separating the conceptual schema from the navigational views we enhance modularity thus allowing easier evolution and maintenance of hypermedia applications. Once implemented, even using sophisticated object-oriented environments, the number of nodes and links depends only on the application's complexity and the information base's size.

Some further issues must still be studied. First what is an appropriate development process for this framework?. If we begin building a business conceptual model first [16], we will not face navigational views at all until late in the process, whereas if we use scenario-driven methods [12] where navigational views are outlined with the users first, the conceptual model will then be built incrementally. Using the latter approach, we are investigating the notion of "navigation context" [21] (similar to collections as defined in [8]) as a unifying primitive, generalizing links, indexes and guided tours, allowing the specification of the navigational design according to different contexts of use.

The difference between both approaches lies in the fact that, in a "conventional" life-cycle, navigational views are a way to check and enrich the conceptual model, whereas in an incremental one, they are primarily a source of information. However, in both cases details about views should not be completed until the conceptual model is completely defined.

Another interesting issue arises when analyzing implementation aspects. As said before, a conventional

hypermedia system can easily be used as the implementation platform in our framework: mapping nodes, links and indexes to card-based environments is straightforward, and common behavioral patterns can be implemented using the native scripting language. However, more integrative architectures can be used. For example, we can implement the shared conceptual model using an object-oriented database server, and make each hypermedia application a client of the database. In this case we must define some rules, and implement them as abstract behavior, stating how information is translated from the shared database into the view and how the view updates it. The definition of guidelines for this translation, such as the ones found in [4] is straightforward. This further approach cleanly supports our view of the process and is strongly compatible with modern distributed hypermedia architectures like the ones mentioned in [14, 22] and with other object-oriented hypermedia settings [15, 13].

We are now exploring the specification formalisms for abstract interface design specification, and how to map different specifications into diverse implementation environments.

Designing and implementing evolving hypermedia applications in the domain of MIS is still a young field and using object oriented modeling techniques and well known software engineering principles is a promising approach for solving many of the problems involved.

8 - References

[1] F. Bancilhon, C. Delobel and P. Kanellakis: “Building an Object-Oriented Database System. The story of O2”. Morgan Kaufmann, 1992.

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

[3] J. Chen and S. Chang: "An object-oriented method for software maintenance". JOOP, January 1994, Vol6 N. 8 pp 46-51.

[4] M. Fowler: "Application Views: Another technique in the analysis and design armoury", JOOP, Vol 7 N 1., pp 59-66.

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

[6] 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.

[7] F. Garzotto, L. Mainetti and P. Paolini, “Adding Multimedia Collections do the Dexter Model”, Proceedings ECHT’94, Edinburgh, Sept. 1994.

[8] F. Garzotto, P. Paolini and L. Mainetti: "Navigation in Hypermedia Applications: Modeling and Semantics". Journal of Organizational Computing (forthcoming).

[9] C. Gilliam: “An approach for using OMT in the development of large Systems”. Journal of Object-Oriented Programming, Feb 1994, Vol 6, N. 9 pp 56-60.

[10] L. Hardman, D. Bulterman and G. Van Rossum: "Links in Hypermedia: the Requirements for Context", Proceedings Hypertext'93, ACM, pp 183-191.

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

[12] I. Jacobson: “Object-Oriented Software Engineering: A Use Case Driven Approach”, Addison-Wesley, 1992.

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

[14] D. Lucarella: "Multimedia Object Retrieval Environment", Proceedings of Hypertext'93, pp 39-50.

[15] M. Marmann and G. Schlargeter. "Towards a better support for hypermedia authoring: The HYDESIGN model", Proceedings of ECHT'92, ACM Press, pp 232.

[16] J. Martin and J. Odell: “Object-Oriented Analysis and Design”, Prentice Hall, 1992.

[17] J. Nanard and M. Nanard. "Using Structured Types to Incorporate Knowledge in Hypertext, Proceedings of Hypertext’91 ACM Press. pp. 329.

[18] J. Odell: "Six different kinds of compositions", Journal of Object Oriented Programming, Vol. 5 N. 8, 1994.

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

[20] D. Schwabe and G. Rossi "OOHDM. An object-oriented Hypermedia Design Model", Technical Report, PUC-Rio.

[21] Schwabe, D. and S.Barbosa, “Modelling Navigation in Hypermedia Applications”, Technical Report, Departamento de Infomática, PUC-Rio, 1994

[22] U.K. Wiil and J.J. Leggett: “Hyperform: Using extensibility to develop dynamic, open and distributed hypertext systems”, Proceedings ACM Conference on Hypertext, Milano 1992, pp 251,261.

[23] M. Winston, R. Chaffin and D. Herrmann: "A taxonomy of part-whole relationships", Cognitive Science, Vol. 11, pp 417-444, 1987.

[24] R. Wirfs-Brock, L. Wiener and R. Wilkerson: "Designing Object-Oriented Software", Prentice Hall, 1990.