From Domain Models to Hypermedia Applications: an Object-Oriented Approach.

Daniel Schwabe and Gustavo Rossi [1]

Departamento de Informática
Pontificia Universidade Católica
R. M. de S. Vicente, 225
Rio de Janeiro, RJ 22453-900, Brazil
Fax: +55-21-511 5645. Telephone: +55-21-529 9544
E-mail: [schwabe,rossi]@inf.puc-rio.br

1 also LIFIA, Universidad Nacional de La Plata
La Plata, Argentina, and Conicet.

Abstract

In this paper we present an object-oriented method for designing hypermedia applications. The approach divides the development process in four steps, namely: domain (or content) design, navigational design, abstract interface design and implementation. We use similar modeling primitives (object and classes) and abstraction mechanisms (aggregation, generalization), during the whole process thus improving traceability; design decisions like the use of complex navigational structures are made explicit through a uniform notation thus allowing a coherent document structure that simplifies the construction of a CAHDE.

1 - Introduction and rationale

Building large hypermedia applications is difficult, and is further complicated by the fact that, once an application has been built, its maintenance is correspondingly more complicated. Moreover, as in other software domains (such as information systems, databases, etc.) hypermedia applications are usually built from scratch: reuse is still a dream. As it has been stated elsewhere [Garzotto91], hypermedia design models and in particular object-oriented models [Lucarella93, Lange94] allow the description of a hypermedia application using high level constructs in an implementation independent way. Step-by-step methodologies can then be defined on top of existing design models [Balasubramaniam 94]. However, design models are still in their infancy; design decisions are often taken at the wrong time during the development life-cycle or are poorly documented, thus difficulting evolution.

In this paper we propose a step-by-step method based on the construction of a sequence of object-oriented models that leads from domain analysis to implementation. Although based on object-oriented concepts, the resulting design can be implemented on top of a conventional (i.e., non object oriented) platform. The key ideas underlying the method are also suitable to other modeling approaches (like HDM [Garzotto 91,93], EORM [Lange94], etc...); in fact they are compatible with other step-by-step methods (for example the one presented in [Balasubramaniam94]).

Using well-known object-oriented modeling concepts allows the formulation of complex designs with a concise yet expressive notation thus simplifying the construction of Computer Aided Hypermedia Development Environments (CAHDE).

The structure of this paper is as follows: in section 2 we briefly overview our approach for building hypermedia applications; in sections 3 and 4 we present the core of our method: specifying a hypermedia application as a navigational model derived from a conceptual schema, discussing modeling constructs and abstraction mechanisms. Finally we summarize the key aspects of our approach and discuss some additional issues.

2 - Overview of our approach

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. In each step a model is built or enriched; after building the last one, we have enough information to implement the hypermedia application. During each step we favor the use of a model-based (as opposed to method-based) approach; moreover as already stated, the order of steps in indicative and not prescriptive.

2.1 - Step 1: Conceptual Model Design

During this step a conceptual domain model is described using an object-oriented hypermedia design model (OOHDM) whose modeling primitives are: classes, relationships and sub-systems. Other hypermedia design models may be used in this step: HDM, EORM, etc... This modeling approach is similar to existing ones in the object-oriented field though enriched with some ideas (like attribute perspectives) and applied with rather different criteria (emphasizing object structure and relationships rather than object behavior). OOHDM can be considered a direct descendant of HDM, providing some higher level modeling constructs (sub-systems) and abstraction mechanisms (class hierarchies).

In terms of OOHDM the following activities are performed: definition of classes, sub-systems and relationships according to the domain semantics, building of part-of and is-a hierarchies, assignation of types to attributes; enrichment of relationships with cardinality information and addition of instance-specific information to the schema. Modeling constructs and abstraction mechanisms are similar to the ones found in object-oriented modeling approaches like OMT [Rumbaugh91]; class attributes may be multiply typed thereby defining different perspectives of the same information.

2.2 - 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, and the set of tasks they are to perform using the system. Typical classes defined during this step are Nodes, Links, Indices, Guided Tours, etc.; we call these Navigational Classes.

Navigational operations like following a link or selecting an item in an index are specified by defining the transformations on the Navigational Space, i.e. the set of accessible navigational objects. The Navigational Space plays a similar role as the “rhetorical space” in Sepia [Thüring et al, 1991].

Nodes represent logical windows on classes defined in the conceptual schema and links are derived from relationships in the schema. Different Navigational Models may be built for the same Conceptual Schema thus expressing different views of the same information base. This approach is highly compatible with modern design and implementation architectures with a shared database acting as an information server for client (hypermedia) applications [Fowler94].

2.3 - Step 3: Abstract Interface Design

The goal in this step is to specify how the user will perceive the navigational objects through the interface; this specification is to be done at a higher level than 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.

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.

We are now using Abstract Data Views (ADV) [Cowan93] 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. After this step has been performed we have enough information to implement the application using a hypermedia system.

2.4 - Step 4: Implementation

By mapping the navigational and abstract interface models – the perceptible objects and their transformations – into concrete 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 Steps 1 through 3 can be implemented in a straightforward way on top of available hypermedia platforms such as Hypercard, Toolbook, KMS, Guide, Microcosm, etc.. Rules for performing this implementation can be systematically applied or incorporated in a CAHDE built on top of the target environment.

In this paper we will discuss in more detail Steps 1 and 2 of our method; a brief description of step 3 will also be presented, and we will only mention Step 4.

3 - Building an Object-Oriented Hypermedia Domain Model

An Object Oriented Hypermedia Modeling Schema (the Conceptual 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 1 shows part of the schema of a Tourism application. The notation is similar to Rumbaugh's OMT ([Rumbaugh91]) enriched with sub-system information as proposed in [Guilliam94, Wirfs-Brock90]. 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.

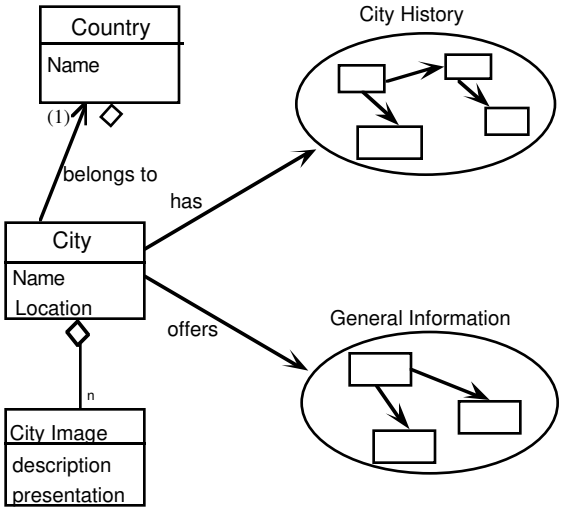

Figure 1: Conceptual Schema of a Tourism Application

In the example we have defined three main classes: City, City Image (in fact Cities are aggregations of City Images) and Country, and two sub-systems: "City History" and "General Information". The rationale for defining Sub-Systems is not application-dependent but rather domain-dependent. Sub-Systems stand for complete hypermedia schemata 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 Views. It should be noted that relationship definition is independent of any particular navigational semantics; when navigating from a city to the country it belongs to, the information could be presented in the same window or not. We will define the navigational semantic of relationships when deriving links during Step 2, and the way navigation will be perceived while defining interface aspects, during Step 3.

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 1, cities belong to exactly one country.

3.1 - Attributes, Types and Perspectives

Class attributes are typed and represent intrinsic or conceptual properties of objects (a City name, its location). 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), the kind of media used to represent the attribute or its rhetorical appearance 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; some examples of this feature are : the attribute "location" in a "City" (Figure 1) might be viewed as a text or a bitmap, the attribute "presentation" in "City Image" might be viewed as a text, a bitmap, or a video. When multiple perspectives exist we use

"[...]" and if one of them is the default one we mark it with a +. In the previous example we would write:

presentation: [Text, Bitmap, 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 as relationships in the schema, namely, those connecting different perspectives of the same attribute (perspective links in HDM).

3.2 - 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 (e.g., book 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, corresponding to the 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 [Odell94, Winston87]). Figure 1 shows an aggregation (of Images) for Class "City"; in Figure 2, "Country" is defined as an aggregation of "Political System", "Geography" and "People" Classes.

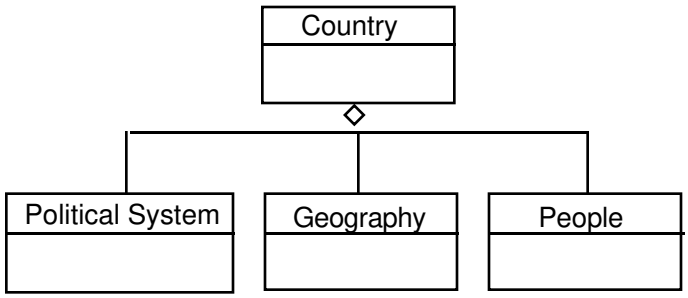

Figure 2: Country as an aggregate of other Classes

Classes inherit attributes, parts-structure, relationships and behavior of their super-classes. In Figure 3, the relationship between "Tourist Attraction" and "Town", reflects the fact that instances of each sub-class, "Tourist Place" and "Events", are located in a "Town" and that different kinds of "Tourist Places". share attributes defined in "Tourist Attraction". In our modeling approach, inheritance follows the usual semantics in object-oriented data models as defined in [Lucarella93, Rumbaugh91]. As previously stated, the use of class and aggregation hierarchies is a natural extension to HDM providing concise and easy to extend schemata. In Figure 4 we show the Conceptual Schema for sub-system "General Information" .

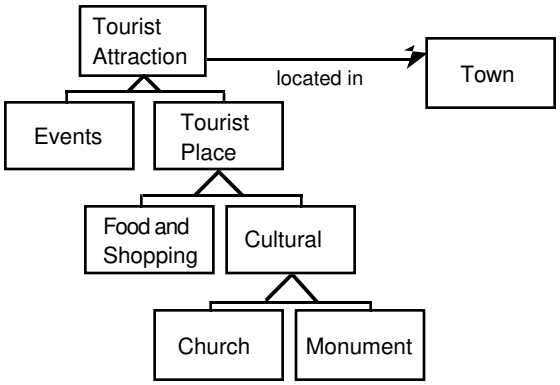

Figure 3: A Class hierarchy of Tourist Attractions

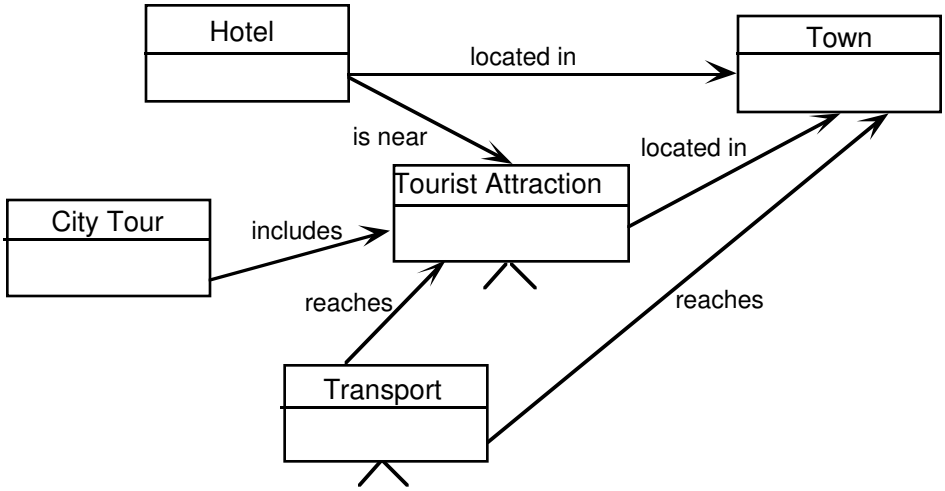

Figure 4: "General Information" Sub-System

In Figure 4, some classes, such as "Tourist Attraction" and "Transportation" have sub-classes (not shown to simplify the diagram). In Figure 5, we refine the definition of "Hotel"; in section 4 we will show Node definitions for classes "City Tour" and "Hotel". Note that we could have defined a more abstract class "Lodging" including hotels, hostels, bed and breakfast, pensions, etc.

3.3 - Documenting the Model

A very common problem when building large applications (in particular hypermedia applications) is that it is difficult to document the schema; to make matters worse Computer-Aided Software Engineering Environments are scarce in the hypermedia field.

To document the schema, we propose using cards similar to CRC cards [Wirfs-Brock90]. These cards are easy to manipulate and may contain, among other things, trace information, critical during maintenance.

We use Class, Relationship and Sub-system cards for documenting the model. Figure 5 shows Class and Sub-System cards templates and the Class card for "Hotel". Cards include information about the artifacts they document and trace information. Backward trace allows answering questions such as: where does this artifact come from? (a real world entity or relationship for example). Forward trace is important in system evolution for analyzing the impact of changes in the model: (What Navigational Classes does this change impact?). Cards are easy to manipulate and can be readily automated in a hypermedia environment.

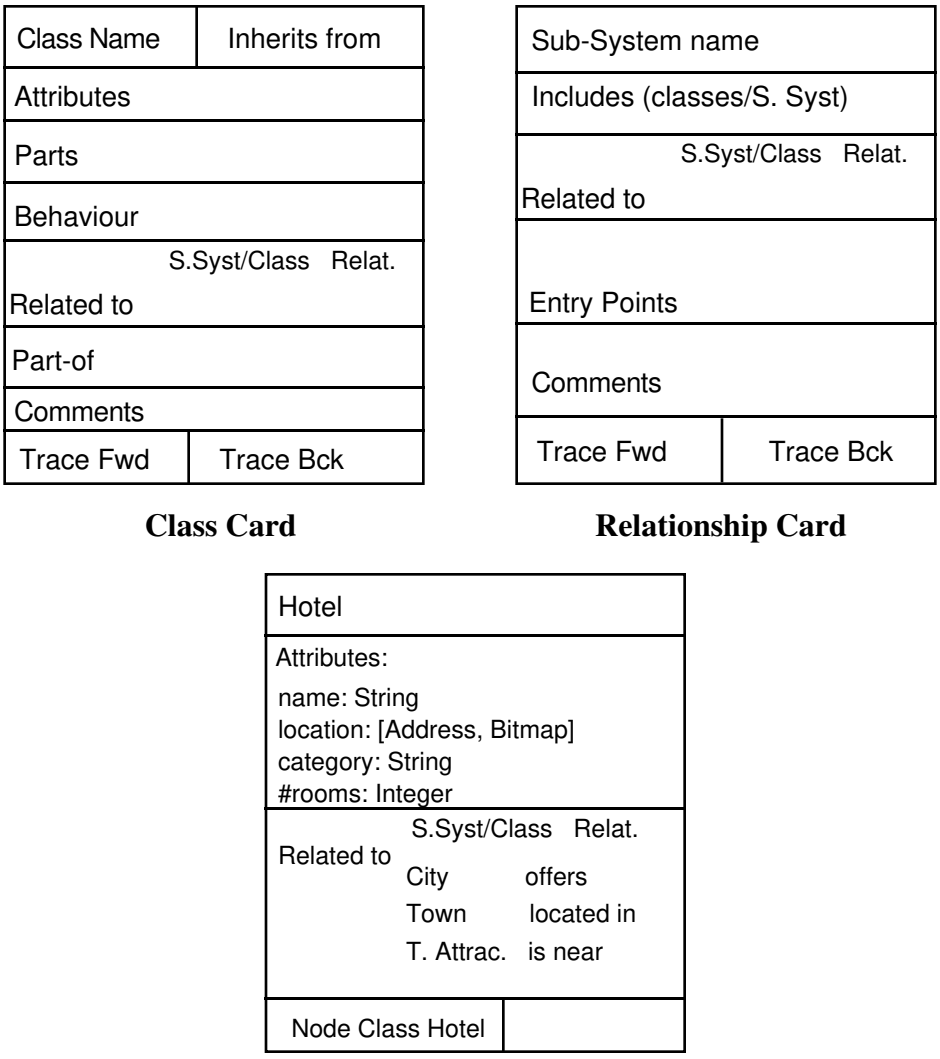

Figure 5: Class and Relationships Cards. An example: Class Hotel

In class card "Hotel" we have omitted comments and information about parts and behavior. The field on the bottom left indicates that Node class "Hotel" is derived from this Class; in a computer-based implementation, this reference could be implemented as a hypermedia link.

4 - Navigational Design

Though we have already defined a conceptual schema, it may be rather abstract in terms of user needs, and as previously stated it does not include information about navigational aspects.

In our approach a hypermedia application is derived from the conceptual model by defining nodes, links, access structures, etc. that act as logical windows on the classes defined in the conceptual schema. Different hypermedia applications may be derived from the same schema, each one supporting the needs of a particular kind of user; in our example we can build a hypermedia application providing detailed access to accommodation resources like hotels, stressing information on transportation, etc. for use by a tourist visiting the city. In the same way we can build the touring view in which the user (a prospective tourist) will navigate through "Tourist Attractions"; in this view we will emphasize multimedia presentations, like videos, images, etc...

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, the conceptual framework presented in this paper naturally leads to this implementation scheme.

Deriving a hypermedia application from a Conceptual Schema involves defining Nodes, Links, Access Structures, Navigational Contexts and transformations on the Navigational Space.

4.1 - Defining nodes

Nodes are the basic information containers in hypermedia applications and their structure 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 (its subject(s)), its attributes and the anchors it contains. Node attributes must be single-typed, i.e., when multiple perspectives exist in the conceptual class, different attributes or different node classes must be defined (one for each perspective). It should be kept in mind, however, that having different node classes for different perspectives 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. Node Classes may be grouped in inheritance hierarchies (see example below) that in general will mirror the class hierarchy in the Conceptual Schema.

In Figure 6 we show Node Classes “City” as defined for the navigational views “Tourist” and “Touring".

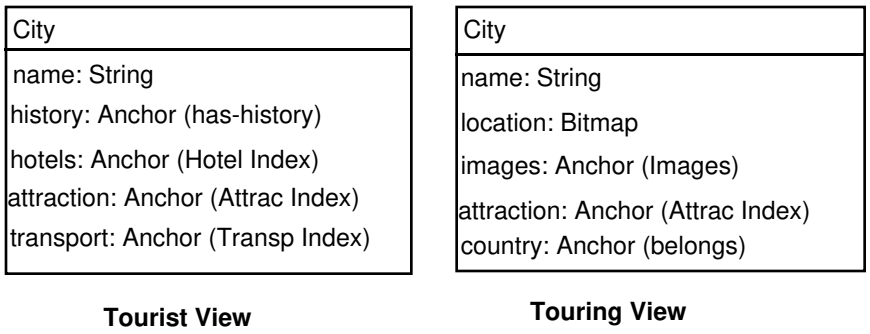

Figure 6: Node Class City for Tourist and Touring views

In Figure 6 the structure of nodes is derived almost directly from their corresponding conceptual classes; the node subjects are the conceptual classes with the same name. Node attributes are derived from some of the corresponding conceptual attributes and relationships (see Figures 1 and 4). Anchors allow accessing links (such as “has-history”) or access structures (such as “Hotel Index”). In the “Tourist” view, we have omitted anchors for accessing “City History”, and correspondingly we will omit defining links from “City” to “City History”. In the same way, the “Touring” view includes city images and excludes information about hotels.

In Figure 7 we show node class “Hotel” (derived from conceptual class “Hotel”).defined for the navigational view “Tourist".

Note that in this case we have filtered some attributes and relationships and added others. Links and access structures are further explained in the next sub-section.

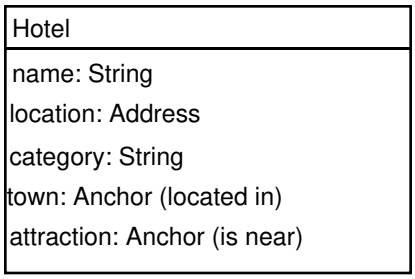

Figure 7: Node Class Hotel for Tourist view

Nodes may be formed out of a combination of features from different conceptual classes (similar to derived entities in HDM) combined among them or with access structures.

4.2 - 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. In such cases the link may behave as a node (similar to the web 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 [Lange94], 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 “offers” (Figure 1) requires some design decisions. In particular, though “offers” has been originally defined as relating “City” with the Sub-System “General Information", when refining this Sub-System “offers” must be detailed further.

In Figure 8 we show link class “is near ” using a Link card. Such a 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 the one implementing the aggregate structure in “City". Anchors for bi-directional links may be derived directly from the link definition.

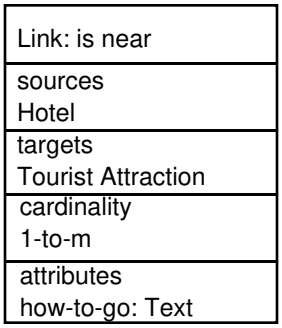

Figure 8: Definition o f Link class “is near”

Access structures act as indexes or dictionaries and are useful for helping the final user find the desired information (the list of “Monuments” or “Restaurants” in a “City”, a “Hotel Catalog”, a guided tour through some selected “Tourist Places”, 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.

Behavior in access structures is useful for specifying the way in which the access structure will be used; for example a guided tour will include operations such as “Start", “Next", “End", etc. implementing different semantics as discussed in [Garzotto94].

Some access structures may be defined in the conceptual schema level, and may be used in any navigational view; for example the “Hotel Index” or the “List of Monuments” are useful in different views. Note that this is almost always true for access structures allowing access to all instances of a class. In Figure 9 we show the Access Structure “Hotel Index” using the card formalism; we have not included a predicate on target objects because we want the whole set of hotels to be reachable. A predicate could have expressed, for example, the fact that we wanted to access those hotels located in a particular town or with fares within a particular price range. Note that this definition resembles conditional indexes as discussed in [Balasubramaniam94].

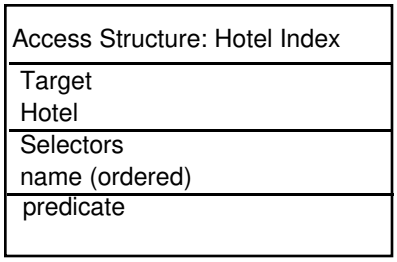

Figure 9: Hotels Index

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 10, we can see the definition of a “Monuments Catalog” as a composition node. The value of the attribute “index” (an access structure) is fixed for a particular catalog and that the value of attribute “monum” varies dynamically according to the actual selection made within the index (as defined in the class behavior, which is not shown). We have encountered this kind of node in many actual examples; it seems to be a recurrent pattern in hypermedia applications, with a clear navigational semantics and easy to implement in hypermedia platforms.

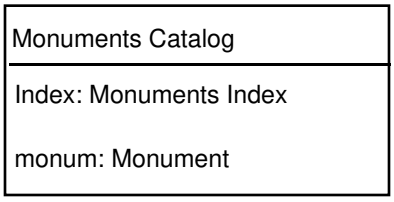

Figure 10: A Composition Node – Monuments Catalog

Finally, in Figure 11 we show part of the navigational schema for the “Tourist” 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.

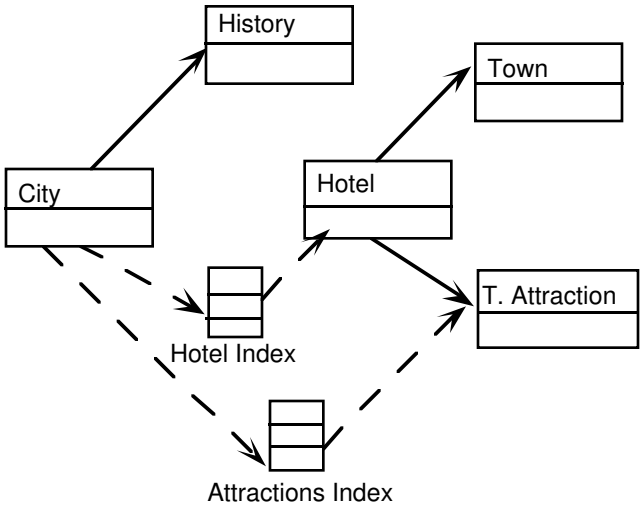

Figure 11 A Navigational schema

In the example above (Figure 11) we kept the navigational schema simple since we defined "Tourist Attraction" as an abstract node class mirroring the hierarchy in Figure 3. However, a more detailed design would need different access structures for Churches, Monuments, etc. In fact, "Attractions Index" can be defined as an index of indexes, where each of the indexes points to all instances of a particular kind of "Tourist Attraction".

4.3 - Navigational Contexts and the Navigational Space

As previously said, defining the Navigational Model (in the OOHDM terminology, a Navigational View) involves defining which are the objects that the user will navigate; those objects are: Nodes, Links and Access Structures. We complete the Navigational model by specifying Navigational Contexts and transformations to the Navigational Space.

To make the discussion concrete let us return to our previous example and suppose that each "Tourist Attraction" has a detailed aggregation structure or that some attractions (e.g., monuments) are further linked to nodes representing the artists that created them. To avoid disorientation we may want to specify that a user navigating through hotels is not able to navigate "into" "Tourist Attractions"; in other words, when the user navigates from a "Hotel" to a "Tourist Attraction" he can only see the top level of the hierarchy (a "snapshot") of the attraction, and is prevented from navigating into the aggregation structure.

On the other hand, we want to allow tourists wanting to directly explore "Tourist Attractions" by navigating from the corresponding index, to be able to do so. To solve this problem we define two Navigational Contexts: "Hotels" and "Attractions" and refine node class definitions, specifying attributes and anchors for each Navigational Context that makes sense for the node. In this way, when the user navigates to a "Tourist Attraction" in the "Hotels" context, the corresponding node will not contain anchors to its parts or to other nodes.

When the user explores the "Attractions" context, nodes will contain "full" information. In Figure 11 we show how the previous discussion is reflected in the definition of node class "Monument". Defining the class consists in defining which are the global "attributes" (i.e. those accessible in all Navigational Contexts), and which ones are "private" to a given context. In this case, we have only described the "Attraction" context; navigating to nodes of this class in other contexts gives access only to global attributes ("name" and "description").

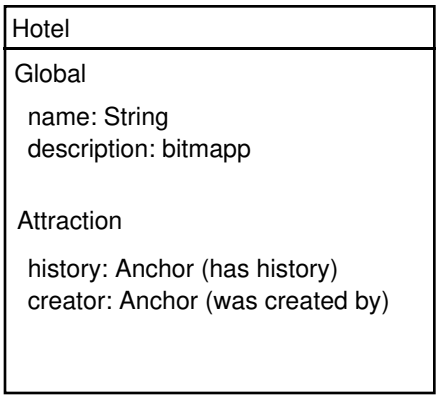

Figure 11: Defining a Node Class according to Navigational Contexts

Note that the definition of Navigational Context does not take into account the user's profile but the navigation paths that may be followed. Navigational Context may be nested in a similar way as in [Casanova 91] and they solve the problems addressed in [Garzotto94a]. Note that link traversal may change the actual navigational context; we specify this fact as part of the link behavior. Reference [Barbosa 94] discusses Navigational Contexts in detail.

Each time the user follows a link, navigating from one navigational object to another, the Navigational Space, i.e. the set of available navigational objects, may change: for example we may want to specify that when following a link from "Hotel" to "Attraction" the node "Hotel" is no longer accessible; or we may want to define navigation on hierarchical structures such that when leaving a part (component) of the structure, by navigating to a different structure, the whole aggregate structure disappears (i.e., is no longer accessible). These transformations on the set of navigable objects, which are specified as part of link behavior, constitute the navigation semantics.

5 - Defining the Abstract Interface Model – an Overview

Once the Navigational Model has been built we must define the appearance of navigational objects as well as the interaction style. Many authors consider this step as being an implementation concern, since 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.

Given the definition of navigational objects and of the navigational style – i.e. which transformations occur to the Navigational Space and in which way objects will be navigated according to the navigational contexts they belong to – we now turn to the interface metaphor and the perceptible transformations occurring during execution of the application.

In our method defining an Abstract Interface Model consists in the specification of Interface Classes and their connections to navigational classes. In its simpler form, the Interface Model will include a Class (an Abstract Data View) for each node class (the View owner); its attributes will express the way in which each Node attribute and anchor is perceived by the user. In more complex cases, Interface Classes will correspond to combinations of Navigational Classes.

The dynamic behavior of the interface, i. e. the way in which user operations affect the perception context is specified as methods in the Abstract Data View. Methods may be associated with external events: the user selecting an anchor, or with "internal" ones: a node is being opened or closed. Interface behavior complements navigational behavior as specified in Step 2: during navigational design we specify the cognitive aspects of navigation while during Interface design we emphasize perceptive aspects.

Depending on the actual definition of the Abstract Data View classes, one may obtain a more "concrete" or more "abstract" interface definition. A "concrete" one would specify actual screen locations, window panes, formatting, etc..., whereas an "abstract" one would only specify which

objects are perceptible, and among these which can be activated. In this latter case, activating a perceptible object will cause transformations to the set of perceptible objects (called the perception context), removing some from the current context and adding others.

An Abstract Interface Model may also include interface objects not previously defined that implement the interface metaphor and that communicate by message passing with other interface objects. For example, the notion of "focus of attention", which oftentimes is realized concretely as a cursor or highlight.

This object-oriented approach to interface specification is highly compatible with modern trends in User Interface Design and, and is similar to the way in which some commercial hypermedia environments such as Toolbook and Hypercard treat the user interface. In these systems, however, all objects are interface objects, thus merging navigational and interface objects. Once we have finished the interface model (that can of course be prototyped using modern user interface libraries) we have all the information we need to begin with implementation.

6 - Implementation

Being implementation highly dependent of the target environment we will only outline two possible implementation styles: using a hypermedia environment and using a general purpose object-oriented system.

In the first case the usual alternative will be mapping the interface objects to the system defined objects: cards, pages or frames. As we mentioned before many conventional hypermedia environments do not separate user interfaces from nodes, but provide a scripting language in which we will have to code both the interface and the navigational transformations. Nevertheless, provided we have clearly documented the products produced in Step 2 and 3, implementation is rather straightforward.

In contrast, it is more difficult to build different Navigational Views of the same Information Base, mainly because data objects and interface objects are the same objects, thus requiring the implementor to manage all the mapping between the two types of objects. Using an object-oriented hypermedia platform like MacWeb [Nanard91] may simplify the translation from model to implementation, reducing the "impedance mismatch" between concepts in both worlds.

The second alternative is the most interesting in some domains such as Management Information Systems. In this case the conceptual model can be implemented as a shared object-oriented repository (possibly on top of an object-oriented data base management system, see [Bancilhon92]), and different Navigational Views will be built as object networks that refer to the shared database for getting and storing information. Abstract Data Views can be easily implemented using this alternative, as well as other interface paradigms (like the model-view-controller). It is obvious that this alternative is the best for dynamic hypermedia applications, i.e. those in which nodes and link are modified/added while the system is used: CASE environments and decision support systems are a good example of this kind of application.

As previously said the four steps of the life-cycle of a hypermedia application design and development are performed combining different process models: cascade, incremental, iterative, etc. In this sense our approach favors the use of a model-based approach over a method-based.

7 - Summary. Goals of our approach

We briefly summarize in this section the main goals of our method in relation to the problem of hypermedia design. First, our approach identifies three steps previous to implementation in which different problems are addressed:

1 - domain modeling and conceptual schema construction.

2 - definition of navigational structures and semantics.

3 - definition of perception aspects and dynamic behavior.

In each step we use high level modeling constructs: classes and abstraction mechanisms – aggregation, inheritance – to reduce complexity. Modeling constructs enrich conventional object-oriented approaches with hypermedia oriented abstractions like perspectives, anchors, access structures, etc...

We also propose a simple documentation framework: the use of Cards that not only allows documenting each artifact defined during the design process but mainly support a simple but powerful traceability model.

Clear separation of concerns from steps 1 to 3 allows building different hypermedia applications for the same domain model by deriving different Navigational Schemata from the same Conceptual Schema. Moreover, different interfaces may be built for the same hypermedia application. Some novel concepts are also introduced in this approach, the most important ones being the specification of Navigational Contexts, the transformations on the Navigational Space, and the notion of Abstract Interface Design. Finally, it is worth saying that though using rich modeling primitives, the resulting model can be implemented using a conventional hypermedia environment.

8 - Related Work

Our work is similar to other model-based approaches to hypermedia design like HDM [Garzotto93] and its descendants, 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.

Recently HDM has been extended with the notion of collections [Garzotto 94a], which are similar in spirit to Navigational Contexts. One important difference is that collections are composed of the equivalent to our "conceptual objects" and not of navigational nodes. Furthermore, these objects are the same in all collections they belong to, and navigational nodes can be particularized depending on the context.

A step-by-step methodology for designing hypermedia applications, named RMD has been recently proposed [Balasubramaniam94]. RMD is built on top of HDM and HDM2 [Garzotto94b]. 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 [Lange94], objects have been mainly used as implementation artifacts (See for instance [Marmann92, Nanard91]). 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 attribute perspectives.

Recently, [Lange94], has proposed extending an object oriented modeling technique (OMT, [Rumbaugh1]) 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 attribute perspectives (the object-oriented equivalent of HDM perspectives or HDM2 slots) 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 like [Fowler94]. The key difference between our object-oriented method and those like Fowler's or Wirfs-Brock's [Wirfs-Brock90] is that ours emphasizes structure and relationships over behavior (during Step 1) and uses behavior for specifying Navigational and Interface Transformations.

9 - Conclusions

We have presented an Object Oriented Hypermedia Design Method. Using this method it is possible to build a complex hypermedia application as a stepwise process beginning with domain analysis and proceeding with navigational and abstract interface design. The method is based in the construction of different models: Domain, Navigational and Interface model, using known object oriented concepts such as object structure and behavior, abstraction mechanisms such as aggregation and generalization/specialization which provide inheritance. Though not mentioned in this paper each model may be further enriched with instance-specific information when in a particular application, certain instances of a class exhibit exceptional features particular only to those instances.

The four steps outlined in section 2 provide a smooth path from high level domain modeling to implementation. Using Navigational Classes it is possible to provide a seamless transition from domain and application modeling to concrete hypermedia design. Using abstract interface specifications, it is possible to map the hypermedia objects defined in Navigational Classes into perceptible objects, which can in turn be mapped into concrete implementation objects.

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 complexity and the information base size.

Using well known object oriented analysis and design guidelines it is possible to obtain modular and well-structured hypermedia schemata in which connection patterns among instances reflect patterns defined in the class schema. In this way, it reduces the gap between design and implementation and helps during testing and evolution of the application.

Other aspect that have been addressed in our work is the need for a traceability model allowing to trace design decisions into an implementation, for helping during maintenance, reuse of existing artifact, etc. The use of a uniform formalism in which all key decisions in hypermedia application building must be recorded, together with a set of documentation aids (Class Cards) goes straight in that direction. As previously discussed designing and implementing very large evolving

hypermedia applications is still an open field and using object oriented modeling techniques is a key approach for solving the problems we are facing with.

10 - References

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

[Bancilhon92] F. Bancilhon, C. Delobel and P. Kanellakis: “Building an Object-Oriented Database System. The story of O2”. Morgan Kaufmann, 1992.

[Barbosa 94] Barbosa, S. and D. Schwabe, “Navigation Modeling in Hypermedia Applications”, Technical Report, Departamento de Informática, PUC-Rio, 1994

[Casanova 91] Casanova, M. A; Tucherman, L.; Lima, M.J.D.; Rangel Netto, J.L.M.; Rodriguez, N.; Soares, L.F.G.;, “The Nested Context Model for Hypertdocuments”, Proceedings of the Hypertext’91, San Antonio, 1991.

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

[Fowler94] M. Fowler: “Application Views: Another technique in the analysis and design armoury", JOOP, Vol. 7 N 1., pp. 59-66.

[Garzotto91] F. Garzotto, P. Paolini and D. Schwabe. “HDM- A Model for the Design of Hypertext Applications", Proceedings of Hypertext’91 , ACM Press. pp. 313.

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

[Garzotto94a] F. Garzotto, L. Mainetti and P. Paolini, “Adding Multimedia Collections do the Dexter Model”, Proceedings ECHT’94, Edinburgh, Sept. 1994.

[Garzotto94b] F. Garzotto, P. Paolini and L. Mainetti: “Navigation in Hypermedia Applications: Modeling and Semantics". Journal of Organizational Computing (forthcoming).

[Gilliam94] 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.

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

[Lucarella93] D. Lucarella: “Multimedia Object Retrieval Environment", Proceedings of Hypertext'93, pp. 39-50.

[Odell94] J. Odell: “Six different kinds of compositions", Journal of Object Oriented Programming, Vol. 5 N. 8, 1994.

[Marmann92] M. Marmann and G. Schlargeter. “Towards a better support for hypermedia authoring: The HYDESIGN model", Proceedings of ECHT'92, ACM Press, pp. 232.

[Nanard91] J. Nanard and M. Nanard. “Using Structured Types to Incorporate Knowledge in Hypertext, Proceedings of Hypertext’91 ACM Press. pp. 329.

[Rumbaugh91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W.Lorensen. “Object Oriented Modeling and Design", Prentice Hall Inc. 1991.

[Wirfs90] R. Wirfs-Brock, L. Wiener and R. Wilkerson: “Designing Object-Oriented Software”, Prentice Hall 1990.

[Thüring 91] Thüring, M.; Haake, J.M.; Hanneman, J., “What’s Eliza doing in the Chinese Room? Incoherent hyperdocuments – and how to avoid them”, Proceedings of Hypertext’91 , ACM Press. , pp. 161.

[Winston87] M. Winston, R. Chaffin and D. Herrmann: “A taxonomy of part-whole relationships", Cognitive Science, Vol. 11, pp. 417-444, 1987.

[Wirfs-Brock90] R. Wirfs-Brock, L. Wiener and R. Wilkerson: “Designing Object-Oriented Software", Prentice Hall, 1990.