Multimedia Tools and Applications 7, 213–225 (1998)

Design and Query Strategies to Hypermedia Applications

SILVIA GORDILLO gordillo@info.unlp.edu.ar
LIFIA, Universidad Nacional de La Plata, and Comisión de Investigaciones Científicas, de la Pcia. de Bs. As., CC: 11, La Plata, CP: 1900, Argentina

ALICIA DÍAZ alicia@info.unlp.edu.ar
LIFIA, Universidad Nacional de La Plata, and Consejo Nacional de Investigaciones, Científicas y Técnicas., CC: 11, La Plata, CP: 1900, Argentina

Abstract. In this paper we propose an object-oriented model for designing hypermedia applications. As the object-oriented paradigm allows complex and user-defined types, nonconventional and nonatomic attributes, we can take advantage of these capabilities, not only for information modelling, but also for providing alternative ways for accessing information.

A query language is then presented; it is based on an Object-Oriented Database System query language. It combines features of object-oriented databases queries and primitives for hypermedia navigation. The language offers the possibility of querying both the application-domain information, and allowing the designers to obtain information about the schema of the application.

We present some examples of the use of the object-oriented model and the query language.

Keywords: hypermedia query language, hypermedia design model, hypermedia design, object-oriented database

1. Introduction

Hypermedia applications are characterized by the use of information, organized as a graph (nodes and links) where the user navigates in order to find specific information [8]. Modelling information using the nodes and links model and accessing it by navigation yields very flexible applications. However, the nodes and links model with navigational access is not enough to design and query a hypermedia application [10]. Developing a methodology for hypermedia design has become a very important task because there exists a real need to obtain applications in which concepts like reuse, easy maintenance, modularity, etc. can be used to achieve a good quality level in the final product [20].

Besides, another important problem in this field is the "disorientation" problem. The primary method for accessing information in hypermedia is through navigation. When the underlying network is a small and well-structured web, navigation will be probably enough for accessing it, but this is not always true if the hypertext has a lot of nodes and relationships of different types, or if the user wants to retrieve specific information; in this case he/she must navigate throughout the hypertext web to obtain the information. In both cases, he/she will probably have problems with navigation, because he/she may get lost in the hyperspace. Adding the capability of information querying is a way to solve this problem in order to

additional tools to retrive information about the hipermedia application. Besides, designers could need to access, not only the information stored in the hypermedia application, but also the structural information in any stage of the design process. This is very useful because it allows the designer to understand the design structure of the application he is developing.

There are some proposals to define a model for hypertext design, as in [9, 11, 17, 21] where the authors describe a hypermedia application design methodology, but they do not define how to query a hypermedia application.

There are also some efforts for adding the capability of information querying to hypermedia applications, as an additional possibility to navigational access, [1, 2, 7]. Some of these use relational databases as a way to provide knowledge about the graph structure and node types, but the specific information stored in the system cannot be reached by queries. For example, Parunak [22] combines a hypermedia engine with a relational DBMS, and in this way it represents information via atomic attributes which can be queried. An Object-Oriented Database as the support for hypermedia applications is proposed in [16], in this case, information access is provided using the specific database query language.

In this proposal we define a model for designing hypermedia applications using an Object-Oriented Database System for supporting queries and semantic information. We first describe the main features of our model and the different design levels to build a hypermedia application. In the next sections we describe the query language and discuss future works and conclusions.

2. The methodology

Basically, the model consists of two design levels. The first, High-Level Design, models the main components of the application described in an abstract way by using the well-known concepts of Object-Oriented Modelling like objects, relationships, hierarchies, composition and attributes [19]. Low-Level Design, however, allows designers to describe specific characteristics of hypermedia systems like navigation, perspectives, etc.

2.1. High-level design

The goal of this level is to think about the application domain in a conceptual way obtaining a high-level representation of the application. To develop an application-domain model we use object-oriented concepts and relationships between these objects.

Objects. They are represented in terms of their attributes and behavior [4] and are described by classes called *node classes*. Attribute types are those defined in the most common Object-Oriented Database Systems [3, 12], such as atomic, multivalued and derived attributes plus a special one, multidomain attribute, that can take values in several domains and allows us to visualize the information in different ways according to the perspective we want [9].

Relationships. The relationships between node classes are classes too and therefore they are defined by means of attributes and behavior. Relationships classes are called *link classes* defining two attributes: *from* and *to* representing source and target classes, respectively.

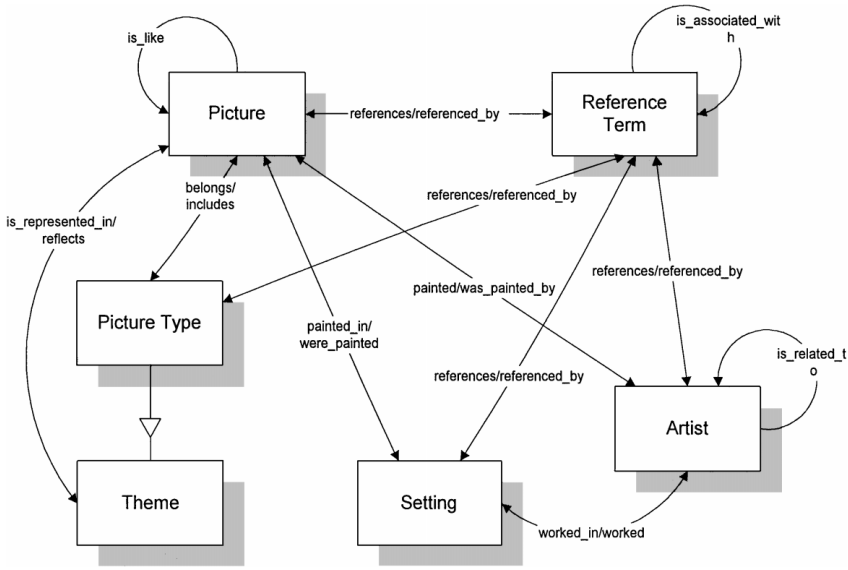

Figure 1. A graphical view of the schema of Art Gallery Application.

From the hypermedia point of view, relationships define links to navigate through the information of the application. In this model they are first class objects and therefore they should be represented in an explicit way. For this reason, in this level we do not allow complex attributes like Object-Oriented Database Systems do, since they define implicit relationships between objects.

Figure 1 shows an Object/Relationship diagram corresponding to the Microsoft Art Gallery CD-ROM.[1] It is a hypermedia application that presents the celebrated art collection of the National Gallery, London. The rectangles showed in this figure, represent the objects such as *Pictures*, *Artists*, and others, and relationships such as *referenced_by* or *was_painted_by* are represented by arrows.

Figure 2 is a representation of some node and link classes. Note that *Artist*'s name is a multidomain attribute which can take values in String and/or Sound domains. Other examples of multidomain attributes are the *to* and *from* attributes of the links classes *referenced_by* and *references*, respectively, since *Picture*, *Artist*, *Picture Types* and *Setting* have associated *Reference Terms* and vice versa. They represent an interesting case where we can describe a general relationship between a node class and a set of node classes.

Both *node classes* and *link classes* are organized in a *is_a* hierarchy, where *Node* class and *Link* class are the roots classes, respectively. The *is_a* relationship has the same semantic defined in the object-oriented paradigm. This means that a class (called subclass) inherits all attributes, behavior and relationships from another class (its superclass) [4]. In figure 3 the reader can see an example of the relationship *is_a* between the node classes *Picture* and *Detailed Picture*. Therefor, *Detailed Picture* inherits the attributes *name*, *painting generalDescription*, *technicalDescription* and *dateOfCreation*, and the relationships *is_like*,

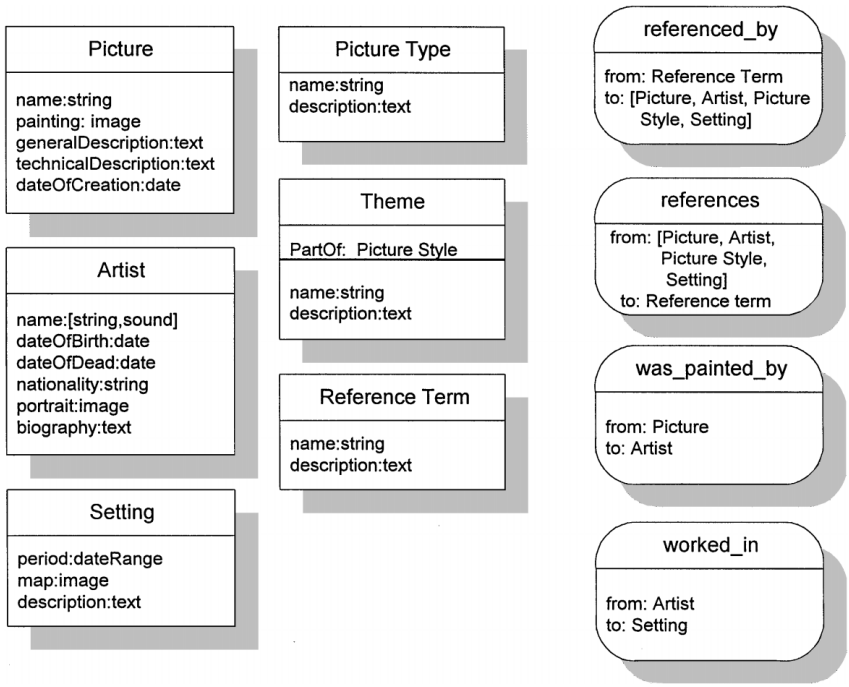

Figure 2. Squared boxes represents some node classes and their attributes and rounded boxes represent some relationships of the Art Gallery Application.

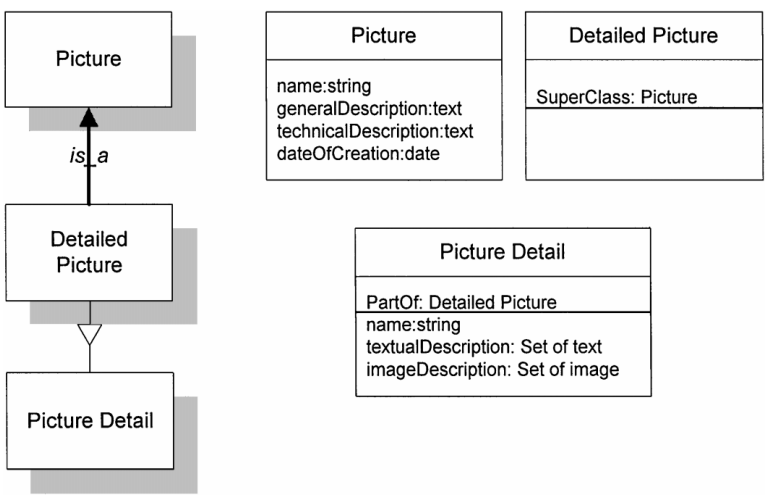

Figure 3. There is an *is_a* relationship between the node classes *Picture* and *Detailed Picture* and the class node *Picture Detail* represents the parts of a *Detailed Picture.*

references/*referenced_by*, *belongs/includes*, *painted/was_painted_by*, etc. from *Picture* node class.

A special relationship exists to design composite objects which is called the *is_part_of* relationship. The composition semantic allows us to define and handle the whole object (including its conceptual parts) or one specific component. From a hypermedia point of view, composite objects define a navigational context where the readers navigate inside the same object. In the above example the relationship between the node classes *Picture Types* and *Theme* is an *is_part_of* one. A *Picture Detail* is part of *Detailed Picture*, so it has its own attributes and relationships and it knows all the attributes and relationships defined in its container. Notice that attributes *textualDescription* and *imageDescription* of the node class *Picture Detail* are two examples of multivalued attributes.

The result of this stage is not very different from those models constructed by Object-Oriented Database Systems designers [12, 13]. In fact, the elements for modelling the application are the same as those used in any object-oriented model. It is a schema composed of a set of node classes capturing the information to be manipulated in the hypermedia application, and the link classes representing the relationships among these node classes. The next level, Low-Level Design, really makes the difference.

2.2. Low-level design

Hypermedia applications have additional features that make them different from database applications and these features are not reflected in the previous level. During Low-Level Design the intent is to define these facilities. In this way we complete the model by incorporating three aspects that are concerned with:

- *navigational structure*: to maintain navigation as the primary method to gain access to the application information;
- *perspectives*: many times it is also necessary to see objects from different perspectives depending on a particular point of view;
- *users interfaces*: different interfaces to visualize different perspectives for the same object can be defined;

To implement these characteristics we define two design concepts: Exemplars and Templates.

Exemplars. They establish different layouts for the information described in the same node [14] done in an abstract way. So while node classes represent the conceptual information of the application specifying the objects in the hyperbase, exemplars are used to navigate because they provide the possible paths from a node and the information that will be visualized.

An exemplar is defined by an attribute list, a behavior list and a set of anchors. *Attributes* in the list are selected from those existing in the associated node class, (this list will contain those attributes defined in the node class which are of interest for this particular exemplar) and in addition, the expected *behavior* list is selected too.

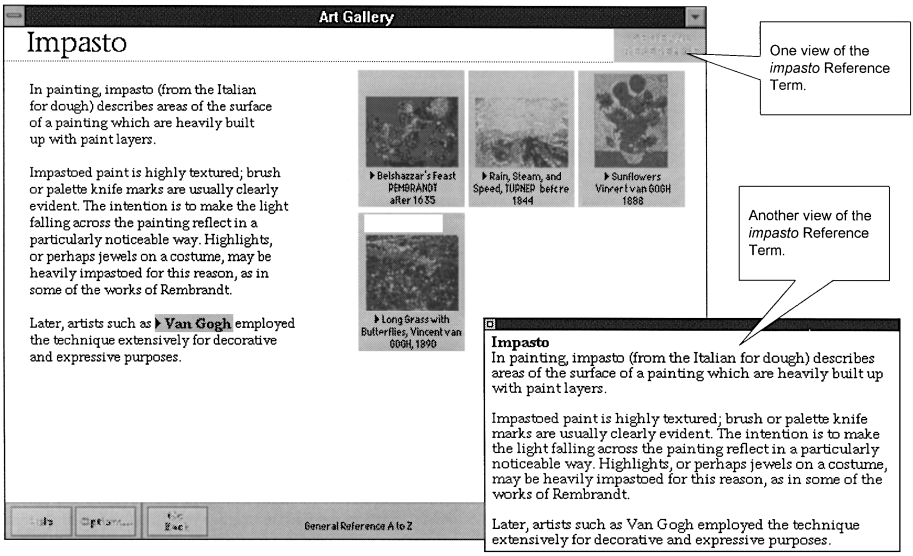

Figure 4. Two perspectives for the instance *Impasto* of the class node *Reference Term*.

When the designer designs the navigation she/he has to embed the link origin inside nodes. Then each exemplar has the *anchors* selected to be shown in a perspective. The anchor is the visualization of a link origin inside the node where the node class of each exemplar is the domain of the *to* attribute. By defining many exemplars for the same node class we can obtain different perspectives of the same information. Each node class always has at least one exemplar associated by default. So each exemplar constitutes one perspective of the information where the designer decides what information is relevant, and what anchors define navigational structure. In figure 4, two snapshots representing two different exemplars for the instance *Impasto* of the node class *Reference Term* are shown. In figure 5, the reader finds the description of these two exemplars *Reference Term* 1 and *Reference Term* 2.

Multidomain attibutes also allow us to define different points of view for the same information. Designers have to select which domains will be used in a particular perspective and for each selected domain, they have to define one visual object as an atomic attribute which takes values in this domain. As an example, in figure 2, *Artist* node class has a multidomain attribute, *name*, whose domains are String and Sound. In figure 5, this multidomain attribute has been solved by splitting it in two attributes *name*1 and *name*2. Notice that it is not necessary to involve all domains in the exemplar. The designer could have selected only one domain to show the multidomain attribute.

Moreover, each exemplar could add new relationships to model exceptional relationships between exemplars. These kinds of relationships are modeled by link classes where *to* and *from* attributes have domains in exemplar classes. A particular case of this is the *More Detail* relationship between the exemplar *Reference Term* 1 and *Reference Term* 2, showed in figure 5, where this relationship allows us to navigate among different perspectives of the same node.

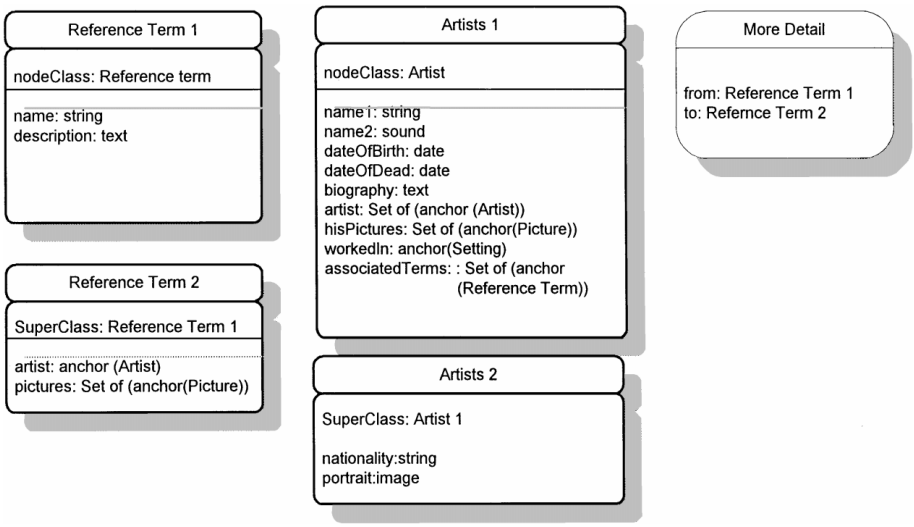

Figure 5. Some exemplars for the node classes *Reference Term* and *Artist*. *More Detail* is a relationship between exemplars.

Like node and link classes, exemplars are organized in a specialization/generalization hierarchy where the root class is the *Exemplar* class. Figure 5 shows the exemplar *Artists* 1 as one perspective of the node class *Artist* where some attributes and anchors are defined. *Artists* 2 is a subclass of *Artists* 1, adding two new attributes.

Templates. They define "how" an exemplar will be perceived while Exemplars describe in an abstract way "what" information will be manipulated. In this way, templates allow us to separate the abstract information from its graphical representation. Each exemplar has one associated template describing its interface and for each element in this exemplar (attributes and anchors) one graphical element in the corresponding template is defined. Template definition is a task where it is necessary to use human-computer interaction techniques and it is out of the scope of our software engineering point of view.

3. The query language

The existence of an application schema lets us increase the model's capabilities by defining a query language allowing access to specific information when navigation is either unnecessary or it becomes a difficult process. For example, when the user wants to retrieve pictures that were painted by *van Gogh*, is better to use a query language than navigation.

We describe a query language which allows us to retrieve two kinds of information:

- one is information about the schema and it is oriented to help designers in the design process, and

	operation	operator
Querying Information	selection	*select* *
	projection	*select* *
	quantifiers	*each, exists* *
	comparators	=, ==, <, >, <=, >= *
	functions	*isPartOf, relatedBy, path*
Querying the scheme		*hierarchy*
		properties
		source, target
		examplar
		relatedTo, relatedFrom

Figure 6. The query language operators.

- the other one is information from hyperbase and it is useful for users of the hypermedia application.

Query language is based on the model defined in [12] and some changes and additions have been made to work with hypermedia applications. In particular, from the Kim's query language we take the *select* operator, *each* and *exits* quantifiers and comparison operators. But we add some special functions for retrieving information about relationships among objects and scheme information. In figure 6, the reader can see a table showing the operators of our query language. The tables are organized in two parts, one showing the operators to query the application information and the other showing the operators to query the application schema. Asterisks indicate those operators took from Kim's query language.

3.1. Querying information

The essential operation for querying information is the selection operator and it is similar to selection in Object-Oriented Database Systems.

Selection. The selection of objects from one or more classes, when a condition holds, is similar to selection in relational databases.

```
select [targetClause]
    from rangeClauses
    in [source]
    [where qualificationClause]
```

where *targetClause* is the specification of classes to be output; *rangeClause* indicates the binding of variables to corresponding sets of instances of classes; *source* is the hypertext or set of objects that will be queried; *qualificationClause* is a Boolean combination of predicates; and [. . .] indicates optional parameters.

The query example given below retrieves the sub-hypertext with pictures and their artists where pictures were painted before 1870.

```
Q := select Picture, Artist
       from Picture :p
       where (p.dateOfCreation < 1870)
```

Assign. By means of the assign operation we are able to store the results of queries in variables. This capability allows one to perform additional queries, by using these results as the source of information.

The result of a query is a hypertext made up of instances of the node classes in the *targetClause* plus the relationships among them. In other words, the result is another hypertext created out of the source hypertext but having only those instances satisfying the *where* clause and their relationships. So it can be assigned to a variable (i.e., Q), in order to obtain a new navigational context to query and navigate. Here Q is a view of the source hypermedia application made of node classes *Picture* and *Artists*, link classes *painted* and *was_painted_by*, and the instances of these classes satisfying the whereClause $p.dateOfCreation < 1870$.

Quantifiers. The qualification clause can include multivalued attributes. For this reason, the model includes two quantifiers: EACH and EXIST, corresponding to the universal and existential quantifiers, respectively.

The example below retrieves all *Picture Details* instances whose *descriptionText* is aText

```
Q := select Picture Detail
  from Picture Detail : pd
  where exist pd.textDescription = aText
```

IsPartOf. This operator allows to retrieve parts of an instance. For example, the query below retrieves those *Picture Type* instances which have a *Theme* called "*flowers*".

```
Q := select Picture Type
  from Picture Type : ps, Theme :t
  where t isPartOf ps and t.name = "flowers"
```

Projection. The projection operator allows the retrieval of some attributes and methods from objects belonging to one or more classes. This operation is similar to the project operation of relational databases systems.

```
select [targetClause]
     from rangeClauses in [source]
     [where qualificationClause]
```

where the definition of targetClause, rangeClause, source, qualificationClause are the same as the select operation.

The answer to a project query always is a set of lists of specified attributes.

In the next example, the result is a set of lists containing pairs representing the name of the Artists and the name of the Pictures of those Pictures painted before 1870.

```
Q := select Artist.name, Picture.name
     from Picture: p
     where (p.dateOf Creation < 1870)
```

Related_by. Retrieving objects where a condition holds is very useful in hypermedia applications. The Related_by operation allows one to establish if there is a relationship between two node objects. The result of this operation is a Boolean value, it is true if the relationship exists, and false otherwise.

The syntax of this query is:

```
related_by (fromObject, relationshipName, toObject)
```

The example shows how to retrieve the *Artists* of *Pictures* whose *Picture Types* is "*portraits*".

```
Q := select Artist
     from Artist :a, Picture :p, Picture Types :ps
     where (ps.name = "portraits"
               and related_by (p, WasPaintedBy, a))
```

Path. The Path operation allows one to establish if there is a path of relationships among node objects. The result of this operation is a Boolean value, it is true if the path exists, and false otherwise.

The syntax of this query is:

```
path (fromObject, relationshipsList, toObject)
```

where *relationshipsList* is a list of link classes.

The following example retrieves *settings* where worked those *artists* who painted *pictures* whose *pictureType* is "*portrait*" and these *pictures*.

```
Q := select Setting
     from Artist :a, Picture :p, Picture Types :ps, Setting: s
     where (ps.name = "portraits"
               and path(p, was_painted_by, worked_in, a))
```

3.2. Querying the schema

This kind of queries is useful for the designer; it allows her/him to obtain information about any of the hypertext schema hierarchies.

Hierarchy. This operation is used to establish which are the superclasses or subclasses of a given class.

hierarchy↑ *className* From *Hierachy_Type*

or

hierarchy↓ *className* From *Hierachy_Type*

where *className* is a class name of any hierarchy and *hierarchy_type* is one of *Node*, *Link* or *Exemplar* hierarchies.

The difference between *hierarchy*↑ and *hierarchy*↓ is that the first retrieves *className*'s superclasses and second retrieves its subclasses.

Properties. Retrieves the properties defined in a class.

properties *className* from *hierarchy_type*

className and *hierarchy_type* are defined as in *hierarchy* operation.

Source and target. Retrieve the source and the target node class of specific relationships, respectively.

source *className* target *className*

where *className* is the class name in the *Link* hierarchy.

Exemplar. This query retrieves the names of exemplars which are associated with a specific node class.

exemplar *className*

where *className* is node class name.

Related_to and related_from. The *related_to* operation determines what node classes are reachable from a specific class, and *related_from* operation defines all node classes from which the specific class can be reachable.

related_to *nodeClass*
related_from *nodeClass*

where *nodeClass* is a node class name.

4. Conclusion and future work

We have presented a model for hypermedia design, which provides the base for structuring information through nodes, relationships and exemplars. Using it we can model the application in different abstraction levels, like modern software engineering methods do, without loosing the traditional hypermedia facilities. Maintaining applications designed with this model is easier than the traditional approach because there is a better understanding about the application domain.

Having defined a schema, a query language definition adds the necessary characteristics for manipulating hypermedia applications, and provides additional tools for retrieving information, avoiding navigational access when specific information is required. Also this query language is useful to designers because they can query the schema, improving their knowledge about the application domain.

Now we are completing the query language defining additional operations, and testing the model using different applications.

We are beginning to implement the model in an Object-Oriented Hypermedia Framework [18]. This framework allows modelling a schema and its instances and it is possible to navigate the schema and the application information. We will extend the present work to be able to query information based on our language.

Note

1. Art Gallery is a trademark of Microsoft Corporation.

References

1. B. Amann and M. Scholl, "Gram: A graph data model and query language," in Proceedings of the 4th ACM Conference on Hypertext and Hypermedia (ECHT'92), Milano, Italy, 1992, pp. 201–211.
2. B. Amann and M. Scholl, "Querying typed hypertexts in Multicard/O2," in Proceedings of the 5th ACM Conference on Hypertext and Hypermedia (ECHT'94), 1994, Edinburgh, Scotland.
3. E. Bertino and L. Martino, Object Oriented Database Systems: Concepts and Architecture, Addison Wesley: Reading, MA, 1993.
4. G. Booch, Object Oriented Analysis and Design with Applications, Addison Wesley: Reading, MA, 1994.
5. V. Chistiphides and A. Rizk, "Querying structured documents with hypertext links using OODBMS," in Proceedings of the 5th ACM Conference on Hypertext and Hypermedia (ECHT'94), Edinburgh, Scotland, 1994.
6. D. Chorafas and H. Steinmann, Object Oriented Databases, PTR Prentice Hall, 1993.
7. M.P. Concens and A. Mendelzon, "GaphLog: A visual formalism for real life recursion," in Proceedings of the ACM SIGACT, Nashville, TN, 1990.
8. J. Conklin, "Hypertext: An introduction and survey," IEEE Computer, Vol. 20, No. 9, pp. 17–41, 1987.
9. F. Garzotto, P. Paolini, and D. Schwabe, "HDM—A model based approach to hypermedia applications design," ACM Trans. Info. Syst., Vol. 11, No. 8, pp. 1–26, 1993.
10. F. Halasz, "Reflections on NoteCards: Seven issues for the next generation hypermedia systems," Communications of the ACM, Vol. 31, No. 7, pp. 836–852, 1988.
11. T. Isakowitz, E.A. Stohr, and P. Balasubramaninan, "RMM: A methodology for structured hypermedia design," Communication of ACM, Vol. 38, No. 8, pp. 34–44, 1995.
12. W. Kim, Introduction to Object Oriented Databases, MIT Press: Cambridge, MA, 1990.

13. W. Kim, Modern Database Systems, ACM Press: New York, NY, 1995.
14. W. LaLonde, "Designing families of data types using exemplars," ACM Toplas, Vol. 11, No. 2, pp. 212–248, 1989.
15. M. Loomis, Object Oriented Databases. The Essentials, Addison Wesley, Reading, MA, 1994.
16. M. Marmann and G. Schlargeter, "Towards a better support for hypermedia authoring: The HYDESIGN model," in Proceedings of the 4th ACM Conference on Hypertext and Hypermedia (ECHT'92), Milano, Italy, 1992, pp. 232–241.
17. J. Nanard and M. Nanard, "Using structured types to incorporate knowledge in hypertext," in Proceedings of Hypertext'91, San Antonio, TX, 1991, pp. 329–343.
18. G. Rossi and A. Garrido, "Extended object oriented applications with hypermedia functionalities," in Workshop on Hypertext Functionalities, ECHT'94, Edinburgh, Scotland, 1994
19. J.J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, Object Oriented Modeling and Design, Prentice Hall Inc.: New York, NY, 1991.
20. D. Schwabe and G. Rossi, "An object oriented hypermedia design model," Communication of ACM, Vol. 38, No. 8, pp. 45–46, 1995.
21. D. Schwabe, G. Rossi, and S. Barbosa, "Systematic hypermedia design with OOHDM," in Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences, Maui, HI, 1995.
22. H. Van Dyke Parunak, "Don't link me in: Set based hypermedia for taxonomic reasoning," in Proceedings of Hypertext'91, San Antonio, TX, 1991, pp. 233–242.

Silvia Gordillo is a full Professor at Universidad Nacional de La Plata inLa Plata, Argentina. She is a member of LIFIA (Laboratorio de Investigación y Formación en Informática Avanzada). Her areas of work include Object-Oriented Databases and Geographic Information Systems. Silvia has presented results of research projects at many Computer Science Conferences. She is a Visiting Professor at Universidad Nacional de Rio Cuarto. She is a member of SADIO (Sociedad Argentina de Informatica e Investigacion Operativa).

Alicia Díaz is an Associate Professor at Universidad Nacional de La Plata in La Plata, Argentina. She is memebr of the LIFIA (Laboratorio de Investigación y Formación en Informática Avanzada). Her areas of work include Hypermedia Systems and Application, specializing in Hypermedia Design. Alicia has presented results of research projects at many Computer Sciences Conferences. She is a Visiting Professor at Universidad Nacional de Rio Cuarto.