AN OBJECT-ORIENTED DATA MODEL FOR HYPERMEDIA SYSTEMS

M. Hatzopoulos
D. Gouscos
M. Spiliopoulou
C. Vassilakis
M. Vazirgiannis

Department of Informatics
UNIVERSITY OF ATHENS
Panepistimiopolis, Ktiria TYPA
GR-157 71, Ilisia
Athens, GREECE

1.INTRODUCTION [#]

Hypermedia [1] is a fast growing field of research interest because of the new aspects introduced as far as it concerns the interaction between the machine and the user. The innovative features mainly refer to

- the presence of various information types (multimedia environment)
- the mapping of the semantic connections between information items into a flexible, efficient representation.

This feature enables navigation through the information network in a meaningful way. Our effort concentrated on the definition and partial implementation of a flexible, general-purpose model through which the full capabilities of object-oriented programming can be exploited.

2.HYPERMEDIA SYSTEMS PHILOSOPHY AND FUNDAMENTAL CONCEPTS

Hypermedia systems have appeared as an evolution of traditional databases and aspire to prove much more flexible and efficient for information storage and retrieval. The key difference between a hypermedia system and a traditional database system is not the broader spectrum of information that the former can accommodate but rather the fact that traditional database systems contain information that is structured, while hypermedia systems contain interconnected information.

A hypermedia system must serve two major tasks :

1) like a traditional DBMS, it must render the whole or an appropriate part of the stored information ready to be examined and processed by end users [2], and
2) unlike a traditional DBMS, it must render the whole or an appropriate part of the connections between the stored information ready to be examined and processed by end users [2].

A design metaphor which seems to grant such an equilibrium for concepts and relationships between concepts is a graph. A graph consists of nodes and arcs. From this very description it is clear that arcs and nodes are treated as entities of equal importance. Arcs are not parts of nodes, exactly as nodes are not parts of arcs. Such a scheme seems suitable to model the structure of a hypermedia system.

In order to build a more efficient metaphor it is better to talk about a network instead of a graph, about items instead of nodes and about links instead of arcs. In this way, we come to the

[#] This work is accomplished within the DELTA project SAFE, task HYPERATE, by the team of the University of Athens.

following definition of a design model for a hypermedia system :

A hypermedia system is an information network. This network consists of items and links. An item is an entity that contains some hyp ermedia data. A link is an entity that contains some information about a logical connection between two items. Each such connection is implemented s a link.

3. HYPERMEDIA SYSTEMS, CHARACTERISTICS AND REQUIREMENTS

The information stored in a hypermedia system can not be uniformly described. There is information coming from different media. There is information on which no structure can be imposed and there is information with a varying degree of structuring. Therefore items, which are the entities that store information, must be implemented in such a way that as few characteristics as possible are predefined and as many characteristics as possible are left to be defined for specific applications.

Connections between hypermedia information contain a large amount of semantics [3]. Therefore, they must be implemented in a way that supports the declaration of semantics. Moreover it is quite important that connections must be represented at least on the same level of functionality as the information itself.

There are often multiple connections between the same information, which denote different semantics. Therefore, the implementation must support multiple connections between the same information. Furthermore, connections between information often form hierarchical structures. So, such hierarchies must be supported as naturally as possible.

Finally, hypermedia systems' designers must often define information structures and connections that resemble already defined structures and connections. Thus, the adopted design must support property inheritance.

So, any design for a hypermedia network of items and links must support the following features:

(1) generic modelling constructs for items
(2) declaration of semantics for links
(3) augmented functionality for links
(4) multiple linking of items
(5) item hierarchies
(6) property inheritance

4. COMPARISON OF DESIGN METHODOLOGIES AGAINST HYPERMEDIA SYSTEMS REQUIREMENTS

[4] describes a number of requirements that must be fullfilled by a database retaining multimedia information. These are:

Requirement 1 : Generic modelling constructs for items

The relational model employs a unique modelling construct, the relation. Therefore, in a relational design, items would be tuples in relations and thus would have a predefined structure, which would most certainly prove unsuitable for the majority of applications. Furthermore, as interface procedures must be defined once for all relations, all relations would have the same interface.

An object-oriented model employs a generic modelling construct, the class [5]. Class constructs are completely unrestricted and if items are designed as objects of classes, their structure can be defined according to the requirements of each specific application. Furthermore, as interface methods can be defined separately for each class, different classes can have different interfaces.

It is important to note that an object-oriented model can easily simulate a relational one, by defining a Relation class, behaving exactly as the standard, relational model relations.

Requirement 2 : Declaration of semantics for links

Requirement 3 : Augmented functionality for links

In a relational model, links must be implemented implicitly through relations. In this way, they are treated similarly to the actual information (also retained in the form of relations) but on the cost of increasing complexity, especially in an environment where a large number of links must be implemented.

In an object-oriented model, links can be implemented explicitly as separate objects holding some information of their own. Therefore they can be assigned as many semantics as desirable and their functionality is equal to the functionality of items.

Requirement 4 : Multiple linking of items

The relational model theoretically supports the connection of the same data through different relations, but these multiple connections can be implemented either by copying data, which results in data redundancy, or by making excessive use of primary keys.

In an object-oriented model however, the same items can be connected with multiple links (denoting different semantics for example) without any data redundancy involved.

This happens because in the relational model all data exist in the database as parts of one-to-one relationships.

For instance, if datum A relates to datum B and must also relate to datum C, then datum A must

be copied into the respective relation. On the other hand, in an object-oriented model data can exist in the database as isolated entities without relating to each other ; therefore if datum A relates to datum B and must now also relate to datum C then a new link can simply be added to datum A itself and not to a copy of it.

Requirement 5 : Item hierarchies

In a relational model, complex hierarchical connections of items must be flattened and stored across many relations. As a result:

1) to retrieve such a hierarchy in its entirety, a number of queries is necessary: these queries are executed sequentially and as their sequence is not known in advance, it can not be optimised by the DBMS
2) once hierarchies are flattened to fit in relations, they cannot easily be reconstructed from relations and presented to end users.

In an object-oriented model, complex hierarchical connections can be stored in their form, without any flattening. As a result:

1) each such hierarchy can be accessed with only one query that retrieves the root item using internal message passing to retrieve the rest
2) as hierarchies are not decomposed before storage, there is no need either for reconstruction before presentation; hierarchies are retrieved in their original form and they are ready for presentation to users.

Requirement 6 : Property inheritance

It is evident that property inheritance is supported only by an object-oriented design [5].

5. OBJECT-ORIENTED DESIGN FOR HYPERMEDIA SYSTEMS

5.1. CLASS HIERARCHY DESCRIPTION

We are going to define a data model supporting items and links as already analyzed. Since we describe an object-oriented data model, it is clear that all entities within the data model are treated as objects and must belong to some subclass of a class Object. Therefore, a generic class Object is defined as the root of the entire class hierarchy.

It is also clear that the hypermedia system will contain objects behaving as items and objects behaving as links. As already stated, items and links play different but equally important roles, and as a consequence they must be assigned different but analogous functionality [3].

In an object-oriented data model, objects with analogous functionality generally belong to classes

on the same level of the class hierarchy. Therefore in our case items and links must belong to subclasses of class Object located on the same level of the class hierarchy. As items store information and links store connections, we define two subhierarchies of classes, one for items and another for links.

5.1.1. Subhierarchy of information classes

We define a class Item, as an immediate subclass of Object. Class Item is the root of the entire hierarchy of information classes. Items (i.e. objects of class Item) contain hypermedia data such as text, graphics, images, sound, video and probably more. It must be decided whether the same item should be allowed to contain data of different type or of the same type only:

From users' point of view, an image and some text for example may form a logical entity and must be retrieved together. From designers' and software developers' point of view, data of different types must be processed in a different way and therefore, they may not be stored on the same item.

We have chosen to satisfy designers' needs. This choice :

1) allows for independent development of methods particularly applicable to items containing information of certain type
2) does not degrade system performance, as items containing logically relevant but multimedia information can still be linked tight enough to attract users' attention.

As a consequence we have decided to define subclasses of class Item like Text, Graphics, Image, Sound, Video etc. Each subclass is provided with the corresponding Content and methods for Content presentation and modification. New subclasses of class Item can be defined to support new information types.

5.1.2. Subhierarchy of connection classes

Since links are the only entities in the hypermedia system that retain connections between information items, this hierarchy comprises one class only. This fact may create certain doubts concerning the definition of a separate class for links. At the heart of the matter, there is again a users' versus designers' satisfaction trade-off:

From users' point of view, each item "has" some links which emanate from the item and point to other items. Therefore, links should be included in the structure of an item as an additional field, according to the philosophy of traditional database design.

From designers' and software developers' point of view, links and items behave in a completely different way and links often have to be retrieved and processed independently of the items they connect. Therefore, links must belong to a separate class Link.

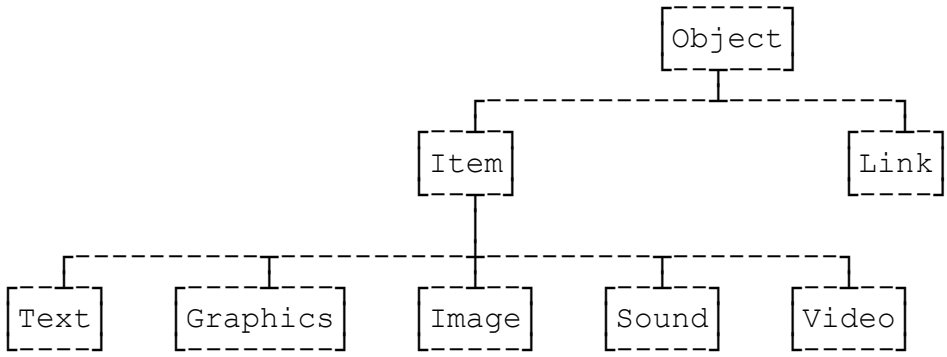
We have adopted the second alternative and we have defined a class Link as an immediate

subclass of class Object for the following reasons:

1) the definition of a separate class Link allows for direct manipulation of links (which would be impossible if links were stored as data variables of items) and for the development of methods particularly applicable to links for link creation, deletion, modification and retrieval
2) the definition of two independent classes for items and links allows to modify or expand each of the two corresponding class hierarchies without any modification to the other one.

As a result of those decisions, we maintain the functionality equilibrium required in the previous sections and define a semantically consistent and very flexible data model.

The class hierarchy defined in our proposal has the structure shown below :



Nodes in this tree structure correspond to classes. Children nodes represent subclasses and parent nodes represent superclasses. Each class inherits all the data variables and the methods defined for all its superclasses up to class Object and has some data variables and methods of its own. Data variables and methods are either inherited without change or inherited and redefined.

5.2. CLASS DEFINITION

In this section we define all the classes already described in the previous section. Each class is defined by defining its data variables and its methods. Each method is provided with a set of parameters which are mainly selected to assure design clarity. It is not definite yet whether the parameters defined here will be the actual parameters of the corresponding methods or not. This remains to be decided during implementation.

Each class inherits all the data variables and the methods defined for all its superclasses up to class Object and has some data variables and methods of its own. Data variables and methods are either inherited without change or inherited and redefined.

5.2.1. Class Object definition

Class Object is an abstract class [6], i.e. it has no instances. Furthermore, no data variables or

class methods are defined for class Object. This class serves only as the root of the class hierarchy.

5.2.2. Class Item definition

Data variables

Name

Every item has a *Name*, i.e. a user-specified string which serves as an identifier for the item and is unique for the entire system. The *Name* of an item also denotes the type of information that the item contains (text, graphics, images etc.).

Title

Users must be able to get a quick feeling of the content of an item. Therefore, each item has a *Title*, which is a short textual description of its content.

Keywords

It seems desirable to retain some relational features in order to query a hypermedia database. It has also been observed that even hypermedia systems do not totally suppress the need for relational queries. If a system is going to support queries aiming to items with specific content, then the content of all items must somehow be formally indicated.

So, each item is provided with a set of *Keywords*. *Keywords* are words or phrases denoting important concepts found in the content of an item. In effect *Keywords* will most probably be nouns, e.g. 'mathematics', or noun phrases, e.g. 'economic european community'. The keywords of an item do not have to be chosen from its content, which permits the definition of Keywords for items with non-textual Content too.

LinkIdentifiers

Items relate to each other by links and therefore an item must be related with the links that emanate from it. However, links are objects of a different class and as such they are separately stored, retrieved and processed, so an item must not contain links (this would be as unnatural as if we said that a link must contain items), but pointers to links instead.

That is why we have decided to provide each item with a set of *LinkIdentifiers*, i.e. identifiers of links that emanate from the item and connect it to other items (see below for the data content of links). As each link has a unique identifier, the LinkIdentifiers of an item can serve as pointers to the corresponding links. In this way we do not have to use physical addresses of links, which would create certain updating problems.

To summarise, each item contains the following data :

- *Name* (unique identifier for the item)
- *Title* (short-hand textual description of content)
- *Keywords* (set of important concepts found in the content)

- *LinkIdentifiers* (set of identifiers of emanating links)

Methods

First of all, we must define methods to process items in the traditional way that database entities are processed, i.e. methods to create, retrieve, save and delete items. We consider the following methods:

Create(Name,Title,Keywords,LinkIdentifiers)

This method creates an item containing the values passed as parameters. It does not save the item to mass storage. It does not present the item.

Retrieve(Name)

This method fetches item information from mass storage to a memory buffer and assigns it to a new Item-object's variables.

Save()

stores an item from a memory buffer back to mass storage and makes any changes permanent.

Delete()

deletes an item from mass storage. It does not delete the links that emanate from or point to the item. The method deleting these links must be called at transaction level. The updates on the link network are discussed further on.

Note that items are uniquely identified by their *Names*. Also note that the method for item creation requires values for all data variables to be passed as parameters. When a user creates an item, the only value that he/she must be obliged to provide is the item's *Name*. All other variables may or may not be assigned values (if no values are assigned there are default values that can be used). This, however, is a matter of user interface. When the *Create* method is invoked all data variables have obtained values, either user-specified or default, which are passed to the method as parameters and are stored to the created item.

The name of an item may be separately retrieved by the method

GetName()

to return the value of the *Name* variable of an item that has been fetched into a memory buffer.

The title of an item may be updated. We consider the method

GetTitle()

to return the value of the *Title* variable of an item that has been fetched into a memory buffer, and the method

SetTitle(NewTitle)

to set this variable to value NewTitle.

Keywords may be added to or deleted from an item. We consider the method

GetKeywords()

to return the value of the *Keywords* list-variable of an item that has been fetched into a memory buffer. Then the method

AddKeyword(Keyword)

is used to add a new keyword to the *Keywords* variable, while the method

DropKeyword(Keyword)

is used to delete a keyword from the *Keywords* variable.

Keyword modifications can be implemented as successive deletions and additions.

These methods do not affect the *LinkIdentifiers* of an item. Furthermore, when a link is created or deleted then its identifier must be added to or dropped from the *LinkIdentifiers* of the source item respectively. We consider the method

GetLinkIdentifiers()

to return the value of the *LinkIdentifiers* variable of an item that has been fetched into a memory buffer. Then the method

AddLinkId(LinkId)

is used to add a new LinkId to the *LinkIdentifiers* variable, while the method

DropLinkId(LinkId)

is used to delete a LinkId from the *LinkIdentifiers* list-variable.

5.2.3. Definition of subclasses of class Item

One subclass of class Item is defined for every different physical type of information that the system supports. We propose the subclasses Text, Graphics, Image, Sound and Video, but information of different type can always be supported by new subclasses.

The additional data variables of classes Text, Graphics, Image, Sound and Video is exactly their *Content*, i.e. the actual information that the objects of those classes hold. The content of each class is defined in a different way, i.e. the content of class Text is defined as a sequence of characters, the content of class Graphics is defined as a set of graphics primitives. This is why we can not define a generic *Content* variable for class Item.

For every subclass of class Item we need new methods for presentation and content modification. We consider the method

Present()

to present on an output device an item that has been fetched into a memory buffer.

The method

GetContent()

returns the value of the *Content* variable of an item that has been fetched into a memory buffer. Then the method

SetContent(NewContent)

is used to set the *Content* variable to value NewContent. Content editing is left to be dealt with during the user interface development.

The content of subclasses Text, Graphics, Image, Sound and Video is not structured. In cases where the imposition of structure is desirable and feasible, designers can define structured subclasses and polymorphically redefine inherited methods.

5.2.4. Class Link definition

Data variables

The definition of data variables for links is the most crucial point of the entire design, as it influences the ability of the system to

1) support navigation,
2) hold semantic information about relations and
3) perform efficiently

We believe that our choices satisfy all three requirements.

LinkId

Every link has a LinkId, i.e. a system-specified identifier that is unique for the entire system. LinkIds are positive integers assigned sequentially to links by order of creation. LinkIds of deleted links are stored and used again when new links are created.

Source, Target

As links are stored, processed and possibly queried independently from items, each link identifies both its source and its target item by storing their names in variables *Source*, *Target* respectively. Using this information links can be very easily traversed either forwards or backwards. Traversing a link forwards means retrieving its target item. Traversing a link backwards means retrieving its source item.

Anchor

Links relate items to other items, but as some links express relations defined very precisely it is not practical to have them emanating from entire items but rather specific regions within the content of items. For example, a link may start from a word or phrase in a piece of text, a frame or sequence of frames in a video sequence etc. Therefore, we have chosen to provide links with **anchors**. The notion of the anchor of a link is quite popular in the hypermedia literature [7, 8].

An anchor consists of a list of numbers fully defining the departure region of a link within the content of an item. If a link is not anchored, the list is empty. The anchors of links departing from content of various media may be defined as follows :

1) The departure region of a link in a piece of text may be a phrase, in which case the anchor may contain the position of its first character within the text and its size in characters.
2) The departure region of a link in a structure of graphics may be an area of graphics, in which

case the anchor contains a set of corresponding identifiers.

3) The departure region of a link in an image may be a rectangle, in which case the anchor may contain the coordinates of a vertex and the dimensions of two non-parallel sides.
4) The departure region of a link in a sound item may be a sequence of whatever units the sonar data are divided in, in which case the anchor may contain the position of the first unit within the sequence and the number of participating units.
5) The departure region of a link in a video item may be a sequence of frames, in which case the anchor may contain the position of the first frame within the sequence and the number of participating frames.

Type, Weight

Saying that two information items relate to each other is not sufficient, if the semantics of the relationship are not indicated. The study of [9] about graph modelling stresses the importance of typed and weighted connections between hypermedia information. [10] also mentions the necessity of typed and anchored relationships among objects.

We have decided to provide each link with a Type, indicating the semantics of the corresponding relation. This Type can either be selected from a predefined menu or defined freely, in which case users are responsible to maintain some practical and meaningful typing conventions.

Furthermore, each link has a (normalised) Weight. Weight indicates how strongly related the source and target items are, with respect to the type of relationship denoted by the link. Users are again responsible to maintain some practical and meaningful weighting conventions.

In brief, each link contains the following data :

- LinkId (unique identifier for the link)
- Source (name of source item)
- Target (name of target item)
- Anchor (anchor of the link within the content of the source item)
- Type (indication of the semantics of the relationship)
- Weight (indication of how strong the relationship is)

Methods

We must first of all define methods to process links in the traditional way that database entities are processed, i.e. methods to create, retrieve, save and delete links. We propose the following methods, bearing in mind that links are uniquely identified by their LinkIds.

Create (Source,Target,Anchor,Type,Weight)

This method creates in a memory buffer a link, assigns to it the first available LinkId and stores the parameter values in the corresponding variables. It does not save the link to mass storage and it does not update *LinkIdentifiers* of source item.

Retrieve (LinkId)

This method fetches the link from mass storage to a memory buffer.

Save()

stores a link from a memory buffer to mass storage and makes any changes permanent.

Delete ()

deletes a link. It does not update the LinkIdentifiers of the source item. This has to be handled at transaction level.

Note that the method for link creation requires values for all data variables except LinkId to be passed as parameters. LinkId is specified by the system during link creation. When a user creates a link, he/she must be obliged by the User Interface to provide at least the link's Source, Target and Type. Anchor and Weight may be assigned either user-specified or default values. In this way, users can not create dangling links with undefined source or target items or of undefined type, and once a link is created it is ready for full processing.

The LinkId of a link may be separately retrieved. We consider the method:

GetLinkId()

to return the value of the LinkId variable of a link that has been fetched into a memory buffer.

All other data variables of a link may be modified. We consider the methods :

GetSource()

GetTarget()

GetAnchor()

GetType()

GetWeight()

to return the values of the *Source, Target, Anchor, Type* and *Weight* variables respectively, for a link that has been fetched into a memory buffer.

Then the methods:

SetSource(NewSource)

SetTarget(NewTarget)

SetAnchor(NewAnchor)

SetType(NewType)

SetWeight(NewWeight)

can be used to set the corresponding variables to the new values NewSource, NewTarget, NewAnchor, NewType and NewWeight respectively.

When the source item of a link is changed, the higher level procedure that invokes the SetSource method should take care of modifying the LinkIdentifiers of both previous and new source item.

Editing of anchors is left to be dealt with during the user interface development.

6. CONCLUSION

We have proposed an object-oriented design model for a hypermedia system and we have shown the capabilities of this model for

1) accomodating hypermedia information
2) capturing the semantics of relations between information

We believe that the most important characteristics of our proposal are:

1) the adoption of object-oriented design philosophy, which allows for full exploitation of the capabilities of object- oriented programming, and
2) the representation of connections between information on the same level of functionality as the information itself.

The hypermedia system based on this design will form a flexible and powerful environment for information retrieval.

REFERENCES

[1] F.G.Halasz, "Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems", CACM, Vol.31/7, July 1988, pp.836-852.

[2] G. Crane, "Standards for a Hypermedia Database, Diachronic vs Synchronic Concerns", Hypertext Standardization Workshop, 1990.

[3] F. Halasz, M.Schwarz, "The Dexter Hypertext Model", Hypertext Standardization Workshop, 1990.

[4] K.E.Smith, S.B.Zdonic, "Intermedia: A Case Study of the Differences Between Relational and Object-Oriented Database Systems", OOPSLA'87 Proceedings, 1987.

[5] B. Cox, "Object-oriented Programming: an Evolutionary Approach", Addison-Wesley, 1987.

[6] L.J.Pinson, R.S.Wiener, "An introduction to object-oriented programming and Smalltalk", Addison-Wesley, 1988.

[7] L.Garrett, K.Smith, N.Meyrowitz, "Intermedia: Issues, Strategies and Tactics in the Design of a Hypermedia Document System", Proceedings of the Conference on Computer-Supported Cooperative Work (CSCW'86), Austin, Texas, December 1986, pp.163-174.

[8] R.Akscyn, D.McCracken, E.Yoder, "KMS: A distributed Hypermedia System for Managing Knowledge in Organisations", CACM, Vol.31/7, July 1988, pp.820-835.

[9] P.A.M.Kommers, "HYPERTEXT and Conceptual Mapping", pp.55-70, SAFE/HYP/6/HCI-del/mk/fw/lr/pk, September 1989.

[10] J.Nanard, M.Nanard, H.Richy, "Conceptual Documents: a Mechanism for Specifying Active Views in Hypertext", ACM Conf. on Document Processing Systems, 1988, pp.37-42.