24 MuseumsIndex — An object oriented approach to the design and implementation of a data driven Data Base Management System

Jørgen Feder

24.1 BACKGROUND

The story about MuseumsIndex began in 1985, when it was decided by the Danish museums co–ordinating body, *Statens Museums Nævn*, to set up an advisory group which was given the task of helping museums who wished to enter the confuse and obscure world of computers and their ways of dealing with reality.

After the conclusion of reports on which computers to buy (IBM PC's or compatible) and which word processors to use, the focus was directed to the issue of documentation, where the group was very well aware of the dangers and complications lurking ahead.

The Danish museums are in the issue of documentation like most other museums in the world: diverse, with scarce guidelines in writing of their documentation practice and differences in practice over time. But they have one thing in common: each and every of them are convinced that their way of doing things is the best.

24.1.1 The Case Management System

Before 1985 there had however been various attempts at standardisation, and one of them turned out up to have a major influence on the way MuseumsIndex later became structured: The Case Management System.

The idea behind the Case Management System (CMS), is, that every bit of information in a museum can be referred back to a case — the event that originally caused the information later to enter the museum. This could be a phone call from a man who found a gold bracelet while ploughing, the museum initiating an excavation on a promising piece of land or an investigation on the distribution of computers in museums in the 20th century.

Every time such an event takes place, the museum opens a case. The case is given a number, a short description and an opening date. When objects (if any), photos, maps, letters, personal information, etc. have been collected and meticulously examined and described, then the case can be closed. If more information relevant to the case later emerges, the case is reopened until all relevant information again is collected and documented, after which the case can be closed again. As it can be seen the CMS — beside its documentation element — also has an element of administrative control. The weight of these elements will of course vary from museum to museum.

In the CMS the different objects for documentation (artefacts, photos, maps, etc.) are linked to the case through an ingenious number and symbol system. If a case, for instance, has the number 1901, then the first artefact in the case will be given the number 1901X0001, the first photo will get the number 1901F0001 and so on.

Although only a part of the Danish museums was using the CMS (and among those who used it many were using home brewed variants) it became apparent that it was much easier to reach agreement on the overall structure following the guidelines of CMS, than when it came to more subtle details like dating, material and terms as such. Furthermore it became evident — as many in the group had little or no experience at all with computers — that there was an urgent need to demonstrate a model, which could be used as a subject for further discussions.

24.1.2 Prototyping in HyperText

At that time I was rather occupied by the concept of HyperText as outlined by Ted Nelson in the Xanadu–project, an ambitious multi media

project for Sun work stations. Ted Nelson founded his work on the work in the sixties by Douglas Engelbart from Stanford Research Institute and — further back — the thoughts of Vannebar Bush, scientific advisor to United States president Roosevelt.

The HyperText concept stresses two important characteristics of information:

- Information is interconnected with other information, and the connections are frequently different from those in accordance with which the information is ordered in the first place
- The importance of a specific set of connections between blocks of information may vary between persons, purposes and/or time

Vannebar Bush criticised the existing information systems for being based on rigid hierarchical index methods based on the system for libraries:

> «The human mind does not work that way. It operates by association. With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails, carried by the cells of the brain» (Bush 1945).

HyperText, in the Ted Nelson sense of the word, is a collection of pieces of text and other types of information in various sizes and forms coupled by roads, among which the user is free to choose. Typically, a hypertext system may be viewed as a network of text fragments linked together by a complex network of pointers. A HyperText document is not necessarily read sequentially like a novel. It is read according to contents and associative thoughts like an encyclopaedia. One reads the text, spots a relevant or interesting term or concept and is then able to go directly to the text explaining the term by pressing a key or clicking a mouse. HyperText is thus not one piece of text, but in principle an unlimited number of texts depending on how — and how many times — the user traverses the text pieces and their connections. Such a system will today often include support for texts, graphics and even music fragments and moving images, and the more general (and hype!) term "hypermedia" is to be preferred to "HyperText".

In many ways the odd pieces of information material collected in a museum could be looked upon as an enormous HyperText or hypermedia structure. A freedom to organise and browse through information this way looked promising. At that time we had no HyperText tools at hand, so instead we forced an outline processor to simulate a prototype of a museum information system organised after these principles. The demonstration of the prototype harvested an overwhelmingly positive response, and it was quickly decided to try and follow up these principles in the design and development of the real system.

The starting point and guidelines for the development were thus a user interface, HyperText–like, and to a certain extend the functionality and data structures inherent in CMS. Top–down design in its pure form.

24.2 AN OBJECT ORIENTED APPROACH

One of the central problems of software engineering is that of coping with complexity. It was clear cut from the beginning, that developing a system able to handle the inherent complexity of the data in a museum, combined with the diversification in museum types and documentation practices, constituted a tremendous challenge to a computer system created to deal with reality in nice square boxes.

It was obvious, that the system should not only have a high degree of configurability but also be extremely flexible. Managing complexity simply requires great flexibility. Object oriented design methods address this issue precisely.

> «After years of relative obscurity, object orientation appears to be entering the mainstream of commercial computing for both software developers and end users. A shift to object orientation is occurring simultaneously across a wide range of software components, including languages, user interfaces, databases, and operating systems. While object oriented programming is no panacea, it has already demonstrated, that it can help to manage the growing complexity and increasing costs of software development» (Winblad *et al.* 1991).

The above quotation from the book *Object oriented Software* by Winblad, Edwards & King, is an excellent summary of what is happening in the world of computing today. Although exciting research and development are taking place on many fronts, no single software topic currently enjoys as wide a scope or as heavy an impact as object–orientation.

One of the fundamental reasons that object orientation is enjoying such success as a programming paradigm is very simple: the real world is made up of objects. A sword is an object. A grave is an object. An excavation is an object. An entire museum is an object!

The catechism of object oriented is that *encapsulation, polymorphism,* and *inheritance* are the criteria of a "true" object oriented programming system.

Encapsulation means that classes of objects (and therefore, of course, the objects themselves) are defined not only in terms of what they are, but also in terms of what they are able to do. The methods are an integral part of the object. Polymorphism means that a particular operation can be performed on more than one type of object. Inheritance refers to the ability to create new objects (or, more properly, classes of objects) that take over all properties of a previously defined object.

Despite the virtues of object oriented systems, it does not make sense to insist on conformance to the narrow technical definition of "object oriented". A tool is not automatically useless simply because it does not meet the strict criteria of "object oriented". The best criterion by which to judge a tool is still its suitability to the job one needs to do.

24.2.1 The design

As mentioned above, one of the key features of the system should be its run time configurability and ability to adapt to different situations including different user groups and complex data structures.

With this in mind, I took a quite radical decision regarding the design: the system should be constructed around a Hyper Object Engine (HOE), *without any explicit references to museums at all,* but of course with sufficient power in its methods and structure generation ability to handle the data structures and operations necessary for using the Engine in a museum documentation application.

The idea was, that the HOE should take care of all the essentials in the handling of objects inclusive the building and maintenance of the hyper structure, the navigation tools, validation, security levels, reporting, etc., so that only a *very tiny* application–specific module had to be linked in to conclude an application. As a matter of fact the application–specific part of the Danish MuseumsIndex amounts to less than 5 % of the total code!

Furthermore the HOE should draw all its run time parameters from externally maintainable sources, so that no changes in the HOE were needed because of changes in say, menus, parameter table structure, user language, etc. Only if new methods had to be introduced, a recompilation should be needed, and then again — only of the application–specific module.

24.2.2 The objects

The primary object types or classes in the system are the *Frame* and the *Link*. These classes are the fundamental building blocks needed to construct a hyperstructure.

It is necessary here to stress a vital point about the Link. In a traditional relational system two tables can be linked or joined via a common unique key–field. But this link is purely a *passive,* "hard–wired", link. Let us say we have a group of painters and a set of paintings. With a passive link, you must choose between attributing a particular work to an artist or not doing it. With an intelligent link, you would be able to classify and/or explain your premises for the attribution. The validity of a link is neither a quality of the painting nor the painter. The same problems exist with most connections. If you attribute an archaeological artefact to a specific dating, the same problem exists.

What is mandatory in a genuine hyper structure is, that links are *active* connections with attributes which make it possible to qualify and explain the link. This is also important if the system should be able to accommodate differences in methods and working style. Although artefacts, photos, paintings, maps, etc. ordinarily are looked upon as the salt of a museum's collection, there are other schools who value *the context* as the area of primary interest or in its extreme form: objects without context are worthless. The HOE should, of course, be able to cope with both views (and those between).

Among the secondary classes *Block* (an attribute in Frame) *Menu* and *Blob* (Binary Large OBjects), are the most important ones.

Now let us look at the classes in more details and describe some of their features, attributes and methods.

24.2.2.1 The Frame class

The Frame class could be described by its attributes and methods like in Table 24.1.

The list of attributes (or instance variables) is — as the methods — by no means exhaustive, but merely serves to convey a general idea about the FRAME class.

The appearance and behaviour of an instance of FRAME will be controlled by data stored in external files. The Show method will for instance use Rows, Columns, ScreenOffset, ScreenSize, etc. to paint the screen.

One of the attributes deserves a special consideration — the *Block*. As mentioned before, *Block* too is a class or object type. The Block Class could be described as in Table 24.2.

The attributes of the Block Class are what comes closest to an ordinary field in a data base record (or tuple): An instance of BLOCK has a name, a type, a length and a value. But it also contains attributes that are not standard for a data base field. It has a Picture clause, which can be used to structure its appearance and editing on the screen. As shown it also contains LookAccess and EditAccess attributes which make it possible to administer who has the right to see the contents and to edit it. Normally the instances of Block will correspond to either a data base field or a look–up field, where Row and Col will be used to position the contents on the screen.

The persistence of some of the attributes of the object, or instances, from session to session are secured through transferring their values to and from records or tuples in a database with a name related to the FrameId. The methods in FRAME, CreateRecord, FetchRecord and UpdateRecord are taking care of the transfer between records and the actual BLOCK object.

FRAME CLASS:

FrameId	Constructor()
Name	Destructor()
Indexes	Show()
EditArrays	FetchEditInfo()
Mode	Edit()
Blocks	CreateRecord()
Rows	FetchRecord()
Columns	UpdateRecord()
BorderColour	DeleteRecord()
InnerColour	FetchNext()
ScreenOffset	FetchPrevios()
ScreenSize	FetchFirst()
Status	FetchLast()
TextNo	CreateLink()
NumLinks	DeleteLink()
.....	.....

Table 24.1: The Frame Class

BLOCK CLASS:

Name	BlockInit()
Type	ExecMethod()
Length	.....
Row	
Col	
Value	
Picture	
MethodName	
Mode	
Help	
LookAccess	
EditAccess	
Cargo	
.....	

Table 24.2: The Block class

An extremely important attribute in BLOCK is *MethodName*. To this attribute, a name of a method can be assigned, and the method itself can be executed by the objects ExecMethod–function later on. Cargo and other attributes can be used as parameters. The method is thus not "hard wired" to the class, but can be externally specified among available methods or functions in the HOE and/or in the programming language.

One of the main uses of this technique is to call look–up functions, pick–lists, validation–routines, etc. Through a special coding structure of the MethodName, one or more of these routines can be called before editing, during editing and/or after editing.

A particularly important use of this facility is as a mechanism to access and/or create (and control access to) subframes linked to the present frame. The linking is *semantically* attached to Block:Name, but this attribute merely serves as a placeholder for the number (automatically taken care of by the system) of subframes attached to the present object of the particular type denoted by the Block:Name and the content of the Block:Cargo. The Block:Cargo also controls whether or not it is allowed to link to existing subframes or whether only links to new subframes can be created. The Block:Picture denotes (together with the Block:Cargo) *how many* subframes to which it is allowed to construct links.

Another feature available through the Block:Method mechanism is its use in Multi Media applications. Depending on the contents it is possible this way to show digitised pictures, play sections of video disks, video tapes or CD's or play digitised sound or speech.

24.2.2.2 Implementation of free text

The use of text in documentation systems is a questionable issue. On one hand it allows the documenter to elaborate freely on softer issues in the documentation. On the other hand, free text has a tendency to quickly overflow a system, take pressure away from a needed discussion about terminology and classification, and unstructured text has only a limited value when searched due to differences in culture, education, documentation style, terminology, vocabulary, etc.

In the Danish Museums HOE application the use of text has almost exclusively been confined to the links. The objects in the FRAME subclasses (artefacts, photos, persons, etc.) is supposed to be described sufficiently by their attributes (with con-

trolled vocabulary wherever possible). But the content in the LINK objects is some times of a more non–conclusive type, where a simple code is not enough to characterise the reasoning behind a link.

A *LINK* object always has — besides a Status attribute — an attribute which can be looked upon as a "hard wired" link to a *TEXT* object. If no *TEXT* object is attached, the attribute is NIL. The *TEXT* object has an attribute, TextLine, and a "hard – wired" link to a free text (NIL when not used). If free text is needed in a *FRAME* object — for instance to a verbal description of a site or a book — the sub frame link mechanism is used to attach a *LINK* object with only its inherent *TEXT* object initialised. All texts in HOE are indexed in a way that makes searching possible not only by single words, but also in various combined ways. Via the text search you can find the objects the text is attached to, and it is possible to narrow the search to only consider texts attached to specific FRAME subclasses.

2.2.3 The subclasses in Dansk MuseumsIndex

The *Danish* museums HOE application has 30 sub–classes of *FRAME* defined at present. In Table 24.3 is a list of the most important with the subclasses that it is possible to link to, indented below the class–name

Let me here emphasise once more, that *none* of the characteristics of these subclasses are stored within the HOE itself. The definitions of the classes together with the affiliated *BLOCK* definitions are confined in external files, and only a few of the application specific functions implied by the MethodName attribute, are placed in the separate application–specific module.

As it probably appears, the structure here is predominantly a hierarchical one. The HOE has the ability to handle more complicated structures as well. For instance MAP objects linked to other MAP objects, or ARTEFACT objects linked to sub artefacts. These structures call for a very strong morale (and deep knowledge) on the part of the user however, in order to avoid circular references with objects in the end pointing to themselves. Although the HOE is able to handle these kinds of structures, it would be very difficult and time consuming to check for reports that ultimately would send the system in an endless loop. Instead, it is recommended to make reference to objects of the same class (or above) in text or by utilising a special reference class. During a browse of the hyper structure it will be possible to utilise the HOE's built–in macro facility to find and show another object referred to by a member of the same class.

24.2.2.3 The data structure

The data structure for a *dynamic* network database system — as the one the HOE is suitable for — could be quite complicated to describe with ordinary data structure description tools.

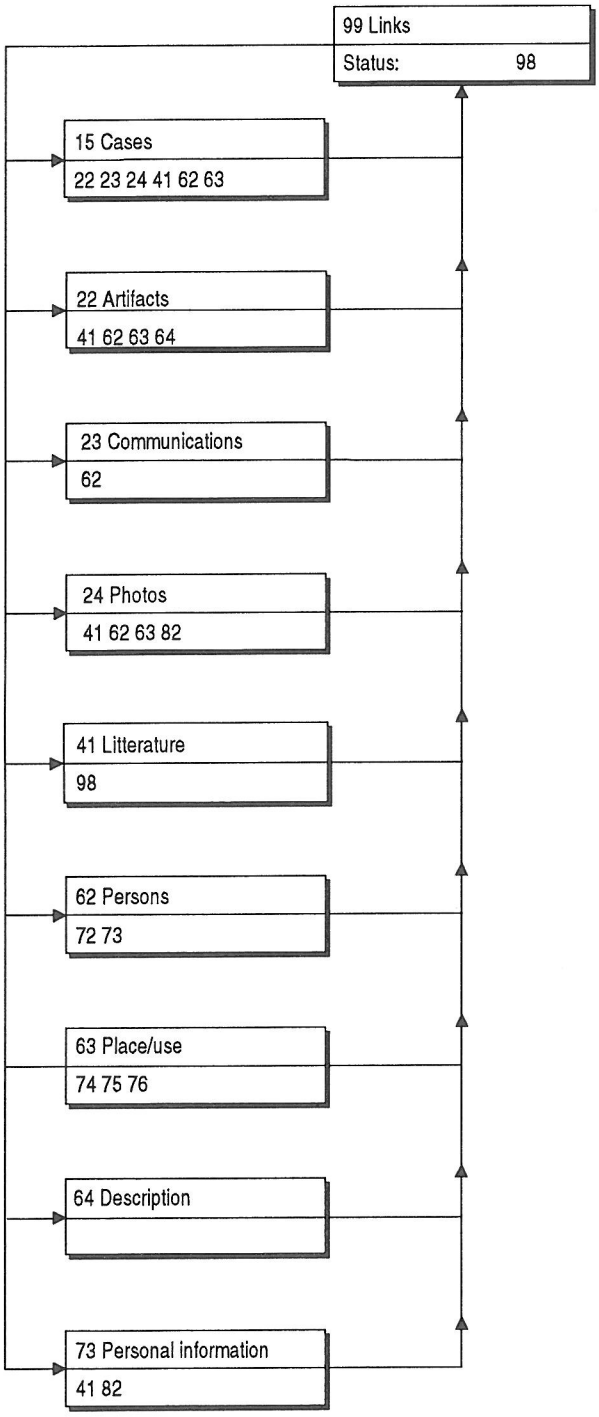
Case	
Correspondence	Conservation
Artifact	Literature
Photo	Deposit
Map	
Tape	Map
Report	Participant
Archive	Place/use
Literature	Dating
Participant	Conservation
Place/use	Literature
Dating	Deposit
Conservation	
Deposit	Tape
	Participant
Artist	Place/use
Photo	Dating
Correspondence	Conservation
Tape	Literature
Literature	Deposit
Report	
Artifact	Report
Exhibition	Participant
Participant	Place/use
	Dating
Artwork	Conservation
Participant	Literature
Exhibitions	Deposit
Report	
Dating	Exhibition
Literature	Participant
Correspondence	Dating
Photo	Conservation
Conservation	Literature
Tape	
Deposit	Literature
	Participant
Artifact	
Participant	Correspondence
Place/use	Participant
Dating	
Conservation	Conservation
Literature	Participant
Deposit	
	Archive
Photo	Participant
Participant	Place/use
Place/use	Dating
Dating	Conservation
	Literature

Table 24.3: The subclasses in Danish MuseumsIndex

A way to do it could be the following chart, which shows a part of the data structure for the Danish museums HOE–application.

24.3 SEARCHING WITH THE HOE

Searching for data in a Hyper Structure is not like searching in a flat file. Certain search structures have to be employed and the primary choices when searching are:
1) What FRAME sub class are you searching *FOR*?
2) What FRAME sub class(es) are you searching *BY* ?
The easiest case is, if you want to search directly in a specific FRAME sub class. For instance among the artefacts, where the answers to both questions will be ARTEFACT. You can here choose between an indexed search, browsing, QBE or by setting up a search criterion. Of course the indexed searches are by far the fastest. An indexed search for a specific object number will, for example, in a 32 MB file will take less than one second with a 386/33 PC and a reasonable fast hard disk. As with other parts of the HOE, the number of indexes in each FRAME sub class is externally defined and it is thus possible to introduce new indexes without recompiling the HOE.



24.4 SEARCHING ACROSS SUBCLASSES

A little more complicated are combined searches where more than one sub class is involved. Let's say you are searching for those *cases* that contain *artefacts* of a specific type. Your answer to the first question will then be CASE and to the second, ARTEFACT. Normally these choices are made through the MENU objects and — depending on the number of "hits" — the system will, as conclusion of the search, provide you with the possibilities of browsing the hits, searching via one or more indexes, a QBE search or using the search language.

Things get even more complicated when the search has to be carried out on several classes. Let us say again that your answer is CASE to the first question and PARTICIPANT (person or institution) to the second. After you have specified the search criteria for a person, the system will now not only find those CASE to which this person is linked, but also the CASES where a sub–class to a sub–class to the CASE contains a link to that person.

Figure 24.1: Part of the data structure for Danish MuseumsIndex. The boxes on the left are subclasses. The upper part of the boxes contains the FrameId and Name and the lower part the FrameIds for the subclasses it is possible to connect to. In the lower part some of the FrameIds are underlined. To these subclasses it is possible to connect to existings *frames (n:m relations) — otherwise it is only possible to connect to new frames (1:n relations). These conditions are stored in the block objects and can as such be changed without recompiling the program. Connections can only be made by sending messages to the Link class where the connection also can be classified and/or annotated.*

24.4 MANOEUVRING IN THE HYPER STRUCTURE

When — as in the last example — a case has been found, the object can be edited (or deleted), or the object can be used as a *starting point* for traversing *the complete* hyper structure. Every time you have an object on the screen, a press on a button, <F6>, gives you the complete structure or hierarchy for the object and its sub class objects in a semi–graphical form. The structure is shown in a quite terse form in order to put as much information about the structure on one screen as possible, but moving the browse cursor around will provide more information about the individual objects in the structure. And if you choose to do so, you can — by pressing <Enter> on a specific object — call the whole object on the screen, and afterwards return to the hierarchy.

Another possibility is to press the <F7>–key. This gives you the object level above and below the present level on–screen, and by pressing <F10> on any of these levels you will make this level your new central level. With new presses on <F7> you will be able to move around in the complete structure and — as with <F6> — you can at any place press <Enter> and see the actual object.

This associative browsing is one of the strong points in a Hyper Structure, but you may also get lost in the structure! Fortunately a press on <F5> gives you — in pick list form — a list of the places you have visited, including the starting point, and a move of the cursor and a selection with <Enter> brings you right to the object you want.

24.5 REPORTING WITH THE HOE

For each HOE application there will be some standard reports defined, but the system contains a quite powerful REPORT generator with which the user has the ability to create his own report masks. The report generator contains some special functions which allow the user to create reports across different FRAME sub classes. Another function allows the user to output bar codes.

Beside the special functions. it is also possible to use quite a few standard functions in the reports and some counting functions. It is possible to send the report output to a file, so the report afterwards can be further treated in, say, a desk top publishing program or word processor.

24.6 OTHER HOE APPLICATIONS

The system maintenance part of the Museums-Index is also built using the HOE. The application–specific part is here the code which takes care of the management of user codes, passwords, the user definable colours in the program, reindexing of files, etc.

A particularly important feature of this application is its *Export* and *Import* functions. Through these functions *complete* hyperstructures can be exported and imported. This feature can also be utilised to perform non–standard searches by exporting a selected set of data in hyperstructure form, and afterwards — with the standard HOE ! — perform new searches in the exported structure.

And needless to say — the development of the HOE is also taken care of in a HOE application where the HOE's hyperstructure ability is utilised as a data and function dictionary including screens, index lists and object structures together with menus (and menu generator), block information, etc.

References

Bush, Vannebar
1945 As we may think. *Atlantic Monthly.*
Winblad, Edwards & King
1991 *Object–Oriented Software.* New York.

Author's address
Jørgen Feder
Dansk MuseumsIndex
Nørrebro 17
DK–5900 Rudkøbing