OTHY: Object To HYpermedia

Franck Barbeau[1] and José Martinez[2]

[1]MAIF – Service Bureautique, 200, Av. Salvador Allende
79038 Niort Cedex 9 – France
Franck.Barbeau@irin.univ-nantes.fr
[2]IRIN – Université de Nantes, IRESTE – La Chantrerie – BP 60601
44306 Nantes Cedex 3 – France
Jose.Martinez@irin.univ-nantes.fr

Abstract. In this paper, we present a Web-based universal browser for heterogeneous and non federated databases. Recently appeared hypermedia methods are at the core of this system. However, contrary to these methodologies, our tool directly supports the conception, the navigation, and the presentation phases without requiring any modification of the databases. It is mostly based on the OTHY framework, which is a library of classes to develop different kinds of presentation based on well-established hypertext concepts. The design of our tool and a first implementation resulted in a prototype under the O_2 OODBMS. This implementation was convincing. Consequently, a Java development and two querying and relational database retro-engineering modules are on the path to be added to this prototype.

1 Introduction

MAIF (*Mutuelle Assurance des Instituteurs de France*) is a French insurance company devoted to teachers. It is quite large with over one hundred agencies, approximately 4,500 employees and as many computers. In the short term, all the applications will use the Web technology. MAIF needs allow us to stress three important characteristics required by users of an heterogeneous and non-federated information system. These characteristics are (1) to navigate on the data, (2) to query them with an intuitive, non formal interface, and (3) to present different views on the data for different users and even for a given user.

Hypertext [6] and more recently hypermedia [24], [10] are not isolated research domains. Indeed, outside "classical" applications, like help, documentation, or workgroup, hypertext concepts have been adopted by many application areas. Thereby, their flexibility is greatly enhanced. More precisely, Web user-friendliness is nowadays demanded by users. Conversely, a review of the proposed architectures, including the seminal Dexter model [9], highlights the fact that hypermedia systems need database functionalities. As an example, Hyperwave is an hypermedia system based on an object-oriented database engine. In the field of databases mature methodologies such as RMM, OOHDM, EORM, etc., have emerged [16].

In a way, our proposition merges these two complementary approaches by (1) offering an hypertextual view of databases and (2) offering a customisable set of hypermedia tools and concepts. The basic aim of the work is to allow conventional database accesses with an hypermedia interface. This interface must not be intrusive, i. e., it must not impact the used databases.

This paper is organised as follows: First, we detail user needs at MAIF, but they may be generalised to any important organisation. Then, we present related work: methodologies and tools for navigating/querying databases. In section 4, we present the first part of our proposal: the system architecture based both on hypermedia and database concepts, in an Internet/Intranet-based platform. In section 5, we see in details the principal tool of this project: a Java applet. This applet is composed of four modules, the most important of which is OTHY, a framework developed to present information to the user by translating database instances into "pages." Section 6 validates our approach by presenting our V0 prototype. In fact, in order to alleviate the initial programming effort, OTHY was implemented under the O_2 OODMBS [1]. The conclusion points out important issues in the development of the on-going V1 operational prototype: (1) reverse engineering, (2) query possibilities, and (3) schema integration.

2 The Needs

The architecture synoptic of the information system of many organisations may be as follow. A single desktop computer has to access a lot of independent sub-systems (different databases, files, different kinds of database management systems), located on different servers, sometimes physically distant. This is true at MAIF with contract databases (textual data), precious object databases (with pictures and descriptions of sold objects with their auction prices), etc.
A study of the users' needs at MAIF highlighted three main requirements: The most important need is the possibility to navigate easily among this information mass, with the ease offered by the Web technology. This *navigation* must be possible not only in one database, but also between databases with common or related elements. In other words, users demand a single interface to the whole information system.

However, navigation alone is not sufficient; some techniques of database *querying* must be added to the system. But experience demonstrated, through some old systems based on classical querying, that too formal queries are supplanted by manual querying, e. g., users preferred to access directly to the printed version of a textual database! Queries are envisaged solely as a means to retrieve rapidly an entry point close to the wanted information, therefore avoiding a lengthy and tortuous initial navigation.

Finally, the system should offer to the user the possibility to customise his or her *view* of the information, an extension of the well-known concept in relational database management systems.

3 Related Work

Some authors claim that an hypertext is not a database [22]. On the one hand, they are right, because data managed by hypertext are often poorly structured, if not at all. On the other hand, the use of the database approach is still the best means to organise a collection of data as far as we are concerned. A database stores information with relationships between them. Data and relationships are modelled by conceptual schemas presenting common concepts. These concepts may be network, relational, entity-relationship, semantics or object-oriented notions like aggregation, generalisation, etc. [2]. An hypertext model is a complicated model that needs high level tools. [29] recommend the use of an OODBMS as the more rational solution to store hypertext data. An underlying structure to the hypertext exists and it is usually clearly translated by the interface. The real difficulty is the intensive use of texts, i. e., semi-structured data, which are badly managed by DBMS. This lack is emphasised by the current growth of the forms of semi-structured data: images, videos, and audio. However, this pitfall is being solved thanks to meta-modelling, e. g., SGML (*Standard Generalized Mark-up Language*) for texts [30], MULTOS for office documents [20], HyTime for multimedia documents [23].

We shall detail some methodologies and tools that influenced are design and tool.

3.1 RMM (Relationship Management Methodology)

RMM [11], [3] is a conceptual method for the design and construction of hypermedia applications. It is based on the entity-relationship conceptual model. It introduces hypertextual navigation concepts during the logical description step of a database schema.

To simplify, each entity type has a corresponding page model, or sometimes several pages (slices) if the number of information is too large. Each entity instantiates the page model with its attribute values.

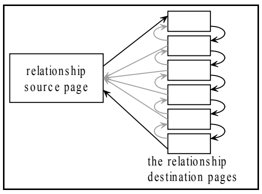

Fig. 1. A guided tour

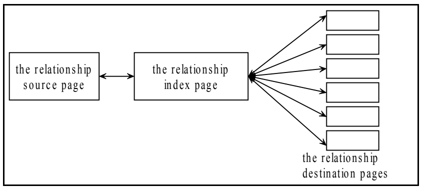
the relationship source page
the relationship index page
the relationship destination pages

Fig. 2. An indexed tour

Each relationship role has zero, one or more anchors according to its cardinality. A zero-valued cardinality is translated into a text indicating the role absence. A one-valued cardinality is translated into an anchor representing the relationship nature. A multi-valued cardinality can give rise to different presentation of the relationships, namely a guided tour, an indexed tour, or a guided indexed tour. A guided tour (Fig. 1), unidirectional or bi-directional, allows the user to follow sequentially all the related (and linked) instances of the source object, possibly with a direct return to the initial object. An indexed tour (Fig. 2) needs a discriminating element, e. g., at the best a foreign key in relational terms. A multiple but limited cardinality can be translated into a page containing all the accesses to the linked elements. Too large a cardinality would give rise to a separate page. Moreover, a very important cardinality would give birth to a hierarchical index. Finally, it is possible to combine guided tours and indexed tours (Fig. 3)

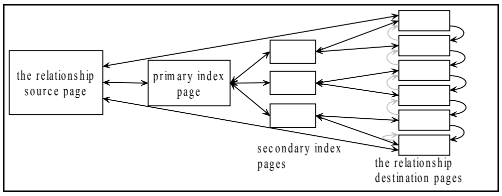
the relationship source page
primary index page
secondary index pages
the relationship destination pages

Fig. 3. The most complicated case of navigation between a source and related destinations

The slice concept avoids scrolling in windows. One slice is selected as the entry point of the object (in general, it is the slice containing the key.) (Notice that index pages may be considered as object slices.)

A relationship may itself be viewed as an object. Therefore, it is possible to create pages for relationship instances. These pages are composed of main slices of each object participating in the relationship. (Furthermore, that if we encompass sets of relationships instances in the definition, we merely obtain the classical tabular view in relational databases.)

3.2 OOHDM (Object-Oriented Hypermedia Design Model)

OOHDM [27], [28] is a direct descendant of HDM [8]. It differs from HDM both in its object-oriented nature, and in its integration of special purpose modelling primitives for navigational and interface design.

With OOHDM, a hypermedia application is built in a four-step process.

In the conceptual phase, an application model is built using object-oriented modelling principles. The main concern is to capture the domain semantics, hence, any object-oriented method may be used, in practice OMT has been chosen [25]. The single extension is the possibility to offer several types for one attribute [28].

During the navigational design step, the structure of an hypermedia application is described in terms of navigational context (classes such as *nodes, links, indexed tours, guided tours*.) Different navigational models may be built for the same conceptual schema to express different *views* on the same domain. Nodes are defined as object-oriented views of conceptual classes, links reflect relationships intended to be explored by the final user and are considered as views on relationships.

The abstract interface model is built by defining perceptible objects (*picture, city map* ...) in terms of interface classes. These interface classes are built as aggregations of primitive classes (*text fields, buttons*.) The aim of this step is to define the way in which different navigational objects are going to appear.

The last step maps interface objects to implementation objects. This step may implement interface objects in different platforms like Director, HTML, etc.

RMM and OOHDM are methodologies, i. e., they give guidelines. Our tool takes into consideration the general rules of these methods but introduces a great deal of freedom through extensibility of the OTHY object-oriented framework. The framework that is to be presented in the sequel supports the well-established concepts of hypertext design, but can be customised for very special requirements too.

3.3 HyperWave

HyperWave [19] is an hypermedia system, conceived to be a distributed Web-server, that is based on the principle of recursive collections. A collection is a composite object, and may contain objects or other collections [14], [15]. To further improve navigation tools in a HyperWave site, the notions of clusters and sequences were added. Clusters permit to partition documents of a given collection with respect to some attribute value, e. g., presenting either the French, or the English pages of a large multi-lingual document. Sequences organise documents as doubly linked list, thereby offering guided tours.

Each object of HyperWave has system-specific attributes (title, author, keywords, creation and modification dates ...) These attributes or meta-information are used to perform more accurate queries. Links between documents are stored in a document independent database. They are bi-directional, i. e., they allow to find document sources from document destinations.

HyperWave is based on the principle that users may be contribute to the server life. This involves that the system is multi-users, multi-developers. This allows registered users to own a personal collection where they may store bookmarks, history, etc.

In our opinion, the problem with HyperWave is that the schema model has been fixed. We believe that the system must adapt to the initial organisation of the data rather than the reverse. OTHY relies on this hypothesis.

3.4 PESTO (Portable Explorer of Structured Objects)

PESTO is a user interface that supports browsing and querying of object databases [4] [1]. It allows users to browse classes of objects and to navigate across the relationship that exist among them.

All attributes of a given class are shown in the same window: simple attributes are presented as being contained in the object window, reference attributes are presented as buttons that bring up new windows for displaying the referenced objects.

In addition, users may formulate complex object queries through an integrated query paradigm (*Query in Place*) that presents querying as an extension of browsing.

With PESTO, all the objects are presented according to a default presentation: a window with lists of attribute/value pairs. Contrary to our system, PESTO does not afford views, which is a severe limitation for end-users who do not want to deal with the totality of the information stored in a database.

As a consequence, PESTO querying capabilities are much like conventional database ones, i. e., formal though graphical. In addition, PESTO works in a single envi-

[1] This paper gives a nice (short) survey of the history of querying/browsing tools since the seminal QBE (*Query By Example*) system.

ronment. We plan a more intuitive way to query an heterogeneous set of databases, based on information retrieval techniques.

4 System Architecture

Our proposition is based on the architecture presented in Fig. 5. The possibility for desktop computers in one organisation to access to different conceptual and physical sources of information, requires the use of a "universal client." Therefore, we use a Java applet to obtain this uniqueness. A side effect, and an important advantage of using a Java-enabled browser is that there is no software deployment during the system installation step, nor the forthcoming modification, nor the progressive integration of new information sub-systems. At the very most, only the browser needs some upgrades from time to time. At MAIF, this is undoubtedly a desirable feature.

For reasons that will become clear in the sequel, we adopted object orientation [21]. In contrast, the information sub-systems are heterogeneous and certainly not based on a single conceptual, nor physical model. Therefore, specific integration modules are responsible for reverse engineering of the underlying database meta-schemas. Currently, we are working on two such integration modules. Since relational database management systems (RDBMS) are wide-spread, it is an important issue to access them with our tool. Fortunately, all of them conform to the JDBC/ODBC standard. Therefore, any vendor-specific RDBMS is now accessible as a standard and unique model. But, this module is always required since it must translate meta-schemas from first normal form relations to non-normalised classes. This module will implement an algorithm developed by our research team [26].

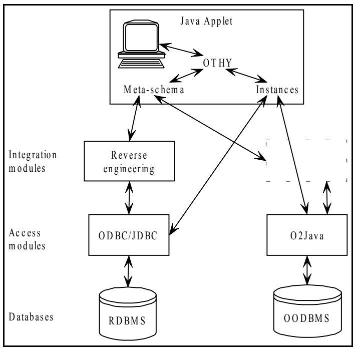

Fig. 4. System architecture

The major alternative to relational technology is object-orientation (equivalently, object-relational.) We hope that the progressive adoption of the ODMG standard [5] will make this module unnecessary in the future. In the first version of the applet, only the O_2 OODBMS and its *O2Java* binding module are to be used, which facilitates our work.

Finally, the “Java applet”, i. e., the run-time intensive part of the whole architecture, is still a complex module. It is composed of four sub-modules. First, the meta-schema is in charge of loading the database meta-schemas, and to federate them in order to allow for inter-database navigation.

The accessed instances have to be interpreted by the applet, i. e., they will be split up into their very basic components: basic data types (integer, boolean, char ...), type constructors (list, set, tuple …), as well as simple methods (observers, which return a view of the object without modifying it.)

Then, the OTHY module, which is at the core of our proposition, is responsible for translating instances into “pages.”

Lastly, the user interface presents “pages” as visual components (widgets), and constitutes the visible part of the iceberg.

5 The Applet Architecture

Now, let us detail two sensitive aspects of the applet architecture to develop: (1) the meta-schema and (2) the OTHY framework. Our tool works at the meta-level; it has to manipulate instance components, hence it has to know about their structure and relationships. We rely on an object-oriented meta-schema. Then, the OTHY framework uses this information, as well as the hypermedia concepts that it incorporates, to translate database objects into “pages.”

5.1 The Meta-schema

Fig. 5 presents the proposed meta-schema. It is relatively usual but it contains some interesting details: The first important notion is the use of simple inheritance, which is easier to understand and to implement. With simple inheritance, we do not care about attribute and method conflicts. Contrary to the last proposal of the ODMG, neither is the interface concept retained at that time.

Next, we use an external object identifier, i. e., a key in the relational world. The identifier is a set of particular attributes that permits naturally to distinguish an object from other objects on the same class hierarchy. More loosely, this identifier can be only a local key, i. e., a value differentiating instances that are related to a given instance. In the worst case where no key exists, this “identifier” is still useful: it allows our application to present a significant anchor to the user rather than the mere instance class name. However, this default behaviour can be overridden for special application needs.

Then, this meta-schema stresses the relationship and role concepts. More precisely, we are interested in the cardinalities of the roles. Knowing their values makes it possible to choose between different kinds of navigation detailed in section 5.2 (guided tour, indexed tour, direct access; separated pages or anchor incorporation in the source object.)

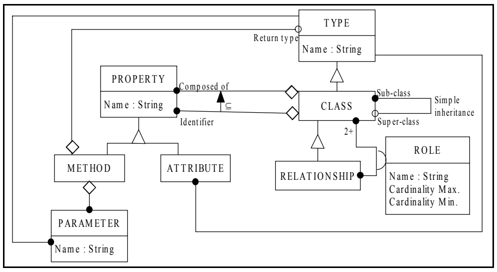

Fig. 5. An object-oriented meta-schema model

The applet meta-schema will contain several separated schemas. A long-term idea is to create relationships between these different schemas. For instance, the system may retrieve identifiers with the same name and may consider that this information comes from the same source, but is replicated in different databases. Such obvious situations arise with the employee number, or his or her first and last names. This is part of the issues to be explored in order to improve navigation and thereby to reduce the recourse to querying.

5.2 The OTHY Framework

OTHY, the core of our proposal, can be seen as a framework [12], [13], [7], [17] for translating objects into hyper-linked components. The aim of OTHY is to create a presentation "page" (with information and anchors to navigate) from a specific instance.

The OTHY framework implements prominent points of the RMM and OOHDM methodologies: (1) guided tours, indexed tours from RMM, and (2) the use of the objet-orientation, and the concept of views from OOHDM (See Sect. 3.) The philosophy of RMM is to construct hypermedia applications on top of a database conceptual model. We saw in section 2 that we are typically in this application case. Nevertheless, this method is too rigid, because it is impossible to express external views of a database application. Nevertheless, this point is taken into account by OOHDM which introduces navigational models on top of the conceptual schema. From a schema, several views may be defined.

In mathematical terms, the main function of OTHY consists in translating an instance into a "page," with respect to a given view:

$$\texttt{OTHY : DB} \times \texttt{View} \times \texttt{Object} \rightarrow \texttt{Page} \quad \textbf{(1)}$$

Fig. 6 presents the needed framework skeleton to perform this transformation! This framework integrates most of the concepts of RMM and OOHDM. But, above all, it realises all the steps from navigation to implementation without requiring any modification of the database, nor development of code in the framework, except for very specific needs.

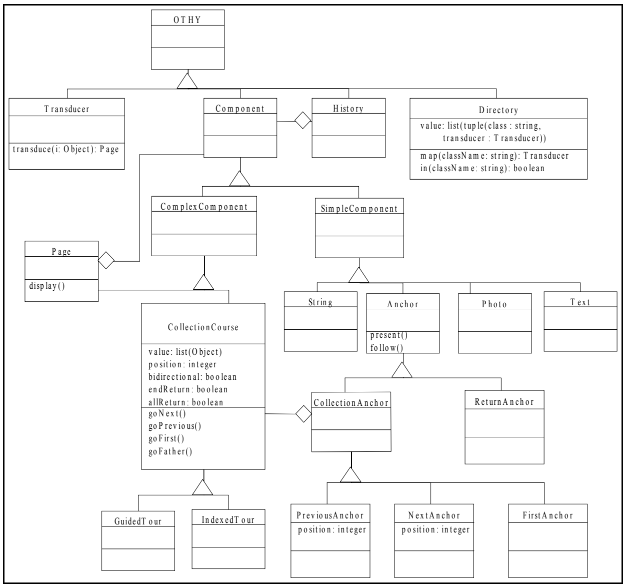

Fig. 6. The OTHY framework

Navigational Views

The class `Transducer` is the essential support of the transformation function. This class is a generic class and allows to transform any object into a page. It is the root of a concrete and/or abstract class hierarchy. This is very useful for extensibility and adaptability; each class may own a specific transducer. To go ahead, the system is able to manage several transducers for each class: specific, generic or parametric transducers.

The relationship between the transducer and the application class is made via the `Directory` class, which manages an associative relationship between class types (or, more precisely, the names of the class instances in the meta-schema) and transducer instances. Each known class name in the meta-schema is linked with a transducer reference. Each (kind of) user may obtain a distinct directory, where he or she may use both different transducers for the same instances, and different presentations or navigation according to his needs. This is the way we implemented view and user profile concepts.

Interfaces

The most important hierarchy, in terms of number of classes only, is for the hypermedia presentation of data. There are usual page concepts, anchors, guided tours, indexed tours and different kinds of media (*text*, *string*, *picture* …) Instances of these classes are the result of applying a transducer to a data instance. This result is presented to the user for browsing. Properties extracted from an object can be attributes or observers without parameters. (In the case of query extension, we should take into account other observers to allow users to provide parameters at run-time.)

In this hierarchy, we find the class `Page` with the method `display()`. This class is the only one having this method. This page amalgamates `Components` which may be `SimpleComponent` or `ComplexComponent`.

Among simple components, we have the usual items: `Text`, `String`, `Photo`, `Anchor`, etc. A complex component may be a collection of objects to browse. The tours may be guided tours or indexed tours. Objects `Collection` and `Anchor` are closely linked because three kinds of anchors over four are specific to collections (`PreviousAnchor` to return to the previous object in a guided tour, `NextAnchor` to go to the next object, `FirstAnchor` to go to the first object of the collection to browse).

The class `History` is a common tool, for the implementation step. It allows to return to a previously visited page. At that time, it is only a stack. But it is possible to extend this historic concept in order to browse it like a graph.

6 The V0 Prototype

A first implementation of our model was realised under the O_2 OODBMS. The major advantage of this choice is in the simplification of the implementation. This was done to validate our approach before the development of the full tool. More precisely, advantages are three-fold: (1) homogeneity, (2) direct use of the O_2 meta-schema, and (3) less need to interpret the instances. Using a single environment eliminates connections and the "impedance mismatch" between a host language and the database. The implementation was achieved with the native O_2 meta-schema, though it does not have all the needed characteristics. More precisely, the notion of natural key is still absent and we choose to use the value of the first attribute as the anchor value in a page. Also, the development was easier because instances are directly accessible as objects. In the final version, we will have to decompose objects as they contain components (attributes and methods.)
A simple application was coded. There are five classes (`Author`, `Publication`, `Book`, `Magazine` and `Publisher`) linked by two relationships (`Wrote` and `Published`) or by inheritance. Note that the O_2 classes do not reference OTHY, as implied by the principle of total independence that we impose to ourselves (See Table 1)

OTHY is already a usable framework but –and this is one advantage of the object-oriented approach– it is possible to extend it. This is really true for the instance presentation definition. Consequently, we added some subclasses for `Publication`. Relationships between these classes and application classes are notified to OTHY through a directory.

At run-time, link between an application instance and the instance of an OTHY transducer is made via the directory associative memory. There are two input points: the instance of the directory used (function of the user connected) and an application object. First, from this object, thanks to a generic O_2 method named `title` that returns the instance class name, we obtain the string "`Author`." Secondly, this name is transferred from the application domain to the OTHY framework. This step uses a `Directory` object . It is an associative memory that is able to retrieve the instance of `Transducer` that has been previously associated by the user (or a default one if such an association has not been specified.) Thirdly, the retrieved transducer is applied to the initial instance. This returns a page, using either a generic, or a specific layout, by instantiating standard components (labels, drawings …) and using values from the object (attributes or method results.) The sub-object `Asimov-Books` allows the user to perform a guided tour of all the books of this author. Notice that it is allowed to manipulate references from the OTHY domain to the application domain. Anchors and references allow us to launch the same process for one book if the user desired so.

Table 1. Some O_2 classes of our example

`class Author`	`class Publication`
`private type tuple(`	`Private type tuple(`
` read FirstName: string,`	` read Title: string,`
` read LastName: string,`	` dateP: Date,`
` dateB: Date,`	` read WrittenBy:`
` dateD: Date,`	`      unique set(Author),`
` read Photo: Image,`	`read EditedBy:`
` read Biography: Text)`	`   unique set(Publisher))`
`method`	`end;`
` public BirthDate: string`	
` public DeathDate: string`	`class Book`
` public Wrote:`	` inherit Publication`
`         unique set(Book)`	`private type tuple(`
`end;`	` read Cover: Image,`
	`end;`

Fig. 7 gives an example of standard presentation with the O_2 browser (`display(Asimov)` or `Asimov->display()`). This is to be compared to a presentation obtained by using our framework, in Fig. 8. The presentation is obtained by a call to `OTHY(BibliograhicDirectory, Asimov)->display()`[2]. The code of this function consists of a single line of code, as follows:

```
  function body OTHY(dir: Directory, inst: Object): Page
  { return (dir->map(inst->title)->transduce(inst)); }
```

The function signature conforms to the mathematical one of section 5.2 except for the DB parameter which is implicit. Though the current presentation is not yet really attractive, its use to navigate in a database is really eloquent. First, the O_2 browser forces to open a new window for each sub-object. All opened windows are stacked and must be closed in reverse order. In contrast, OTHY manages automatically the erasure of windows and allows a return to the previous windows (the history), like a Web browser. This decreases considerably the user concentration, and increases the freedom of move around the objects of the database.

[2] Another version of this function uses a default directory, set for the user session in a global variable (named a root of persistence in O_2.)

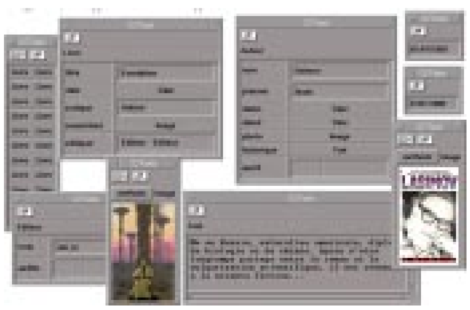

Fig. 7. The standard presentation with the O_2 browser

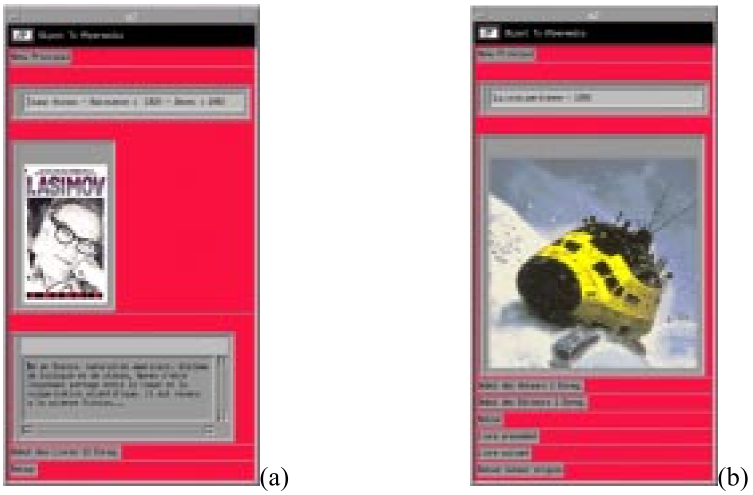

Fig. 8. The OTHY presentation of (a) an author and (b) a book

Note that taking into account methods allows cross navigation between related objects. The O_2 browser relies only on attributes; therefore it is possible to go from a book to its authors but not the reverse way because this role has been implemented by a method.
Another difference is that each attribute is presented in a separate panel with the O_2 browser whereas we are free to mix labels and values in OTHY, e. g., the name followed by constant strings and dates in Fig. 8(a).

Guided tours (Fig. 1) are represented by a next and a previous button (respectively "Livre précédent" and "Livre suivant" in Fig. 8(b) that allow to browse Asimov's books.

Indexed tours (Fig. 2) are not implemented in our example, but they are represented by one (or several) index page(s.) In turn, these pages access to the desired pieces of information. In the case of *Authors*, if one author is very prolix, it is very interesting to group his or her books according to different criteria (date of publication, alphabetic order …) Also not presented here is the possibility to incorporate parts of the related objects into the currently displayed instance page. In fact, since transducers can be derived for very specific needs, their result can be composed of anything that is reachable from the instance parameter[3].

7 Conclusion and Future Work

We saw that there exist needs for large organisations to browse through their heterogeneous information system. Browsing has to be independent both of localisation and of the kind of storage and retrieval systems (object, relational, or object-relational databases, specific systems, even simple files.) Other demands have been made by the users but are not addressed by this paper.

We decided to develop a framework to display objects contained in several databases in the manner of an hypermedia application. This framework is totally independent from the storage sub-systems and from the data models. Localisation independence and hypermedia conviviality is to be achieved through the use of a Web browser.

A first prototype allowed us to validate our approach. Indeed, it is now possible to create an application without taking care of how this application will be browsed. This is very important to decrease the time devoted to the user interface construction (thanks to a default transducer, it can even be reduced to zero.) The OTHY framework integrates the main concepts of hypermedia design (consistency of the presentation, links and anchors, guided and/or indexed tours, history.) However, it is possible to customise our framework by providing additional transducers up to particularising each object presentation. Furthermore, users may choose themselves convenient presentations to obtain particular "views."

We are on the path to develop the V1 prototype. This tool is a Java applet integrating the access layers described in Section 4. It must manage the additional problems of instance interpretation (splitting them into their basic components), user identification and loading of his or her profile. Moreover, a graphical tool to describe pages has to be developed to allow users to customise themselves their views of the information system.

From the theoretical point of view, at least three issues have to be investigated: reverse engineering, querying, and integration. First, our framework is object-oriented whereas the information sub-systems are heterogeneous. Therefore, reverse engineering is mandatory to be able to use the framework for any kind of data. This is currently being added, based on previous works of our research team [26].

The second issue is to incorporate query capabilities. They will be based on information retrieval concepts and techniques, rather than on (almost) formal queries, which were rejected by the end-users in past experiences. In the long term, it is envisaged to integrate also multimedia retrieval capabilities, since they are being studied in our research team [18].

[3] In another application, the visualisation of a class with a single attribute and a recursive relationship leads to the extreme case where the value of the object and the values of all the related objects (used as anchors) were presented in a single window.

The last envisaged issue is schema integration. Effectively, considering the different databases as isolated islands of information is poor practice. The system has to find implicit relationships between the different databases that belong to the information system of the same organisation because they necessarily share common subschemas, objects, or simply attributes. Integration would greatly improve browsing and postpone the moment when user is obliged to query the system.

8 Acknowledgements

We would like to thank Cyril Tiollier, and more particularly Pejman Parandi, engineer students at IRESTE, who implemented part of the V0 OTHY framework in the context of a second year student project. The example of "*Asimov*" is theirs.

9 References

1. Bancilhon, F., Delobel, C., Kannelakis, P.; Building an Object-Oriented Database System: The Story of O_2; Morgan-Kaufmann, 1992
2. Batini, C., Ceri, S., Navathe, S.B.; Conceptual Database Design: An Entity-Relationship Approach; The Benjamin/Cummings Publishing Company, Inc., 1992, 470 p.
3. Bleber, M., Isakowitz, T.; Designing Hypermedia Applications; Communications of the ACM, August 1995, Vol. 38, No. 8, pp. 26-29
4. Carey, M., Haas, L., Maganty, V., Williams, J.; PESTO: An Integrated Query/Browser for Object Databases; Proc. of the 22nd Int'l Conf. On Very Large Data Bases (VLDB'96), Mumbai (Bombay), India, 1996, pp. 203-214
5. Cattel, R. G. G., Barry, D., Bartels, D., Berler, M., Eastman, J., Gamerman, S., Jordan, D., Springer, A., Strickland, H., Wade, D.; The Object Database Standard: ODMG 2.0; Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997, 270 p.
6. Conklin, J.; Hypertext: An Introduction and Survey; IEEE Computer, September 1987, pp. 17-41
7. Fayad, M. E.; Schmidt, D. C.; Object-Oriented Application Frameworks; Communications of the ACM, October 1997, Vol. 40, No. 10, pp. 32-38
8. Garzotto, F., Paolini, P., Schwabe, D.; HDM – A Model-Based Approach to Hypertext Application Design; ACM Transaction on Information Systems, Vol. 11, N° 1, January 1993, pp. 1-26
9. Halasz, F., Schwartz, M.; The Dexter Hypertext Reference Model; Communications of the ACM, February 1994, Vol. 37, No. 2, pp. 30-39
10. Hardman, L., Bulterman, D.C.A., Van Rossum, G.; The Amsterdam Hypermedia Model: Adding Time and Context to the Dexter Model; Communications of the ACM, February 1994, Vol. 37, No. 2, pp. 50-62
11. Isakowitz, T., Stohr, E. ; Balasubramanian, P. ; RMM: A Methodology for Structured Hypermedia Design; Communications of the ACM, August 1995, Vol. 38, No. 8, pp. 34-44
12. Johnson, R.E., Foote, B.; Designing Reusable Classes; JOOP, Vol. 1, n° 2, June/July 1998, pp. 22-35
13. Johnson, R.E.; Frameworks = (Components + Patterns); Communications of the ACM, October 1997, Vol. 40, No. 10, pp. 39-42
14. Kim, W., Banerjee, J., Chou, H.-T., Garza, J. F., Woelk, D.; Composite Object Support in an Object-Oriented Database; Proc. of the ACM Int'l Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'87), 1987, pp. 118-125

15.Kim, W., Bertino, E., Garza, J. F.; Composite Objects Revisited; Proc. of the ACM SIG Int'l Conf. On the Management of Data (SIGMOD'89), Portland, Oregon, 1989, pp. 337-347
16.Losada, B., Lopistéguy, P., Dagorret, P.; Etude de la Conception d'Applications Hypermédias (in french); Actes du XVe Congrès INFORSID, June 1997, Toulouse, France, pp. 133-146
17.Manhes, S.; La réutilisabilité: Patterns et Frameworks (in french); M. Sc. Thesis, IRIN, University of Nantes, 1998
18.Martinez, J., Marchand, S.; Towards Intelligent Retrieval in Image Databases; Proc. of the Int'l Workshop on Multi-Media Data Base Management Systems (MMDBMS'98), Dayton, Ohio, August 1998, pp. 38-45
19.Maurer, H. (ed.); HyperG is now HyperWave: The Next Generation Web Solution; Addison-Wesley Publishing Company, 1996
20.Meghini, C., Rabitti, F., Thanos, C.; Conceptual Modeling of Multimedia Documents; IEEE Computer, October 1991, pp. 23-30
21.Meyer, B.; Object-Oriented Software Construction; Prentice Hall, 1988
22.Nanard, J., Nanard, M.; Hypertext Design Environments and the Hypertext Design Process; Communications of the ACM, Vol. 38, No. 8, August 1995, pp. 49-56
23.Newcomb, S.R., Kipp, N.A., Newcomb, V.T.; "HyTime": The Hypermedia/Time-based Document Structuring Language; Communications of the ACM, November 1991, Vol. 34, No. 11, pp. 67-83
24.Nielsen, J.; HyperText and HyperMedia; Academic Press, Inc., San Diego, California, USA, 268 p.
25.Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.; Object-Oriented Modeling and Design; Prentice Hall, 1991
26.Saoudi, A.; Une approche terminologique pour l'interopérabilité sémantique des systèmes de bases de données hétérogènes; Ph. D. Thesis, November 1997, IRIN, University of Nantes
27.Schwabe, D., Rossi, G., Barbosa, S.D.J.; Abstraction, Composition and Lay-Out Definition Mechanisms in OOHDM; Proc. of the ACM Workshop on Effective Abstractions in Multimedia, San Francisco, California, November 4, 1995
28.Schwabe, D., Rossi, G., Barbosa, S.D.J.; Systematic Hypermedia Application Design with OOHDM; Proc. of The 7th ACM Conf. on Hypertext, Washington D.C., March 16-20, 1996, pp. 116-128.
29.Smith, K.E., Zdonik, S.B.; InterMedia: A Case Study of the Differences between Relational and Object-Oriented Database Systems; Proc. of the Int'l Conf. on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'87), Orlando, Florida, October 1987
30.Van Herwijnen, E.; Practical SGML; Kluwer Academic, 1994