RMCase: Computer-Aided Support for Hypermedia Design and Development

Alicia Díaz

Departamento de Informática
Laboratorio de Investigación y
Formación en Informática
Avanzada(Lifia)
Universidad Nacional de La Plata
also CONICET
50 y 115, 1900 La Plata, Argentina
Email: alicia@info.unlp.edu.ar

Tomás Isakowitz

Information Systems Department
Stern School of Business
New York University
New York, NY 10012, USA
Email: tomas@stern.nyu.edu

Presented at the International Workshop on Hypermedia Design 1995

Montpellier, France

ABSTRACT

We present the design of a computer-aided environment, RMCase, to support the design and construction of hypermedia applications. The environment is based upon the Relationship Management methodology. RMCase supports hypermedia design and development activities. Support for cognitive design processes is achieved through three fundamental premises that form the foundation of RMCase: (1) fluid feedback loops between the various methodological stages, (2) manipulation of objets at the instance level, and (3) lightweight prototyping . To achieve this, RMCase itself is designed as a hypermedia application, where hypertextual navigation implements feedback loops. Instance objects can be cloned and abstraction/instantiation mechanisms are envisioned to facilitate designers back and forth movements between the abstract and the concrete layers of an application. As a result, RMCase will support bottom-up, top-down and middle-out software development styles.

1. INTRODUCTION

Hypermedia design and development is a complex task that involves a variety of activities, at the storage, access and presentation levels. As a consequence, the constituencies participating in hypermedia projects differ from those of traditional software development environments. Hypermedia projects involve content-authors, librarians, musicians and graphic designers, as well as programmers, system analysts, software managers, and, of course, users. Moreover, aesthetic and cognitive aspects, so important for hypermedia applications, are foreign to existing software engineering environments. Thus, there is a need for special methodologies and tools to support the software development process of hypermedia applications.

In this paper we present **the design** of the Relationship Management Case Tool (*RMCase*), an environment to support the development of hypermedia applications. This article represents a continuation of our efforts to construct a CASE tool for the development of hypermedia applications. Earlier efforts resulted in graphical editors and sample showcase applications. RMCase has a dual foundation basis. The Relational Management Design Methodology (*RM*) [Isakowitz 95] provides the methodological foundation for RMCase; its cognitive basis is drawn from work by Nanard and Nanard [Nanard 94,Nanard 95]. RMCase is, in principle, platform independent. It is capable of creating of systems running on the WWW, Toolbook, Hypercard, and other hypermedia environments.

The RM methodology prescribes seven steps to guide hypermedia software development. RM was conceived to be the basis for software development tools. Thus, the completion of each of its seven steps results in well defined software engineering artifacts, which are used as input to subsequent steps. This fundamental characteristic of RM is no different from other CASE tools [Banker 93],[IEF 90], yet RM specifically addresses important aspects of hypermedia application development, not found in traditional software engineering environments, in particular design of navigation mechanisms.

This paper is organized as follows. In section 2 we identify important requirements for a hypermedia software development environment. Section 3, briefly describes the RMD design methodology. The principal components of RMCase are presented next in section 4, and then, in section 5, we elaborate upon RMCase's envisioned prototyping facilities. Finally, section 6 briefly summarizes the contributions we make in this article.

2. REQUIREMENTS FOR A HYPERMEDIA DESIGN ENVIRONMENT

Nanard and Nanard [Nanard 95] identify the following as fundamental requirements for a hypermedia development environment:

I. Fast feedback loop spanning the methodological steps of RM, to facilitate evaluation and re-design activities.

II. Accessible and unconstrained cloning tools at the instance level, to facilitate the generation of material application instances that lend themselves to evaluation by designers, developers and users.

III. Abstraction and instantiation mechanisms that enable developers to alternate between bottom-up and top-down approaches.

As conceived here, RMCase itself is a hypermedia application that uses different "work contexts" to support the methodological stages of RM. We briefly describe how RMCase has been conceived to implement the pre-requisites described above, we will elaborate on some of these aspects of RMCase in subsequent sections.

Graphical editors

The environment we propose is composed of a set graphical editors to build the artifacts pertinent to the RM methodology: E-R, Slice and RMD diagrams, screen designs, etc. These editors are to be interconnected and designed in a consistent manner to facilitate the work of developers.

Fast Feedback Loops

The ability to switch back and forth between methodological stages is a crucial component of the design and development process. Each methodological stage is supported by one such context. Using RM terminology, we can start with any entity, define its slices, its interface, its node-link structure, create an experimental prototype and then go on to the next entity. Feedback loops are implemented by navigating between different work contexts.

Prototyping

A prototype offers designers the ability to experience the application as it would eventually be executed. The prototype-layer contains cloning mechanisms to enable the replication of structures. Thus, designers can reuse design objects from other applications and replicate a basic structure & modify it.

Cloning

Each RMCase context contains “smart” cloning capabilities. By smart we mean that cloning does not happen in isolation, one element at a time. We adopt a holistic approach in which all objects pertinent to the one being cloned are also replicated. For example, cloning an entity within the ER context replicates its complete inner structure. Thus, if slices, access structures and other features have been defined for the entity, they will also be cloned.

Cloning elements at a more advanced methodological stage is also quite interesting, as it enables developers to clone along with an object its complete design structure or parts thereof. For example, when a developers clones a screen in the user-interface context, she can also choose to copy the complete entity wherein the screen originates, and along with it, the screens for all the other slices of the same entity, its access structures, and so on. Of course, developers have control over the granularity of cloning. Thus, it possible to clone complete applications as well as individual elements.

Abstraction and Instantiation mechanisms

These mechanisms allow designers to switch back and forth between the instance level and the conceptual levels of an application, e.g., from E-R designs to HTML pages and vice-versa. We report more fully on this aspect of RMCase in section 5.3.

3. RM METHODOLOGY

RM [Isakowitz 95] is a methodology for the design and construction of hypermedia applications. RM consists of seven steps, some of which can be conducted in parallel. We will briefly explain the RM data model in what follows. For a more detailed elaboration, we refer the reader to [Isakowitz 95]. Although RMD is, in principle, a linear methology, our proposal in this paper will result in an environment that supports feedback loops, cloning and prototyping to achieve a combination of top-down and bottom-up approaches.

3.1 RM Data Model (RMD)

We now describe the Relationship Management Data Model (RMD) which is the cornerstone of the RM methodology. A data model is a set of logical objects used to provide an abstraction of a portion of the “real world.” In our case, RMD provides a language for describing information objects and navigation mechanisms in hypermedia applications. The data model is based on the Entity-Relationship model [Elmasri 90] and on HDM [Garzotto 91], and contains entities, attributes and slices.

ER relationships can be *on-one*, *one-many*, and *many to many* they representing associations among different entity types. As in database modeling, many-many relationships are factored into pairs of one-many relationships.

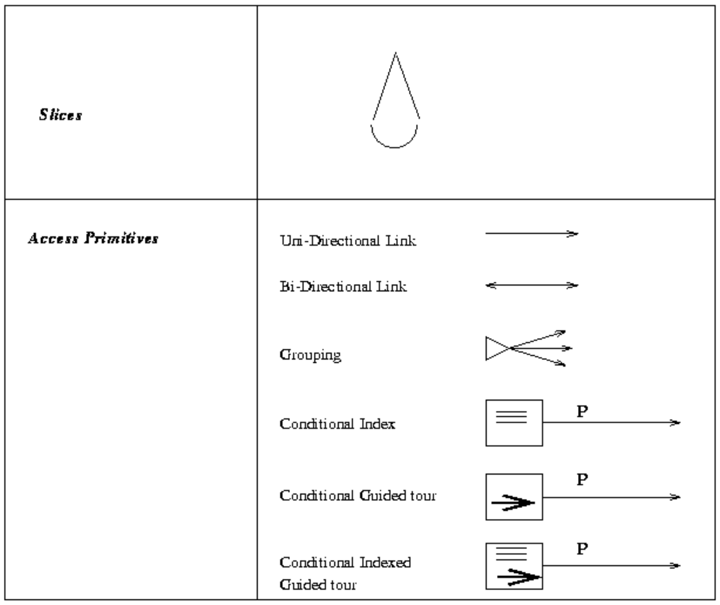

Figure 1: The elements of the RMD Data Model

Because entities may have a large number of attributes of a different nature (e.g., salary information, biographical data, photographs), it may be impractical or undesirable to present all the attributes of an entity instance in one screen. Thus, attributes are grouped into *slice*s. For example, a person entity with attributes *name*, *age*, *picture* and *biography*, may have a *General* slice, containing *name*, *age* and *photograph* and a b*iography* slice, with *name* and *biography*. Hence, each instance of the entity person will have two slices. The notation for slices is shown at the top of Figure1 (it is supposed to resemble a pizza slice!).

Navigation is supported in RMD by the six access primitives shown at the bottom of Figure 1. Uni- and bi-directional links are used to specify access between slices of an entity. The most significant access structures supported by RMD are *indices*, *guided tours* and *groupings*. An index acts as a table of contents to a list of information items, providing direct access to each listed item. A guided tour implements a linear path through a collection of items allowing the user to move either forwards or backwards on the path. The grouping mechanism serves as an access point to other parts of the hypermedia document. For example, the initial screen of many applications contains a menu or set of buttons that provide access to different functions or classes of information.

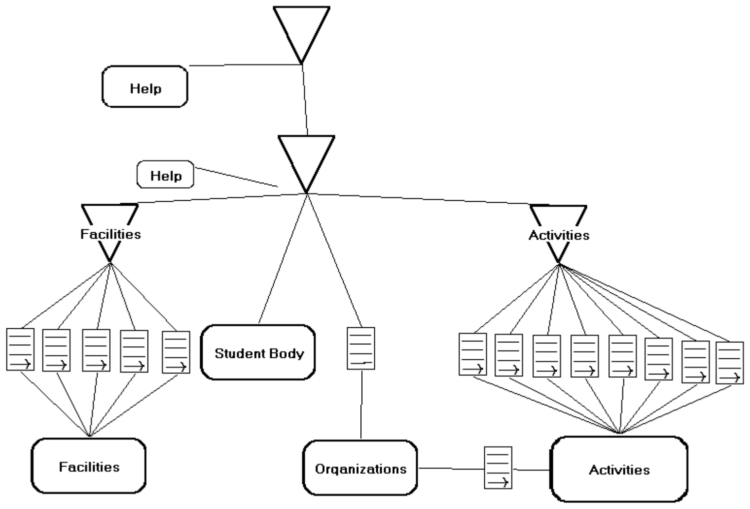

Figure 2: An RMD diagram representing the design of an application about student life in a Japanese university campus

A sample RMD diagram, modeling an a hypermedia tour about student life in a rural Japanese campus, is shown in Figure 2. This application has four entities:

1. *Facilities* available to students, such as gym, cafeteria, library, etc.;
2. *Student Body,* which contains statistical information about the student population and their GMAT and TOEFFL scores*;*
3. *Organizations* to which students belong, such as Graduate Student Organization, MBA Council, etc.; and
4. *Activities* the students engage in, such as public lectures, industrial plant visits, skiing, etc.

The application has only one relationship, *organizes*, that relates *organizations* to *activities,* by specifying which organization sponsors what activity.

4. CONTEXTS IN RMCASE

Each step in the RM methodology handles design objects (e.g., entities, relationships, attributes, slices, access structures, etc.) in particular ways. Yet design objects are not exclusive of any particular stage. For example, entities and attributes are used in the E-R design and in the slice design stages. In the design process the design objects evolve through the different RM stages, and follow an evolutionary trajectory, as part of the software development process. It is thus possible to trace these objects through the sequence of gradual transformations that span a continuum stretching from the conceptual level of E-R design, to the concrete level of object code.

From a cognitive point of view, developers continuously switch back-and-forth between different points in the trajectory of a design object. To support these cognitive processes RMCase includes facilities to rapidly move between methodological stages, while keeping the focus on the same design object. This kind of navigation allows developers to easily alternate between conceptual and concrete levels, e.g., from an E-R diagram to a HTML implementation, and vice-versa.

4.1 RMCase contexts

RMCase provides support for RM stages via special constructs called *contexts*[1]. The concept of context is akin to that presented in [Casanova 91] and to *collections*, as defined in [Garzotto 94], in the sense that a context is a collection of nodes with associated browsing semantics.

DEFINITION: An *RM context* is a hypermedia application represented via an RMD diagram.

RMCase supports the software life-cycle of an application with a set of contexts, one per stage. Rapid transitions between methodological stages are supported in RMCase via hypertextual navigation among contexts. Since design objects are shared among different contexts, this kind of navigation enables developers to focus on one or more design objects while moving back and forth between the various stages in the methodology. (Recall that this was one of the three fundamental requirements for a hypermedia software development environment.)

[1] Note that in this paper we do not use RM contexts to model applications, but to model the different working contexts that co-exist in the design and development of hypermedia applications.

Since RMCase itself is a hypermedia application, we can, and will, model it using RM. Each context is itself a hypertext, that has design objects for nodes and relationships between design objects as links.

4.1.1 The E-R Design Context

The E-R design context, depicted in Figure 2, facilitates the construction of E-R diagrams, capturing the characteristics of the application domain. Three basic design objects are handled in this context: entities, attributes and relationships.

- An *entity* is a conceptual element from the application domain, characterized by a set of attributes.
- An *attribute* represents a unit of information. Attributes have a name, an type and are always associated with a unique entity or relationship.
- A *relationship* is a conceptual tie among two or more entities. A relationship has cardinality be one-one, one-many, many-one or many-many (RM splits many-many relationships into two one-many relationships. Relationships can also contain attributes.

The E-R design context has well-defined functionalities: to manipulating entities, attributes and relationships between entities. Besides the common create/delete operations, there are operations to split an entity into two or more entities and to merge several entities into one. This functionality becomes crucial as designers return to the E-R design context via feedback loops.

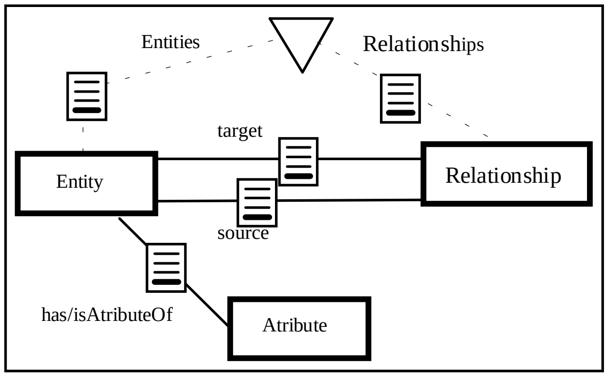

Figure 3: The E-R context implements E-R design activities

Figure 3 presents an RMD model of the E-R context. We observe that a designer can navigate among an entity and its attributes via the relation *has/isAtributeOf*. The *target* and *source* indices are used to navigate among the entities and the relationships where its is involved. There is also a grouping that provides access to all the entities and relationships defined during this stage. This context manipulates and produces an E-R diagram that is used as input for the next methodological step: slice design.

4.1.2 The *Slice Design* Context

Here the designer states the inner structure of an entity from a navigational point of view. Defining slices and links among them is tantamount to defining a navigational structure within an entity. The design objects are: entities, attributes, slices and links among slices. A *slice* is a set of attributes belonging to a given entity. Any attribute could be part of more than one slice. Each entity has a distinguished slice, called the entity's *head*, that is to be used as a default entry point for incoming access constructs. The slice design context generates an "exploded" version of an E-R diagram, which is called an *E-R+ diagram*. This context's functionality includes creation and deletion operations for its design objects, as well as splitting and merging of slices, and selection of entity heads.

Operations at this level may impact the E-R design context. For example, merging slices belonging to two different entities can result in splitting the two original entities into three: one to include the attributes of the merged slices, the other two containing the set differences between the two original entities and the set of attributes in the merged slice[2]. The reciprocal also holds, namely operations in the E-R design context can affect the slice design context. For example, splitting an entity may force a splitting of its slices.

4.1.3 The *navigational design* context

Here developers specify the navigational features of an application. The navigational context also manipulates slices, entities and relationships. In this context designers create menu-like structures using groupings, and other access paths using indices and guided tours. Designers also decide here what relationships will be navigable in the final application. The selected relationships are outfitted with RM access structures. For example, a *teaches* relationship between a faculty member and the courses s/he teaches, may be implemented via a conditional index. The designer also specifies the

[2] However, if all slices of two entities are merged, so are the entities themselves.

conditions that are part of the access structures (e.g., “teaches(Course)=last_name” or “rank=associate”).

Since all RM access structures interconnect entities, it is necessary to specify the specific entity slice that a user will encounter when traversing an access structure. By default, entity heads (defined during slice design) act as destination slice for *all* incoming access mechanisms. Yet, this context provides designers with the opportunity to indicate alternative destination slices, individually, for each occurrence of an access structure.

4.1.4 *Node-link conversion* context

The *node-link conversion* context contains three kinds of facilities to support the conversion of RMD diagrams into node-link webs:

1. Facilities to edit the rules that specify how to convert RMD diagrams into node-link webs. These facilities are to be used mainly by the developers of RMCase itself.
2. Facilities to execute the conversion itself, a kind of “compiler”. When activated, these facilities automatically generate a web of nodes and links.
3. Facilities to manipulate the node-link web itself.

Each slice is mapped into a node, and each access structure into a corresponding set of links and nodes. Links between slices are passed along to the node-link web. In addition, anchors are defined for each outgoing link, including guided tours and indices. In an index, the anchor provides navigation into the item it denotes. In a guided tour, the first anchor selects the first element in the guided tour, the next anchor the second, and so on.

This context is conceived as an automated process, which we incorporate into RMCase mainly for browsing and debugging purposes.

4.1.5 *User-interface design* context

This context provides facilities to design and edit the user-interface. We conceive a set of screen-designing tools like those available in many commercial applications (e.g., Toolbook, Visual Basic, MS Access, etc.) Anticipating different deployment platforms, this context enables designers to assign multiple user-interface designs for each node, such a Toolbook or HTML pages. The appropriate design will be picked at construction time to generate the object application. To assist developers in the design of good user-interfaces it is possible to incorporate design principles such as those advocated by Kahn et al. [Kahn 94] into RMCase .

We contemplate the following set of functionalities for the user-interface context:

a) associate to each node an interface object

b) associate to each anchor an interface object

c) define visual effects to indicate link traversal, e.g., venetian blinds, iris effect, etc.

In sum, the user-interface context assists developers in the generation of a set of platform specific templates. These templates are to be populated with information in the *hyperbase population* context.

4.1.6 *Hyperbase Population* Context

This context supports the generation of an application by populating entities, slices, screen templates, etc. with instance data. For example, a database may be used to generate a collection of HTML pages. If the information resides within a database, there are at least two alternative approaches for hyperbase population.

a) *Pre-populated applications*. The database information is “pumped” into a set of node instance at generation time. Thus, the data is hard-coded into the application. As a result the data can only be updated by manually with the assistance of software developers. The pre-populated approach is recommended for applications that are updated on an infrequent basis, e.g., twice a year.

b) *Dynamic applications.* The hypermedia application obtains information “on demand” from the database by issuing queries. In this case, the database is used to update the data. The dynamic approach is recommended for applications that have high volatility data.

By definition, this context is responsible for establishing a kind of bridge between the hypermedia application and information that is external to it. This is a rather complex task that requires a special kind of system, a Relationship Management System, which will be reported elsewhere.

4.2 Inter-context navigation

Navigation between contexts supports rapid feed-back loops within methodological stages. It enables designers and developers to switch back and forth among different views of the application objects.

Navigating from a context to another means making the new context the current one. Focus preservation is an essential characteristic of inter-context navigation. For example by positioning the cursor on an entity in the E-R design context and selecting a "navigation to slice design" action, a designer will be transported into the slice design context. The focus will be on the set of slices corresponding to the originating entity. Similarly, navigation from the slice design context to the user-interface design context shifts the focus from a slice to the screen templates associated with it.

5. PROTOTYPING

As the development activity progresses, developers continuously move back and forth between abstract structures representing the application to specific application instances. At the instance level, application components can be objectively evaluated, subsequently triggering redesign and re-construction activities. Thus, not only do developers need tools that support feedbacks between the methodological stages, they also need mechanisms to move easily and rapidly between the abstract and instance levels [Nanard 95].

A prototype is useful because it provides enough feedback on the design process without involving a through completion of all development tasks. We envision within RMCase the ability to construct prototypes at any stage during software design and development, even (actually "specially") from incomplete designs. The converse, constructing a design from a prototype supports a bottom-up approach to software development.

5.1 Prototype objects

Prototypes are partial implementations that may cover as little as a single screen, and as much as a preliminary version of the whole application. Prototypes usually contain dummy information and lack many of the application maintenance features. However, even with these limitations, prototypes are useful to test design concepts, to demonstrate an application to users and, overall, to provide important feedback for the software development process.

5.2 The Prototyping context

There a specialized *prototyping* context that represents the instance level. The prototyping context is where software developers can test different aspects of the design of a hypermedia application, such as its information structure and navigation patterns, as well as implementation aspects of the application such as populated screen templates. Although, strictly speaking, the prototyping context does not share design objects with any of the other contexts, here is a close relationship between a prototype and the remaining design objects.

The prototype context provide the following functionality:

- testing capabilities, i.e., fill in specialized data to experience it on the computer, navigate access structures, etc.
- cloning capabilities that enable a designer to replicate parts of an application. The granularity of the cloning operation is adjustable. Hence a designer can also clone design artifacts, such as the RMD-diagram along with an object at the prototype level.

Besides providing these functionalities, the importance of the prototyping context resides in its connectivity to the other contexts. Thus a developer can navigate from the ER context directly to a prototype. A by-product of such a transition is the generation of default slices, screen-layouts, dummy data and the remaining elements necessary to create a prototypical instance of the entity. Developers and users can then evaluate and modify the prototype. Any changes are propagated back into the conceptual level, for example, by adding an attribute to an entity or a new relationship.

As Figure 4 shows, the prototype context has a special relationship with the remaining contexts. It is connected to all of them, and navigation from prototyping carries along important transformations.

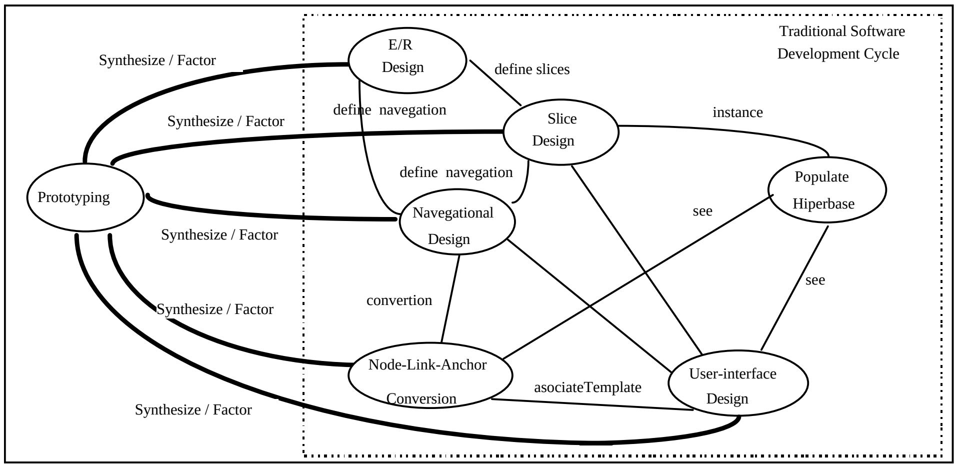

Figure 4: The prototyping context plays a distinguished in RMCase. Navigation form/to this context support both top-down and bottom-up software development approaches. The resulting process is thus an “middle-out” design environment.

6. CONCLUSION

We have presented the design of a computerized environment, RMCase, to support the design and development of hypermedia applications. Not only does the environment follow a methodology (RMD) but it takes into consideration cognitive aspects of hypermedia software development. RMCase is meant to support developers and designers in their evolutionary cycle of experimentation, building, and re-shaping. Moreover, we have identified prototype as an important component of this process, and we envision adequate support for such functionality. As the interface becomes the most common of working environments for users, we also anticipate mechanisms that will allow users to work backwards from a user-interface to a structured design. The main benefits of our approach are that it will facilitate the work of designers and developers of hypermedia applications while simultaneously enhancing the quality of their products. We foresee immediate applications in the realm of WWW and other hypermedia application platforms (Toolbook, Hypercard, Microcosm).

7. REFERENCES

Banker 93 R. D. Banker, T. Isakowitz, R. J. Kauffman, R. Kumar and D. Zweig, "Tools for managing repository objects' ' in Analytical methods for software engineering economics II, P. T. Geriner, T.

Gulledge and W. P. Hutzler, Eds. New York: Springer-Verlag, 1993.

[Casanova 91] M. Casanova, L. Tucherman, J. L. Rangel Neto, N. Rodriguez, L. Soares, "The Nested Context Model for hiperdocumentos", *Hypertext'91 Proceedings*, ACM Press, 1991.

[Elmasri 90] R. Elmasri and S. Navate. *Fundamental of Database Systems*. The Benjamin/Cummings Publishing Company, second edition, 1990.

[Garzotto 94] F. Garzotto, L. Mainetti, P. Paolini, "Adding Multimedia Collections to Dexter Model", in *ECHT94 Proceedings*, ACM Press, 1994.Garzotto 94

[Garzotto 91] F. Garzotto, P. Paolini, D. Shwabe, "HDM - A Model for the Design of Hypertext applications", in *Hypertext'91 Proceedings*, ACM Press, 1991.

[IEF 90] *Information Engineering Facility* (IEF) technology overview, 2nd edition, TI Part # 2739900-8027, Nov. 1990

[Isakowitz 95] T. Isakowitz, E. A. Stohr and P. Balasubramaninan, "RM: A Methodology for Structured Hypermedia Design", *Communications of the ACM*, August 1995.

[Kahn 94] P. Kahn, R. Peters and G. P. Landow, "Three Fundamental Elements of Visual Rethoric in Hypertext". *In Designing User-Interfaces for Hypermedia*, W. Schuler, J. Hannemann, N. Streitz, eds., Springer-Verlag, 1994.

[Nanard 94] J. Nanard, M. Nanard, "Key features in a hypertext design environment for supporting the process of hypertext structure design", in *Methologies for designing and developing hypermedia applications*, collection edited by T. Isakowitz and M. Thüring, September 1994.

[Nanard 94] J. Nanard and M. Nanard, "Hypertext Design Environments and Hypertext Design Process", *Communications of the ACM*, August 1995.