HDM — A Model-Based Approach to Hypertext Application Design

FRANCA GARZOTTO, PAOLO PAOLINI,
Politecnico di Milano
and
DANIEL SCHWABE
Pontifícia Universidade Católica do Rio de Janeiro

Hypertext development should benefit from a systematic, structured development, especially in the case of large and complex applications. A structured approach to hypertext development suggests the notion of *authoring-in-the-large*. Authoring-in-the-large allows the description of overall classes of information elements and navigational structures of complex applications without much concern with implementation details, and in a system-independent manner. The paper presents HDM (Hypertext Design Model), a first step towards defining a general purpose model for authoring-in-the-large. Some of the most innovative features of HDM are: the notion of *perspective*; the identification of different *categories of links* (structural links, application links, and perspective links) with different representational roles; the distinction between *hyperbase* and *access structures*; and the possibility of easily integrating the structure of a hypertext application with its browsing semantics. HDM can be used in different manners: as a modeling device or as an implementation device. As a modeling device, it supports producing high level specifications of existing or to-be-developed applications. As an implementation device, it is the basis for designing tools that directly support application development. One of the central advantages of HDM in the design and practical construction of hypertext applications is that the definition of a significant number of links can be derived automatically from a conceptual-design level description. Examples of usage of HDM are also included.

Categories and Subject Descriptors: H.2.1 [**Database Management**]: Logical Design—*data models*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software; H.4.1 [**Information Systems Applications**]: Office Automation; 1.7.; [**Text Processing**]: Miscellaneous—*hypertext*

General Terms: Design, Languages

Additional Key Words and Phrases: Derived links, HDM, Hypertext design models, Hypertext applications, Hypertext structures, models

This work was sponsored in part by the Commission of the European Communities, in the scope of the HYTEA Project (P 5252) of the ESPRIT II Program. D. Schwabe developed this work while on sabbatical leave from PUC and was partially supported by CNPq-Brasil.

Authors' addresses: F. Garzotto and P. Paolini, Department of Electronics, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy. Phone: +39-2-23993520; fax: +39-2-23993411; email: garzotto@ipmel1.polimi.it; paolini@ipmel1.polimi.it; D. Schwabe: Departamento de Informática, Pontificia Universidade Católica do Rio de Janeiro, R. M. de S. Vicente, 225, CEP 22453, Rio de Janeiro, Brasil. Phone: +55-21-274 4449; fax: +55-21-274 4546; email: pucrjdi@inf.puc-rio.br.

1. INTRODUCTION

1.1 Background

The degree of success of a hypertext application is directly related to the author's ability to capture and organize the structure of a complex subject matter in such a way as to render it clear and accessible to a wide audience. To control the potential explosion of the number of links, a hypertext application does not really interconnect everything, but rather tries to *directly* interconnect only the most important and meaningful parts of the information so as to convey the overall meaning in a more natural way.

In a rational design approach, a hypertext application developer faces at least two different (but strongly correlated) task categories: "global" tasks, such as defining overall classes of information elements and navigational structures of applications, and "local" tasks, such as filling in contents of nodes. This is very similar to what happens when developing a highly modular software system: designing the topology and the interconnections among modules is different from writing the code for the content of the modules themselves. By analogy with software engineering, we use the following terminology: *authoring-in-the-large*, to refer to the specification and design of global and structural aspects of the hypertext application, and *authoring-in-the-small*, to refer to the development of the contents of the nodes [16, 17].

Authoring-in-the-small is very much related to the specific application area, while authoring-in-the-large has common characteristics across many different applications in a given application domain. Authoring-in-the-small is strongly dependent on the tools used for implementation and on the medium used for storing information (e.g., putting the text in a node is much different than putting an animation, or a sound, or a picture). Authoring-in-the-large can be at some extent independent from these aspects. However, authoring-in-the-large and authoring-in-the-small are typically very interwoven, and influence each other, much more than programming-in-the-large and in-the-small in the current software engineering practice.

This paper presents a model for authoring-in-the-large, named HDM—Hypertext Design Model [16, 17, 18, 36]. HDM prescribes the definition of an application *schema*, which describes overall classes of information elements in terms of their common presentation characteristics, their internal organization structure, and the types of their mutual interconnections. A schema, therefore, captures semantic and structural regularities in the representation structures for a given class of applications. Once a schema has been specified, the model also allows it to define a particular application, by providing primitives to describe a *schema instance*, i.e., actual *instances* of information classes and of connection types. In defining a schema instance, a significant number of connections can be left implicit, since they can be automatically derived from a conceptual-design level description.

HDM is mainly a modeling device. It provides ways of describing, concisely and in a system independent manner, existing or to-be-developed hypertext applications; it helps the author to conceptualize a given application without too much regard to "implementation details"; and it can be used as a communication language between designers, implementors, and users.

Additionally, HDM can be used to generate running implementations of hypertext applications [26, 36]. From this perspective, the model is a first step towards the development of application generators, in a similar way as CASE tools are used in software engineering environments. Implementation of an HDM application requires the definition of a browsing semantics, which specifies dynamic behavior and visualization properties of HDM representation structures. HDM is not closed with respect to any particular browsing semantics. So far, we have specified a *default browsing semantics*, more suitable for relatively simple card-oriented classes of systems.

1.2 Current Approaches to Hypertext Application Design

In the very closely related data base design field, models have played a crucial role in changing the data base development task from being a "handcrafted" (and sometime inconsistent) activity into becoming a structured and rational process, based on well defined design methods [46]. Database models were born as means to define useful abstractions on large amounts of raw information ("logical" data models) and to express the intrinsic application oriented data semantics ("conceptual" and "semantic" data models [6, 21]). However, the peculiarities of hypertext (e.g., the role of links, the complexity of structures, the multimedia facility, the navigation paradigm, etc.) require the development of brand new models, specific for peculiar features of hypertext.

In the hypertext field, some approaches attempt to explicitly model the semantics of specific application domains by adopting predefined representation structures which reflect such "deep" semantics. g-IBIS [11], for example, is a hypertext tool for exploratory policy discussion. It helps capturing, storing, and retrieving the large amount of informal information which express the rationale of a system design process. g-IBIS explicitly models the semantics of its domain by assuming a well defined theory of the design process and by providing a set of specific node and link types that represent conceptual objects in the domain model. Thus g-IBIS encourages to share this model while seeking to discourage less disciplined argumentation modes.

Other works should be regarded more as "system" oriented models than as application oriented design models. They are attempts to define in an unambiguous, rigorous way some important abstractions found in a wide range of existing (and future) systems rather than of existing (and future) applications. The goal of the Dexter Hypertext Reference Model [20] is to serve as a standard against which to compare and contrast the static information structures and the functionalities of different hypertext systems. Its building blocks for defining the static aspects of a hypertext system are low level objects: nodes, links, and anchors. The Dexter model purposefully does not model the content and structure within the components of hypertext networks. It treats them, as well as their layout and visualization properties, as being outside the model per se. Garg's set-theoretical model [14] is a formalization of hypertext networks viewed as static, syntactic structures, and provides a mathematical framework to define abstraction mechanisms useful to describe or derive static syntactic properties of hypertext networks. Garg's model (as other similar ones [35]) assumes low level objects as building blocks, such as nodes (and even node components), and node-to-node links. The Trellis model [39] is mainly a "behavioral" model for hypertext. Hypertext networks are modeled as Petri nets, and various browsing semantics (that is, how information is to be visited) are discussed in terms of Petri nets computations.

Tompa [41] adopts a hypergraph formalism to model generic hypertext structures, to formalize identification of commonalties in these structures, and to directly refer to "groups of nodes" having a common link semantics. A dynamic behavior is also specified through the notion of node markings, with an end result much similar to the Trellis model.

Other, less formal, approaches emphasize preferred topological structures as building blocks to create the structure of hypertext networks. This requirement is often embodied in a concrete system implementation, and enforced by the editing tools the system provides. In Hypercard [3], for example, linear structures play a central organization role. Even if each card can be arbitrarily linked to any other card, each Hypercard node must belong to a sequence ("stack") of cards, and links to the successor node, to the first and the last node in the sequence are automatically provided by the system. Guide [7] also prescribes the extensive use of linear structures to clarify the organization of hyperdocuments. KMS [2] prescribes the extensive use of hierarchical structures to organize information, in order to encourage a top-down, step-wise design of hyperdocuments. Each KMS node (frame) belongs to a single hierarchy; however, KMS also allows "special links" that are cross-hierarchical and induce a more complex topology on the network of nodes.

Several existing approaches provide "template" facilities to help designers generate hypertext structures more easily and systematically, by allowing authors to create multiple copies of individual structures all sharing a number of common properties. HyperCard, for example, has "backgrounds" as the lay-out and linking template. The lay-out and button properties of a background are inherited and shared by all the different nodes sharing that background. NoteCards' notion of (hierarchy of) node types and link types lets writers create as many instances of a class of hypertext nodes as they require [19]. Object Lens [28] encourages a similar style of node creation, by providing an object-oriented environment which allows definition of templates corresponding to high-level abstractions in an application domain. IDE [27], an extension of NoteCards, is an interactive design and development system which assists designers with the process of creating complex hypertext material, mainly (but not only) for instruction purposes. IDE provides built-in representation primitives for describing the substance of a course and the rationale for the course design in terms of hypertext structures. IDE moreover, provides a number of domain independent mechanisms ("structure accelerators") that facilitate the rapid and accurate creation of regular network patterns in hypertext by permitting authors to create entire webs of nodes and links in a single operation. The template facility in Intermedia [38, 48] plays a similar role as IDE structure accelerators. Stotts et al. [40] introduce the notion of style templates to denote styles of structuring and browsing Trellis-based hypertext. Different style templates correspond to different translations from generic authoring notations of hypertext content elements and linkages into structured Trellis documents with a specific browsing behavior. Translations are basically mappings from string-grammars (the authoring language) to graph grammars (the abstract language for Trellis documents).

Other approaches help to organize and modularize existing hypertext material. Trigg's Guided Tours and Tabletops [29, 42], for instance, are extensions of NoteCards which help designers to group and visualize existing hypertext material in a way that it is more appropriate for a particular reader, and to provide preferred paths through a hyperbase.

The remainder of the article is divided as follows: Section 2 discusses the advantages of the model-based approach to authoring in the large and presents HDM in detail; Section 3 gives an example of HDM use in hypertext application design; Section 4 draws the conclusions, by comparing HDM to other approaches, by evaluating the model's utility on the basis of a number of experiences done so far, and by outlining our future work.

2. A MODEL-BASED APPROACH TO AUTHORING-IN-THE-LARGE

2.1 Motivations

There are many advantages in having a design model for hypertext applications, which we summarize here.

Improvement of communication. A design model provides a language which can be used by an application analyst to *specify* a given application. Thus it facilitates the communication between the analyst and the end user; between the analyst and the system designer; and between the system designer and the implementor. At the very least, a basis for discussing the similarities of hypertext applications exists. To paraphrase Halasz et al. [20]: *Hypertext Application Models* can be regarded as an attempt to provide a principled basis for answering questions such as "what do hypertext applications such as Voyager's *Beethoven's 9th Symphony* [47], Harvard University's *Perseus* [30], ACM's compilation *Hypertext for Hypertext* [37], Eastgate's *Elections of 1912* [4], have in common?" "How do they differ?"

Development of design methodologies and of rhetorical styles. Design models provide a framework in which the authors of hypertext applications can develop, analyze, and compare *methodologies* and *rhetorical styles* of "hyper-authoring", at a high level of abstraction, without having to resort to looking at the detailed contents of units of information or at their particular visualizations.

Reusability. As a matter of fact, the availability of a modeling language paves the way for (partial) reuse of the back-bone structure of applications, since model-based specifications capture the "essential semantics" of applications, and can therefore be reused when the semantics of two applications are similar enough.

Providing consistent and predictable reading environments. It is clear that tools for specifying hypertext structures can help authors to avoid structural inconsistencies and mistakes [8], and that applications developed according to a model will result in very consistent and predictable representation structures. As a consequence, navigation environments will also be predictable, thereby helping readers to master complex documents and reducing the disorientation problem [31, 33, 43].

Use by design tools. Design models are the basis for the development of *Design Tools* [45] that support a systematic, structured development process, allow the designer to work at a level of abstraction which is closer to the application domain, and provide a systematic translation process to the implementation level.

2.2 HDM Primitives

HDM is a model to describe hypertext applications. It has a number of peculiarities that make it overall different from other models, but at the same time borrows many of its detail features from existing models. From a terminological point of view, we had two extreme choices: either to create a set of brand new terms for all the features of the model, or to use only preexisting terms (and concepts), perhaps reassembled in a novel fashion. We have chosen a compromise course: some of the terms are new, others are taken from preexisting models in the database or hypertext field. Preexisting terms should be understood with the caveat that the reader should never assume that our use of the term exactly corresponds to the notion that he is already aware of.

An HDM application consists of sizeable structures of information chunks called *entities*. An entity denotes a physical or conceptual object of the domain. Entities are grouped in *entity types*. An entity is the smallest "autonomous" piece of information which can be introduced or removed from an application. In this context, "autonomous" piece of information means that its existence is not conditioned by the existence of other information objects. In HDM, only entities are autonomous, while components and units (see below) are not, as discussed in the following sections.

An entity is an hierarchy of *components*. Components are in turn made of *units*. Each unit shows the content of a component under a particular *perspective* (see proper subsection for details and examples). Entities therefore derive their information content from their components, which in turn derive it from their units. Units are the smallest chunks of information which can be visualized in an HDM application, and they have a lot in common with the standard notions of hypertext "nodes."

HDM information structures can be interconnected by *links*. HDM distinguishes among three categories of links. *Structural links* connect together components belonging to the same entity. *Perspective links* connect together the different units that correspond to the same component. *Application links* denote arbitrary, domain dependent relationships and connect together components and entities, of the same or different types, in arbitrary patterns set up by the author. Application links are grouped in *link types*. All perspective links, and most structural links, do not need to be defined explicitly by the author, since they can be *derived* automatically from the structure of entities. Some application links can be derived by exploiting semantic properties (e.g., symmetry and transitive closure) of the corresponding domain relationships.

As any other design model, HDM makes a clear distinction between the notion of *schema* and the notion of *instance of a schema*. A schema is a collection of type definitions that describe an application at the global level; an instance of a schema is a collection of entities, components, units, and links that satisfy the definitions of the schema. *Outlines*, i.e., access structures, provide reader-oriented entry points to directly access information structures in an instance of a schema.

A *browsing semantics* has the purpose of specifying how information structures are visualized to the reader, and how he can navigate across them. HDM provides a relatively simple *default browsing semantics* as built-in; different browsing semantics however could be defined, to describe more sophisticated visualization effects for applications specified with HDM.

2.2.1 *Entities and Entity Types.*

An entity is a (relatively large) structure of information which represents some real world object of the application domain; a law (say "Law 19/8/89"), a musical Opera (say Verdi's "La Traviata"), a piece of equipment (say "electric engine") could be examples of entities.

Entities are naturally grouped in *entity types*, which correspond to classes of objects of the real world. "Law", "Musical Opera" and "Equipment" could be examples of entity types.

The notion of entity (and the related notion of entity type) is commonly recognized as a suitable notion to model information in the data base field. The most popular design model, the Entity-Relationship (E-R) model [10], and the several other related models derived from it, are based upon the notion of entity and entity class.

We should warn the reader, however, that although we have kept the term and the basic nature of the concept of Entity, in practice HDM entities are very different from E-R entities. For example, HDM entities have complex inner structures (with links inside, see Subsection 2.2.7), while E-R entities are essentially flat; HDM entities have a (default) browsing semantics associated with them (see Subsection 2.4.1); HDM entities can be interwoven, via application links (see Subsection 2.2.8), in patterns much more complex than those allowed by relationships in the E-R model.

2.2.2 *Components.*

An HDM entity is a collection of *components* arranged in a tree-like fashion (ordered hierarchy). A component is an abstraction for a set of units (see Subsection 2.2.4), which are the actual containers of information, and derives its content from its units[1]. A component, being part of a hierarchy, in general has a *parent* (but for the root component), a number of *siblings* (which are arranged in a linear order) and a number of *children* (but for the leaf components). Components "inherit" their type from their entity and can only exist as part of an entity; in this sense, they are not autonomous.

Examples of components, for the entities introduced in the previous subsection, could be "Article 1" (component of "Law 19/8/89"), "Article 1—Subsection 1.2" (child of "Article 1" component), "Ouverture" (component of "La Traviata"), "condenser" (component of "electric engine").

Many authors have observed that hierarchies are very useful to help user orientation when navigating in a hypertext [1, 7]. HDM recognizes this via the notion of entities made up of components organized into hierarchies. Hierarchies can be induced by many semantic criteria (i.e., domain relations). The "Is-part-of" relation, for example, is one of the most commonly used. HDM, however, does not acknowledge the (possibly) different semantic nature of different hierarchies, since it is not intended to be a semantic model. In HDM, a hierarchy is a purely syntactical device to organize information in regular building blocks within a (possibly large) application.

2.2.3 *Perspectives.* In hypertext applications it is often the case that the same topic must be presented in several alternate ways. There are several different reasons why this need may arise. In multinational applications, for example, the same topic must be represented with different languages (e.g., Italian, Portuguese, English, etc.). In educational applications, it may be useful to present the same topic using different rhetorical styles (e.g., discursive, synthetic, schematic, . . .), according to different needs of the readers. In recent applications, very often different media can be used for presenting the same piece of information (text, graphics, image, video, sound, etc.).

In HDM this notion of having different presentations for the same content, is captured by the concept of *perspective*. If an entity has two possible perspectives (say "Italian" and "English", or "Score" and "Sound"), all the components belonging to it also have two possible perspectives. The set of perspectives are shared by all entities of a given entity type, and are defined at entity type level.

Perspectives are just a syntactical device to organize information. HDM does not acknowledge any predefined semantics behind this concept, and does not interpret it. The author has to decide when and how to use the notion of perspective, and what is its intended meaning. HDM only prescribes consistency in using the different ways for presenting information in entities of the same type.

[1] For model simplicity, and also because we have verified that so far we do not need the feature, entities cannot be used as components of other entities

2.2.4 *Units.* A *unit* corresponds to a component associated with a specific perspective. A unit is characterized by a *name* (its identifier) and a *body*. Bodies of units are the place where the actual content of an HDM application is filled in. "Ouverture/sound", "Ouverture/score" (of the entity "La Traviata"), "Article 1/Official Text", "Article 1/Annotated Text" (of the entity "Law 19/8/89"), "assembly instruction/Italian-text", "assembly instruction/English-text", "assembly instruction/Italian-graphics", "assembly instruction/ English-graphics" (of the entity "Electric Engine"), are examples of units.

If the reader tries to relate HDM units to traditional hypertext notions, units, roughly speaking, may be thought as corresponding to the standard notions of "nodes". This correspondence is not completely true; for example, units can be created only in the proper context (of components and entities, under the proper perspectives), and arbitrary nodes are not allowed in HDM.

Filling in a body, corresponds to *authoring-in-the-small* (in our terminology), which is purposefully outside the scope of HDM. Therefore HDM units are the border-line between authoring-in-the-large and authoring-in-the-small: identifying the units and putting them in the proper context belongs to authoring-in-the-large; filling the bodies of units with actual content is authoring-in-the-small.

In HDM, different units are allowed to share their body. This is, methodologically speaking, not encouraged in HDM; in fact, sharing of bodies implies that the same content can be accessed in different (not mutually exclusive) contexts, and this is a typical source of disorientation for the reader. Additionally, we have verified that the need for sharing of bodies arises, quite often, from a poor design of the schema and a bad use of application links (see Subsection 2.2.8). Still we have introduced the notion of body sharing since we have acknowledged that, in some applications, a good use of it can simplify the application specification for the designer (but at the possible expense of clarity for the reader).

2.2.5 *Links in General.* Links in hypertext have a twofold role: a representational role (i.e., capturing domain relations) and a navigational role (i.e., capturing navigation patterns). Sometimes these two different purposes are consistent with each other, sometimes they can be at odds. It may happen that domain relations are not suitable for navigation, i.e., they induce navigation patterns that are not relevant for an application, while, on the contrary, useful navigational links may have vague semantic meaning. Definition of links is therefore a trade-off between representational and navigational goals. However, the more the meaning of a link is made explicit and approximates a relationship of the application domain that a reader is aware of, the more the same reader will be at ease in using the hypertext, since links will evoke familiar associations.

HDM is not intended to be a semantic model. However, HDM acknowledges three categories of links: *perspective links*, *structural links*, and *application links*. The distinction among the three classes simplifies the job of the application designer, induces a consistent use of most links, creates more predictable navigation patterns, and finally makes it possible to introduce additional features such as derivation of links (see Subsection 2.3) and definition of a default browsing semantics (see Subsection 2.4.1).

2.2.6 *Perspective Links.* *Perspective links* interconnect units corresponding to the same component. Perspective links will connect, for instance, the units "assembly instruction/Italian-text" and "assembly instruction/Italian-graphics", thus allowing an Italian reader to switch from the text description of an assembly operation to the picture (in Italian) that schematizes it.

Perspective links are navigationally very simple for the reader, since activating a perspective link leaves the current topic (i.e., the reader's focus of attention) unchanged.

2.2.7 *Structural Links.* *Structural links* connect components belonging to the same entity. There are several different structural links, each of them corresponding to a relationship induced by the ordered tree structure of entities. Examples of structural links are "*Up*", connecting a component to its parent, "*Down*", connecting a component to its children, "*Down-1*", connecting a component to its first child, "*Next-sibling*", connecting a component to the following child of the same father, "*Root*", connecting a component to the root of the tree, and so on.

Structural links can be used by the reader to navigate among chunks of information belonging to the same entity. Navigationally, they are a little more complex than perspective links, but they are still quite simple, since they leave the reader inside the same information context, i.e., the same entity, with limited danger of getting lost. In the worst case the "Root" link can always take the reader to a safe point (i.e., the root of the entity).

2.2.8 *Application Links and Link Types.* *Application links* represent the most expressive portion of any non trivial hypertext; they represent domain dependent relationships among entities, or their components. These relationships are chosen by the author as both navigationally and semantically relevant.

Application links are organized into types. An *application link type*, or *link type* for short, is specified in HDM by a *name*, a set of *source* and *target entity types*, and a *symmetry attribute*, which can assume two values—*symmetric* or *asymmetric*. Source and target entity types define what can be linked to what. Once a link type has been defined as having source entity type A and target entity type B, *instances* of this link type are allowed to connect entities or components of type A to entities or components of type B only. The symmetry attribute defines a semantic property of the link type—whether or not each link of that type has an inverse link. This property can be exploited for automatic derivation of application links (see Subsection 2.3).

An example of application link type is "is-author-of", whose instances connect entities or components of type "Composer" to entities or components of type "Musical Opera". Another example is "is-justified-by", whose instances connect entities or components of type of "Contract", to entities or components of type "Law".

The semantic depth or arbitrariness of an application link type can vary, and is left to the author's responsibility. If establishing the composer of an opera corresponds to expressing a purely matter-of-fact truth, deciding that a given law is particularly relevant for a (portion of) contract, may express an arbitrary and debatable judgement.

In HDM, the choice of application link types and the placement of their instances is completely left to the author. By requiring the specification of source and target entity types in the link type definition, HDM only enforces syntactic consistency, and therefore it would not allow, for example, to establish an "is-author-of" link from a "Musical Opera" to a "Law".

From a navigational point of view, application links are typically the most troublesome, since when traversing application links the reader perceives that his information context is abruptly changed. Assume that the reader is examining a contract. While he is navigating among its different parts (following structural or perspective links), he is always within the same familiar information context. If now he follows an application link of type "is-justified-by", he will find himself looking at an article of a law and not at the clause of a contract. After having followed a number of application links, the reader might get disoriented by the different information contexts (i.e., entities of different types) he has traversed.

2.2.9 *Entity Types Revisited.* We can now reexamine the notion of entity type and characterize it in a more precise fashion. All the entities belonging to the same type have certain features in common: (1) the name of the their entity type; (2) the set of perspectives under which their content (i.e., the body of their units) is presented; (3) the types of their outgoing and incoming application links.

2.2.10 *HDM Schema.* An HDM specification of a hypertext application consists of a *schema definition* and a set of *instances definitions*. A schema definition specifies a set of entity types and link types. Instances are allowed to be inserted in the application only if they obey the constraints specified by the schema.

The notion of schema is relatively new in the hypertext field, but it is obviously derived from the current practice in the database field. Database schemas started (in the early sixties) as simple templates, describing file structures, that allowed generic operations such as "find" or "search" to be performed, independently from structural and implementation details of files. With time these file descriptors have become more complex, incorporating more semantic features.

In the hypertext field the evolution seems to start following the same course; developers of applications, who need to perform several times the same operations on complex node/link structures, have acknowledged that "templates"—descriptors of network patterns—are useful to save effort and to ensure consistency [27, 38]. Most existing template mechanisms in hypertext, however, are defined essentially on a syntactic basis. Taking advantage of the historical experience of the Data Base field, HDM attempts to go directly to an advanced notion of Schema that is able to represent some semantic features of hypertext applications, as well as to exploit syntactical regularities.

2.2.11 *Outlines.* According to the HDM terminology, a hypertext application can be, roughly speaking, divided in two portions: a *hyperbase* and a set of *access structures.*

The hyperbase represents the core of the application; it consists of all the elements defined in the previous sections: entities, components, units, and links of all the categories previously discussed—structural, perspective, and application links. The reader can explore the hyperbase by traversing the links defined there.

Before he can start this navigation, however, the reader must have entry points to get a view of what the hyperbase is about, and to locate the most convenient starting point. Access structures have the purpose of allowing the reader to properly select the entry points for further navigation.

In HDM, so far, we have mainly focused our attention on modeling the hyperbase, which represents the most relevant portion of any hypertext. For access structures, at the moment, we provide a unique primitive, the *outline*, explained below. In the Conclusions (Subsection 4.4) current work for the enhancement of this part of HDM is described.

Outlines are structurally quite similar to hyperbase entities, but they have a number of restrictions and peculiarities that make them substantially different from entities. An outline is an ordered tree of components. Each component (but for leaf components) is connected to its child components, to its siblings, and to its parent (but for the root component), via structural links. Differently from entities, however, these are the *only* outgoing or incoming links for nonleaf components; additionally, leaf components *must* have a number of outgoing (untyped) links to entities or components of the hyperbase. Outlines are not typed and are not specified in the schema—they can be freely added or modified, at will, in an application, in order to provide the appropriate entry points for the reader.

An outline for applications in the legal field, for example, provides an access to "European Laws", or "National Laws" in the root component. Selecting "National Laws" corresponds to traversing a link to a component where a further choice is presented among different categories of national laws—say, for example, "Laws concerning real estate sale", "Laws concerning donation", etc. There is an outgoing link for each of the above categories of laws; each one of these links points to a leaf component, which has a number of links to specific entities of type "Law" of the selected subcategory (say, for example, "Law 10/21/87", "Law 3/5/75").

2.3 Derived Links and Derived Link Types

One of the central advantages of HDM in the design and practical construction of hypertext applications is that defining a significant number of links can be left implicit—being induced from structural properties of the model—or can be defined intensionally and algorithmically derived. At the conceptual level, the author needs to provide a much smaller number of links than the number of links that will actually be present in the application; once the proper interpreter is provided, all implicit or derivable links can be generated automatically from the design level description, by using the schema definitions.

All perspective links can be left implicit and can be automatically derived from the definition of components and units. For example, if perspectives "score" and "sound" are defined for entity type "Opera", two perspective links will be generated *for each* component of an entity of this type: the first connecting the unit of the component under "score" perspective to the unit under "sound" perspective of the same component; the second being the inverse of the first.

Defining an entity is equivalent to defining a set of components plus the minimal set of "basic" structural links that are necessary to induce an ordered hierarchy relationship on the set of components: links from any (nonleaf) component to its first child, and links from any component to the next child of the same parent. A large number of other structural links can be defined intensionally, and can be computed, from the basic ones. For example: "*down*(N)" (N positive integer)—connecting a component directly to its Nth child; "*down*"—connecting a parent component to all its children; "*up*"—connecting a component to its parent; "*to-top*"—connecting a component to the root of the entity.

Derived structural link types can be defined in terms of operators such as "*inversion*", "*composition*", "*composition*(N)" (repeated composition, N times), "*transitive closure*", and "*closure*"—applied to basic structural links. All instances of derived structural link types can be automatically computed from the basic structural links, by a proper interpreter "executing" the corresponding derivation clauses.

A similar approach can be applied to application links. For example, from application link types that have been defined as "symmetric" (see Section 2.2.8), the corresponding inverse link types can be specified by using the inversion operator, and their instances can be automatically generated. As a more sophisticated case, we can specify derivation rules such as: "given an application link type, if an instance of it exists between two components of different entities, then an instance of the same link type is derived between the roots of the corresponding entities."

Assume for example that a component of an entity of type "legal document" —say "section 1.1" of a contract "X", has been connected by the author to a component of an entity of type "law"—say "Article 2" of law "Y"—by a link of type "justified-by" (see Figure 1). Then assume that the author wants to represent also the fact that, at a different level of detail, the whole contract X "is-justified" by the whole law Y. This fact can be expressed by a link between the root of contract "X" (which is intended to represent the whole entity "X") and the root of law "Y" (which is intended to represent the whole entity "Y"). This link can be derived by composing the inverse of link "to-top"—outgoing

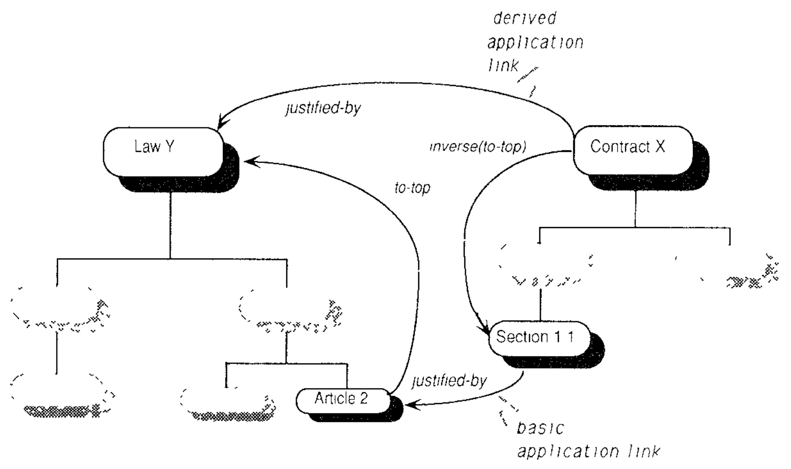


Fig. 1. Example of derived application link.

from component "section 1.1"—with the link "is-justified-by"—outgoing from the same component—and then composing the result with the link "to-top" outgoing from component "Article 2".

2.4 Browsing Semantics

The actual use of a hypertext is largely defined by its *browsing semantics* [39], which will determine what the information objects are for "human consumption" (in HDM terms, entity types, link types, entities, outlines, components, or units), what the perceived links between these objects are, and what the behavior when links are activated is.

Given the specification of a particular browsing semantics compatible with a given target hypertext systems, it becomes possible to translate HDM application specifications into running applications, once the mapping from HDM primitives into implementation structures of the target environment has been provided. By applying different browsing semantics to the same HDM static specifications, it is possible to deliver the same HDM application under different versions, each one characterized by different visual features and dynamic behaviors, or running in different hypertext systems.

The particular browsing semantics will be closely dependent on the particular hypertext development system being used for implementation. HDM, as discussed so far, does not prescribe a priori any specific browsing semantics for hypertexts specified with it, nor does it include any high level primitive for defining browsing semantics. However, we have defined a minimal browsing semantics, compatible with plain *nodes-and-links* structures found in most hypertext systems, and we have adopted it for our experiments on automatic translation of HDM specifications into running applications (see

Section 4.3). For the moment, this browsing semantics is considered as the *default browsing semantics* for HDM.[2]

2.4.1 *Default Browsing Semantics.* HDM default browsing semantics assumes that only units can be perceived by the readers as standard "nodes", i.e., "loci of navigation control", and that only one node is active at any time. As a consequence, readers can perceive links only among units, and so, in the end, actual, navigable connections must be established among units.

Since in HDM only perspective links connect units, while structural and application links are established among components or/and entities, the latter must be properly translated into unit-to-unit links. We will call "abstract" the links among components and/or entities, and "concrete" the unit-to-unit links.

2.4.1.1 *From Abstract to Concrete Links.* HDM default rules for translating abstract links into concrete links are based on the idea of having a *default representative* for each abstract object (component or entity). Defining default representatives of entities and components is done by introducing a *default perspective* for each entity type, and then assuming that the default representative for a component is its unit under the default perspective and the default representative for an entity is the default representative of its root component. This corresponds to saying that the root component of an entity, in its default perspective, "stands" for that entity.

Given this notion, entity-to-entity application links translate into concrete links between their default representatives. Each component-to-component application link is translated into a *set* of concrete links connecting each unit of the source component to the default representative of the target component.

To illustrate this rule, consider the following situation occurring in a hypermedia music listening guide; "La Traviata-Ouverture" component (of entity "La Traviata" of type "Musical Work"), is linked to "Verdi-1" component (root of entity "Verdi" of type "Author"), through a "Composer of" application link. Consider furthermore that "Author" entity type has perspectives "Picture" (showing the person) and "Text" (with a textual description), while "Musical Work" has "Text" (with the music score) and "Music" perspectives. "Picture" is the default perspective for entity type "Author".

In the above case, the actual concrete links corresponding to the abstract link connecting the two components, will connect both "La Traviata-Ouverture: Text" unit and "La Traviata-Ouverture: Music" unit to "Verdi-1: Picture" unit. This is an example where one link at the abstract level corresponds to two concrete navigable links. The situation is illustrated in Figure 2.

[2] The default browsing semantics has been implemented in a prototype system, which uses a relational database description of HDM specifications, and generates a tabular description that can be imported into Hypercard and manipulated using Hypertalk. This prototype is described in detail in [36].

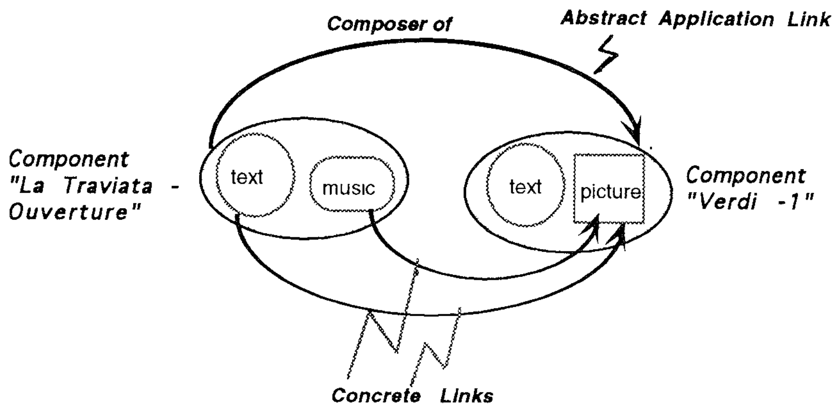


Fig. 2. Example of translation from abstract to concrete links.

The default rule for component-to-component *structural* links translation is the following: if a component C1 has a structural link to a component C2, then, for each perspective P, each unit of C1 having this perspective is linked to the unit of C2 having the *same* perspective. Therefore, in an entity of type T having N perspectives, N concrete links correspond to each structural link defined at the conceptual level.

2.4.1.2 *Perception of Links: Anchors and Anchor Types.* In hypertext, connections among pieces of information are usually perceived through "anchors" (or "buttons"). Anchors are link place-holders that show the existence of connection in nodes, and can be selected ("clicked") by the reader in order to get into the link target(s). An *anchor type* specifies the properties of anchors that represent links of the same type, or of a group of link types.

Consider for example, that a "Procedure" entity type has an outgoing link type "Formal-Legal-Justification" to a "Law" and an outgoing link type "Informal-Justification" to an "Informal-Regulation". The author might want to provide to certain classes of users a single reading link, simply labelled "Justification", since the distinction between formal and informal justifications might not be important for readers of that class. This can be achieved by specifying the anchor type "Justification" to be the *union of application link types* "Formal-Legal-Justification" and "Informal-Justification".

Another design requirement of the author might be to hide links of a given type for a specific application (since they might be relevant at design and specification level but not at reader level). This can be achieved by simply not assigning any anchor type to that link type. Anchor types, therefore, provide a mechanism for the author to present groups of link types together, also renaming or hiding them, as desired.

When an anchor actually refers to several possible destination nodes, the HDM default browsing semantics has the notion of *chooser*, i.e., a structure that is associated with an anchor and allows, via a "menus", to select one of the multiple targets of the anchor.[3]

3. EXAMPLE OF HYPERTEXT MODELING WITH HDM

This section will describe a hypertext application designed with HDM—a subset of a larger prototype application for banking environment named Expert Dictionary [15] which has been developed by ARG SpA and Politecnico di Milano within the European Esprit project SUPERDOC.

The goal of the Expert Dictionary is to provide an organized way to access to a large set of information of vastly different but interrelated nature handled by credit organizations. A credit organization manipulates *Documents* according to *Procedures*. Documents and Procedures are defined according to *Laws*, *Regulations*, and *Informal Norms*. Laws are issued by the state to discipline and control credit granting and taking activity. Most of the times laws are too broad, and must be made more specific by Regulations issued by some authority, on the basis of the text of the law. Finally, these regulations are interpreted within an organization with the addition of *Informal Norms*, which are of course valid only for that organization.

Entity types and application link types that model the above state of affairs are sketched in Figure 3. Each of these entity types has a set of *perspectives* associated with it, which are omitted for lack of space. For example: *Laws* and *Regulations* have *Structure** and *Official Text* perspectives; *Documents* have *Structure**, *Official Text*, and *Description* perspectives; *Procedures* have *Flow Diagram Structure** ("*Structure*" for short) and *Description* perspectives. The perspective marked with a "*" is the default. All application link types are "symmetric" (see Section 2.2.8), i.e., they represent a relation and its inverse. Thus we have drawn link types only once, but we assume that for each application link type, there is a derivable link type that corresponds to its inverse.

A fragment of an instance of the schema above is depicted in Figure 4. It shows an entity of type *Procedure* named *Mortgage Loan Procedure*, which is made up of a root component introducing the whole entity and several other components denoting subprocedures or subprocedure steps. Another entity is *Circular HyperBank 21 / 10 / 89* of type *Regulation*, which is made up of a root and four components: *Units Involved*, *Subject*, *Operational Norms in Request Verification*, and *Data Entry Rules*. These last two components represent information that respectively affect the way the procedure (sub) steps *Request Verification* and *Request Data Entry* are performed; therefore, application links of type *has-effects-on* are included. Entity *Circular HyperBank 21 / 10 / 89*, in turn, is *motivated-by* entities *Law 19 / 8 / 89* and

[3] Choosers can be generated automatically from the specification of the concrete links in the hypertext application.

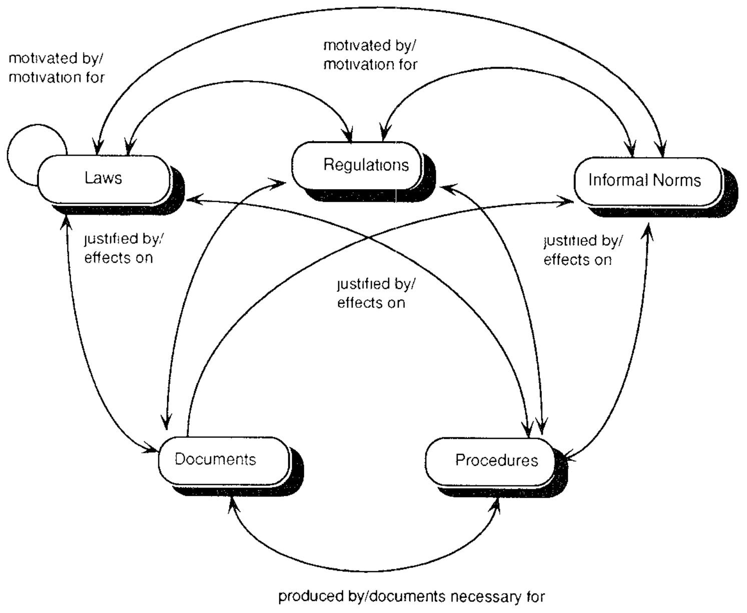


Fig 3. Schema of entity types and application link types of the Expert Dictionary.

Administrative Council Deliberation 20 / 9 / 89. The small black rectangle on the right-hand side of the picture is an outline named *Access to Mortgage Loan Client Information*, which allows us to directly access entities describing documents needed for the compilation of a Mortgage Loan Request (in the picture, *Mortgage Loan Request Form and Notary Statement*).

The following figures show a few examples of screens (corresponding to HDM units of the appropriate components) from the Hypercard implementation of the Expert Dictionary, which was generated adopting the default browsing semantics described in Section 2.4.1. The same application, with the same browsing semantics, has been implemented also in Hyperpad.

Figure 5 shows the "Structure" perspective of a component of entity "Circular HyperBank 21/10/89" of type Regulation. Notice that all the links of type "Justified by" have been grouped under the anchor (button) "Motivations". Figure 6 shows the same component under the perspective "Official Text".

Anchors corresponding to application links ("Effects", "Motivations") and perspective links ("Description" and "Official Text" in Figure 5, and "Structure" and "Description" in Figure 6) are at the bottom of the screen. If the user is interested in seeing, for example, the effects of the regulation shown in Figures 5 and 6, he can click on the anchor "Effects". Since this in reality groups several outgoing links, a chooser is activated; from it, the user can select which of the possible destinations he wishes to go to.

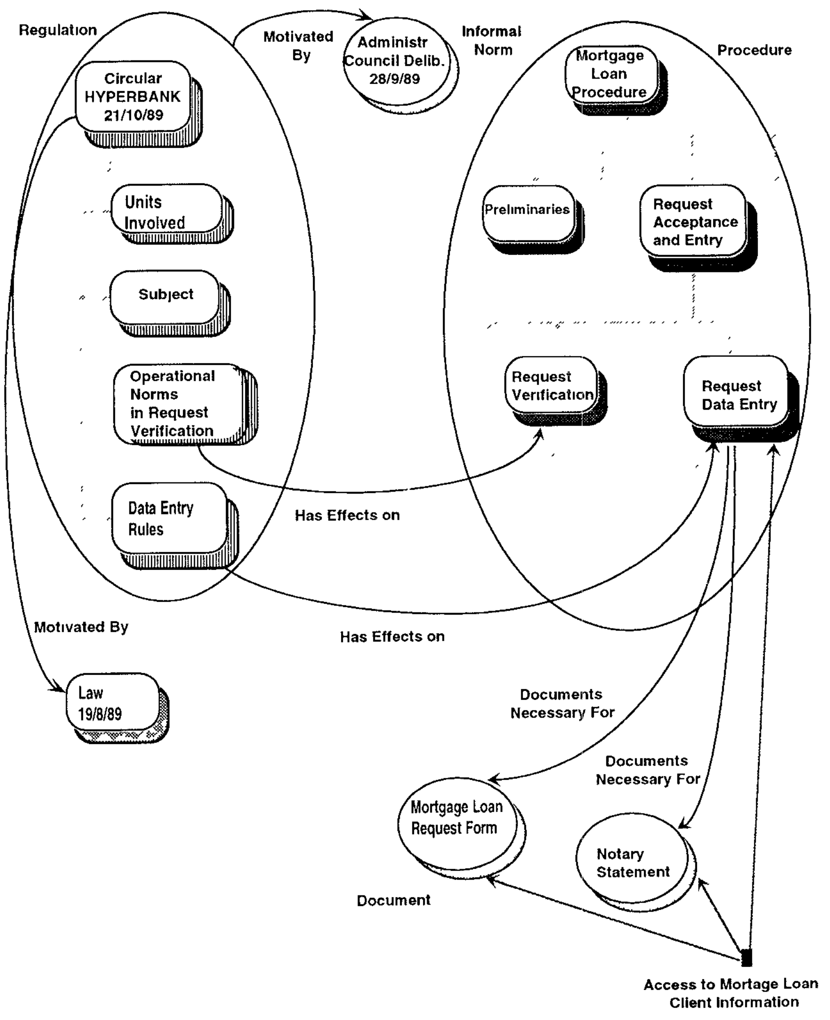


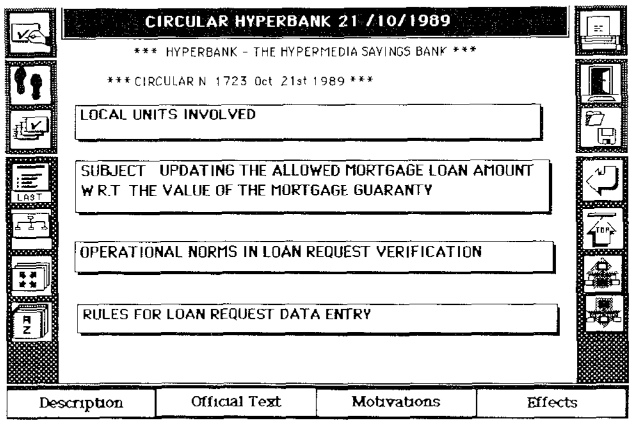
Fig. 4. Instance of Expert Dictionary schema in Figure 3.

4. CONCLUSIONS

4.1 Comparison with Other Works

HDM shares some apparent similarities with the Entity Relationship (E-R) model [10]. However, there is no E-R notion equivalent to HDM perspectives. Additionally, HDM Entities are much more structured than E-R entities, which are essentially flat and do not have structural links. Most importantly, whereas in the E-R model relations are included for representational reasons, HDM links are included also with the goal of providing navigation paths.

HDM differs from g-IBIS [11] in the fact that it does not freeze, a priori, the application domain, and therefore its representation primitives are more


CIRCULAR HYPERBANK 21 /10/1989

*** HYPERBANK - THE HYPERMEDIA SAVINGS BANK ***

*** CIRCULAR N 1723 Oct 21st 1989 ***

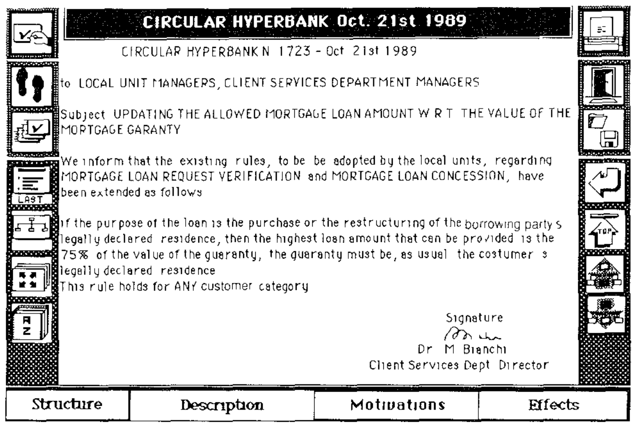
LOCAL UNITS INVOLVED

SUBJECT UPDATING THE ALLOWED MORTGAGE LOAN AMOUNT W R.T THE VALUE OF THE MORTGAGE GUARANTY

OPERATIONAL NORMS IN LOAN REQUEST VERIFICATION

RULES FOR LOAN REQUEST DATA ENTRY

Fig. 5. "Structure" perspective of a Regulation.


CIRCULAR HYPERBANK Oct. 21st 1989

CIRCULAR HYPERBANK N 1723 - Oct 21st 1989

to LOCAL UNIT MANAGERS, CLIENT SERVICES DEPARTMENT MANAGERS

Subject UPDATING THE ALLOWED MORTGAGE LOAN AMOUNT W R T THE VALUE OF THE MORTGAGE GARANTY

We inform that the existing rules, to be be adopted by the local units, regarding MORTGAGE LOAN REQUEST VERIFICATION and MORTGAGE LOAN CONCESSION, have been extended as follows

if the purpose of the loan is the purchase or the restructuring of the borrowing party s legally declared residence, then the highest loan amount that can be provided is the 75% of the value of the guaranty, the guaranty must be, as usual the costumer s legally declared residence
This rule holds for ANY customer category

Signature
[illegible]
Dr M Bianchi
Client Services Dept Director

Fig. 6. "Official text" perspective of the Regulation in Figure 5.

"general", oriented towards allowing the design of hypertext applications in most domains. HDM philosophy shares with IDE [27] the aim of supporting representation tasks in hypertext and of encouraging the creation of regular and consistent network structures. IDE has surely anticipated a number of concepts exploited in HDM. However, IDE is basically a development tool, while HDM is also a conceptual design device which can be useful even without an implementation of its primitives. Moreover, HDM provides richer modeling primitives—the notion of perspective, the distinction between different categories of links, the separation between hyperbase and access structures, and between authoring structures and reading structures (i.e., browsing semantics). Object Lens [28] classes resemble somewhat HDM entity types. Object Lens, however, does not impose any constraint on possible connections between object class instances and no real discipline on actual instances creation.

With respect to other hypertext models discussed in Section 1.2, HDM differs in the fact that it is aimed at modeling applications rather than systems. HDM shares with the Trellis model [39] and Tompa's model [41] the idea of abstracting from the contents and structure of the "nodes", which are allowed to contain arbitrarily complex information structures. Differently from HDM, however, these models are more concerned with behavioral aspects of hypertext, or with operational features, rather than on representational issues.

The approach introduced in [40] shares with HDM the idea of distinguishing between authoring language, for defining hypertext structures in a system independent manner, and browsing semantics language. As far as browsing semantics is concerned, the work presented in [40] goes a step ahead with respect to HDM, by providing a graph-grammar based language for specifying patterns of visual and dynamic behaviors (style templates), and general techniques for translating generic authoring notations into systematically structured hypertext. In this sense, this work is somehow complementary to HDM, whose major focus for the moment is on static representational aspects.

4.2 Evaluation, Experiences, and Feedback

The HDM model can be used in different manners: as a modeling device or as an implementation device. As a modeling device, it allows to lay down clear and concise specifications of applications to be developed, or to describe applications already developed, in a system independent manner. Using HDM as an implementation device, means to directly support the development effort by HDM related tools, as described in Section 4.3.

A number of experiences have shown that significant gains in the quality and speed of the whole process of application development can be achieved by using HDM as modeling device. In particular, we have observed that HDM is especially suitable for applications where regularity, organization, modularization, and consistency are important factors. Typical examples of such applications are, for example, technical documentation, training and education [12, 27], auditing systems [13], and in general, large applications for corporate environments. It is therefore true that "creative" applications [5], where the author tends to follow his feelings in a somehow unpredictable way (e.g., inventing new nodes and new navigation patterns on the fly) rather than planning his exercise in advance, are not conveniently modeled by HDM. Our conjecture (not yet fully tested) is that most of the intensively used, existing or future, hypertext applications are unlikely to be of the "creative" kind, but rather they are planned, cohesive, and carefully organized.

As a modeling device, HDM has been successfully used several times. Beside the authors, several different research groups in four different countries are using HDM in a significant number of different application fields, which range from administration and technical manuals, to university class notes, literature, philosophy encyclopaedias, art collections, and exhibitions [9, 23, 24, 25, 34]. All of them have reported that the model can be relatively easily grasped both by the users and by designers, and that a more productive cooperation can be achieved among the persons involved in the process of application development—analysts and domain experts, analysts and implementors, different implementors. Domain experts are able to discuss the design of the application in HDM terms, and to autonomously modify and improve it. This implies a tremendous gain in the efficiency and quality of knowledge and requirement analysis processes. Members of the implementation team are able to discuss the application with the analysts and to get a deep and precise understanding of the requirements of their work without the need of prototyping it. This makes it possible to split the implementation tasks among different persons, still obtaining consistent applications. Since implementors are guided by clear and precise specifications, arbitrariness is virtually eliminated.

An interesting side effect of using HDM in team work has been the fact that people who use HDM as a modeling device, tend to invest a quite extensive time in discussing "organization style" issues (e.g., the best way of organizing a set of information, the choice between establishing perspective links versus application links, etc.). These discussions are extremely effective: the overall quality of applications improves, in general, and, more interesting, groups tend to develop a common, consistent design style across different application domains.

Additionally, HDM and related tools have been shown to be quite useful in making the development of the same application in different versions easier, by "switching" from one environment to another. Target systems, so far, have been Hypercard, Supercard, Hyperpad, Toolbook, Director (the latter for animations embedded in Hypercard-based hypertext).

4.3 Tools and Implementation

We have developed so far a number of elementary HDM-tools to speed up the development of our applications. For the time being, these tools perform the following operations:

—automatic suggestion of application links (out of link type descriptions in the schema);

—automatic derivation of some application links (the inverses of author-defined application links, and the root-to-root derived links based on the rule exemplified in Figure 1—Subsection 4.3);

—automatic derivation of structural and perspective links;

—maintenance of link tables (which describe the extensions corresponding to link types);

—automatic derivation of a class of outlines—those allowing to enter the hyperbase by first selecting the entity types of interest, and then choosing the desired entity within the list of all entities of the selected type;

—automatic creation of entity templates (out of entity type descriptions in the schema);

—automatic creation of visual templates for nodes, including placement of anchors and association of anchors to link instances.

For implementation purposes, HDM has not been yet tested on a sufficiently large scale; the initial results, however, are quite encouraging. Even in their rudimentary stage of development, the above tools have greatly increased the effectiveness of our development, by a ratio of four to one, and we have achieved enormous savings in the development effort. For example, in an educational application based on six entity types and eighteen link types, six hundred units ("nodes"), and four thousand link instances, all the visual templates for nodes of the different types have been automatically generated, and 80% of the link instances have been derived. It is easy to understand why this happens; just think of the effectiveness of automatic generation of visual templates, including anchors and their associations to links, and, above all, automatic generation of derived links.

4.4 Further Developments

All the above tools, although effective in their own respect, do not constitute a well defined consistent development environment. The research project HYTEA [1], within the CEC ESPRIT program, is currently aiming at the development of a full size HDM-based development environment for hypertext applications. The project involves four different companies and two research institutions, and is scheduled to be completed by March 1993. Within the HYTEA project, the modeling primitives of HDM have been "transferred" into the concrete syntax of a language, named HDL—HDM definition language [22], which has been omitted here for lack of space. HDL is more a reference language than a language which will be directly used by application designers. In the final HYTEA environment, authoring in HDM will be based on visual programming, rather than on traditional programming. Some work on defining visual objects corresponding to HDM design primitives is already in progress. A reference language is still useful, however, in order to define precisely the semantics of the interactive operations offered by the HDM authoring environment.

HDM is, obviously, an evolving model: as we gain applicative experience, we will modify the model according to the new emerging needs. In particular, access structures currently provided by HDM are restricted to the relatively unsophisticated notion of outline, which has been proved to be too limited. Looking at complex corporate environments, we found out that a typical situation is to have several different categories of readers using the same hyperbase. Each category of users naturally perceives the organization of the material according to its needs. From this observation, we came to the conclusion that access structures should evolve from just providing entry points to the hyperbase to becoming something like "hypertext views", i.e., ways of presenting "fictitious" hypertext environments, specifically tailored for a class of readers. "Fictitious" means that these environments are not implemented on their own, but are rather "simulated" by a suitable mapping on the underlying hyperbase. For this purpose, the notions we are working on are "derived entities" (i.e., entities assembled out of pieces of other, preexisting, entities), "user-derived" links (i.e., links derived only within a given reading environment), "parametric guided tours" (i.e., intensionally defined guided tours), and others of a similar kind. Another important aspect we are currently working on is multimediality. In the current version of HDM, images, animation, or video clips can be introduced as part of the authoring-in-the-small process. We have a prototype application, for example, where entities of type "history" are made of "history notes" (as their components), and each history note has two perspectives—text and animation. The reader can choose, according to his interest, either to read the textual description of an historical event or to watch the animated version of the same event. This approach works fine for simple applications, where multimedia has the limited role of presenting small chunks of information with a different perspective. In other applications, however, multimedia may have a larger role. A video showing a maintenance procedure, for example, touches different subjects; it would be artificial (and ineffective) to model it as a unit associated to a single component. We are currently working around the idea that "active" information objects such as animation or videoclips could play the role of "automated" guided tours [32, 42, 44], i.e., devices which automatically take the reader along different active information units of an underlying hyperbase, and are "synchronized" with interlinked chunks of static information. We are currently developing a large-scale application [34] around this idea. On the basis of the fourthcoming results we will introduce the proper extensions to HDM.

ACKNOWLEDGMENTS

The ideas that originated the HDM work were first developed in the context of an ESPRIT project, SUPERDOC, led by ARG—Applied Research Group SpA. We must acknowledge the technical contribution of the ARG team in general, and Cristina Borelli, SUPERDOC Project Leader, in particular. Many of the ideas described here benefited from discussions with Mark Bernstein, Norman Meyrowitz, John Mylopolous, and the members of the HYTEA Project Team, Andrea Caloini and Luca Mainetti in particular. We are grateful to the TOIS Associate Editor, Polle Zellweger, and to the TOIS anonymous reviewers for their extraordinarily helpful comments on previous versions of this paper.

REFERENCES

1. ARG-APPLIED RESEARCH GROUP SPA. HYTEA Technical Annex. Tech. Rep., ESPRIT project 5252 (HYTEA), 1990.
2 AKSCYN, R., MCCRACKEN, D., AND YODER, E. KMS: A distributed hypertext system for managing knowledge in organizations. *Commun. ACM 31*, 7 (1988), 820–835.
3. ATKINSON, W. HyperCard. In *Software for Macintosh Computers* Apple Computer Co, Cupertino, 1987.
4. BERNSTEIN, M., AND SWEENEY, E. *The Election of 1912, Hypertext for Macintosh Computers.* Eastgate Systems Inc, Watertown Mass., 1989.
5. BOLTER, J. D., AND JOYCE, M Hypertext and creative writing. In *Proceedings ACM Hypertext '87* (Chapel Hill, N.C., Nov. 13–15, 1987), pp 41–50.

6. Brodie, M., Mylopolous, J., and Schmidt, J., Eds. *On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages.* Springer Verlag, 1984.
7. Brown, P. J. Turnıng ideas into products: The Guide system. In *Proceedings of the ACM Hypertext '87* (Chapel Hill, N.C., 1987), pp. 33–40.
8. Brown, P. J. Assessing the quality of hypertext documents. In *Hypertexts: Concepts, Systems and Applications* (*Proceedings of ECHT* '90), A. Rizk, N. Streitz, and J. André, Eds., Cambridge University Press, Cambridge, 1990, pp. 1–12.
9. Caloini, A., Garzotto F., and Paolini, P. Hypermedia course notes: The experience of Politecnico di Milano. In *Proceedings of the Italian Conference on Hypertext in Education and Research* (Torino, 1991), pp. 35–42.
10. Chen, P. The entity-relationship approach: Toward a unified view of data. *ACM Trans. Database Syst. 1*, 1 (1976), 9–36.
11. Conklin, J., and Begeman, M. L. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Trans. Inf. Syst. 6*, 4 (1988), 303–331.
12. Crane, G. From the old to the new: Integrating hypertext into traditional scholarship. In *Proceedings of the ACM Hypertext '87* (Chapel Hill, N.C., Nov. 13–15, 1987), pp. 51–59.
13. De Young, L. Hypertext challenges in the auditing domain. In *Proceedings of ACM Hypertext '89* (Pittsburgh, Pa., Nov. 5–8, 1989), pp. 169–180.
14. Garg, P. K. Abstraction mechanisms in hypertext. *Commun. ACM 31*, 7 (1988), 862–870.
15. Garzotto F., and Paolini P. Expert dictionaries: Knowledge based tools for explanation and maintenance of complex application environments. In *Proceedings of the 2nd ACM Conference on Industrıal and Engıneering Applications of Al and Expert Systems* (Tullahoma, Tenn., June 6–9, 1989), pp. 157–169.
16. Garzotto, F., Paolini, P., and Schwabe, D. Authoring-in-the-large: Software engineering techniques for hypertext application design. In *Proceedings of the 6th IEEE Internatıonal Workshop on Software Specification and Design* (Como, Oct. 25–26, 1991), pp. 87–98.
17. Garzotto, F., Paolini P., Schwabe, D., and Berstein, M. Tools for designer. In *Hypertext/Hypermedia Handbook*, E. Berk, and J. Devlin, Eds., McGraw Hill, 1991, 179–207.
18. Garzotto, F., Paolini, P., and Schwabe, D. HDM—A model for the design of hypertext applications. In *Proceedings ACM Hypertext '91* (San Antonio, Tex., Dec. 15–18, 1991), pp. 313–328.
19. Halasz, F. Reflections on NoteCards: Seven issues for the next generation of hypertext systems. *Commun. ACM 31*, 7 (1988), 836–851.
20. Halasz, F., and Schwartz, M. The Dexter reference model. In *Proceedings of the 1st Hypertext NIST Standardızatıon Workshop* (Gaithersburg, Md., Jan. 16–18, 1990), pp. 95–133.
21. Hull, P., and King, R. Semantic database modelling: Survey, applications, and research issues. *ACM Comput. Surv. 19*, 3 (1987), 201–260.
22. HYTEA PROJECT. HDL—HDM definition language. Tech. Rep. D2, ESPRIT Project 5252 (HYTEA), 1991.
23. HYTEA PROJECT. Hypermedia technical documentation for the forms processing system Siemens-SIFORM. Tech. Rep. D4.1, ESPRIT Project 5252 (HYTEA), 1992.
24. HYTEA PROJECT. Hypermedia technical documentation for IVECO-FIAT workshops. Tech. Rep. D4.2-ESPRIT Project 5252 (HYTEA), 1992.
25. HYTEA PROJECT. Hypermedia for cultural applications: Greek Modern Painting between the two world wars. Tech. Rep. D5, ESPRIT Project 5252 (HYTEA), 1992.
26. HYTEA PROJECT. Authoring/delivery HYTEA tools: Specification and design. Tech. Rep. 2, ESPRIT Project P5252 (HYTEA), 1992.
27. Jordan, D., and Russel, D. Facilitating the development of representations in hypertext with IDE. In *Proceedings ACM Hypertext '89* (Pittsburgh, Pa., Nov. 5–8, 1989), pp. 93–104.
28. Lai K. Y., Malone, T. W., and Yu K. C. Object lens: A spreadsheet for cooperative work. *ACM Trans. Inf. Syst. 6*, 4 (1988), 332–353.
29. Marshall, C. C., and Irish, P. M. Guided tours and on-line presentations: How authors make existing hypertext intelligible for readers. In *Proceedıngs ACM Hypertext '89* (Pittsburgh, Pa., Nov. 5–8, 1989), pp. 15–26.

30. Mylonas, E., and Heath, S. Hypertext from the data point of view: Paths and lınks in the Perseus Project. In *Hypertexts. Concepts, Systems and Applıcations (Proceedıngs of ECHT '90)*, A. Rizk, N. Streitz, and J. André, Eds., Cambridge University Press, Cambridge, 1990, pp. 324–336.
31. Nielsen, J The art of navigating through hypertext. *Commun. ACM 33*, 3 (1990), 296–310.
32. Paolini P., Caloini, A., and Garzotto, F. Active medıa and guıded tours. Tech. Rep. 78-91, Dep. of Electronics, Politecnico di Milano, 1991.
33. Parunak, H. V. D.. Hypertext topologies and user navigation. In *Proceedıngs ACM Hypertext '87* (Chapel Hill, N.C., Nov. 13–15, 1987), pp. 43–50.
34. RAI-RADIO TELEVISIONE ITALIANA. The multimedia encyclopedıa of philosophy. Tech. Rep , Dept of Scholarship and Education, RAI, 1991.
35. Richard, G., and Rizk, A. Quelques idées pour une modelizatıon des systèmes hypertextes. *Technıques et Scıences Informatıques. 9*, 6 (1990), 505–514.
36. Schwabe, D., Caloini, A., Garzotto, F., and Paolini, P. Hypertext development usıng a model-based approach. *Softw. Pract. Exper. 22*, 11 (1992), 937–962.
37. Shneiderman, B. (Ed.) *Hypertext on Hypertext*. Database and Electronic Product Series, ACM Press, 1988
38. Smith, C. K., Garret, L. N., and Lauhard, J. A. Hypermedıa templates: An author's tools. In *Proceedıngs ACM Hypertext '91* (San Antonio, Tex., 1991), pp. 147–160.
39 Stotts, P D., and Furuta, R. Petrı-net-based hypertext: Document structure wıth browsing semantics. *ACM Trans Inf. Syst. 7*, 1 (1989), 3–29.
40. Stotts, P. D., and Furuta, R. Hierarchy, composition, scrıptıng languages, and translators for a structured hypertext. In *Hypertexts: Concepts, Systems and Applıcatıons (Proceedings of ECHT '90)*, A. Rizk, N. Streitz, and J André, Eds., Cambridge Universıty Press, Cambridge, 1990, pp. 180–193.
41. Tompa, F. A data model for flexible hypertext database systems. *ACM Trans. Inf. Syst. 7*, 1 (1990), 85–100.
42. Trigg, R. H Guided tours and tabletops: Tools for communicating in hypertext envıronments. *ACM Trans. Inf. Syst. 6*, 4 (1988), 398–414.
43. Utting, K., and Yankelovich, N Context and orientation in hypertext networks *ACM Trans. Inf. Syst. 7*, 1 (1989) 58–84
44. Zellweger, P. T. Scripted documents: A hypermedia path mechanism. In *Proceedıngs ACM Hypertext '89* (Pittsburgh, Pa., Nov. 5–8, 1989), pp. 1–14.
45. Walker, J. H. Supporting document development with Concordıa. *IEEE Computer 21*, 1 (1988), 48–59.
46. Wiederhold, G. *Database Design* McGraw Hill, 1983.
47. Winter, R. T. *Ludwıg Van Beethoven Symphony Number 9*. CD Companions Serıes, The Voyager Company, 1989
48. Yankelovich, N. Haan, B J Meyrowitz, N. K., and Drucker, S M. Intermedıa: The concept and construction of a seamless information envıronment. *IEEE Computer 21*, 1 (1988), 91–96

Receıved January 1991; revised Aprıl 1992; accepted July 1992