

PUC

Series: Monografias em Ciência da Computação,
No. 22/90

HDM - A MODEL-BASED APPROACH TO HYPERTEXT
APPLICATION DESIGN

Franca Garzotto
Paolo Paolini
Daniel Schwabe

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP-22453
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA

Series: Monografias em Ciência da Computação, No. 22/90

Editor: Paulo A. S. Veloso

December, 1991

HDM - A MODEL BASED APPROACH TO HYPERTEXT APPLICATION DESIGN *

Franca Garzotto **

Paolo Paolini **

Daniel Schwabe

* This work has been partially sponsored by the Brazilian Government Office of Science and Technology.

** Politecnico di Milano, Milano, Italy, where this work has also been published, under series no. TR-90-75.

In charge of publications:

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio - Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ
Brasil

Tel.: (021) 529-9386

Telex: 31078

Fax: (021) 511-5645

E-mail: rosane@inf.puc-rio.br

O desenvolvimento de hipertextos deveria se beneficiar de um processo de desenvolvimento sistemático e estruturado, especialmente nos casos de aplicações grandes e complexas. Um enfoque estruturado para o desenvolvimento de hipertextos sugere a noção de *autoria-em-ponto-grande*, que permite a descrição de classes gerais de elementos de informação e de estruturas de navegação em um nível alto de abstração, sem muita preocupação com o conteúdo específico dos nodos, e de uma forma independente do sistema no qual será implementado. Este enfoque é baseado na crença de que decisões de projeto estrutural, que sejam sistemáticas e racionais, podem e devem ser tomadas antes de que o hipertexto propriamente dito seja escrito, permitindo que redes hipertexturais coerentes e expressivas possam se parte integrante do projeto original, em vez de serem adicionadas a-posteriori. A HDM ("Hypertext Design Model") é uma primeira tentativa no sentido de definir um modelo de propósito geral para autoria-em-ponto-grande, invando pelo fato de cobrir tanto aspectos estáticos como dinâmicoa da modelagem de hipertextos. Sob este ponto de vista, especificações HDM podem ser entendidas seja como especificações de requisitos de alto nível de aplicações, ou como base para uma geração (semi) automática de aplicações. Uma definição formal, e exemplos de uso de HDM também são apresentados.

Palavras chave: Hipertextos, Sistemas multimídia, Modelos de Autoria para Hipertexto.

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*, which allows the description of overall classes of information elements and navigational structures of complex applications at a high level of abstraction, without much concern with the contents of particular nodes, and in a system-independent manner. This approach is based on the belief that systematic and rational structural design decisions about a hypertext application can and should be made before the actual hypertext is ever written, so that coherent and expressive hypertext webs can be designed-in instead of added-on. HDM (Hypertext Design Model) is a first step towards defining a general purpose model for authoring in the large, being innovative in discussing both static and dynamic hypertext modelling under this perspective. HDM specifications can be either used only as a high level application requirement specifications or as a basis for (semi) automatic generation of actual applications. A formal definition and 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; I.7.; [Text Processing]: Miscellaneous - *hypertext*

General Terms: Design, Models, Languages

Additional Keywords and Phrases: Hypertext Design Models, Hypertext Applications, Hypertext Structures.

HDM - A MODEL-BASED APPROACH TO HYPERTEXT APPLICATION DESIGN

Franca Garzotto, Paolo Paolini, Daniel Schwabe

Politecnico di Milano, Milano, Italy

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*, which allows the description of overall classes of information elements and navigational structures of complex applications at a high level of abstraction, without much concern with the contents of particular nodes, and in a system-independent manner. This approach is based on the belief that systematic and rational structural design decisions about a hypertext application can and should be made before the actual hypertext is ever written, so that coherent and expressive hypertext webs can be designed-in instead of added-on. HDM (Hypertext Design Model) is a first step towards defining a general purpose model for authoring in the large, being innovative in discussing both static and dynamic hypertext modelling under this perspective. HDM specifications can be either used only as a high level application requirement specifications or as a basis for (semi) automatic generation of actual applications. A formal definition and 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; I.7.; [Text Processing]: Miscellaneous - *hypertext*

General Terms: Design, Models, Languages

Additional Keywords and Phrases: Hypertext Design Models, Hypertext Applications, Hypertext Structures.

1 INTRODUCTION

1.1 Background

The degree of success of a hypertext application¹ 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

Authors' addresses: Department of Electronics - Politecnico di Milano Piazza Leonardo da Vinci 32 - 20133 Milano Italy. Phone: +39-2-23993634; Fax: +39-2-23993587; E-mail: relett34@imipoli.bitnet

Paolo Paolini is also with A.R.G. - Applied Research Group Via Pio La Torre 14 - Vimodrone (Milano) Italy. Phone: +39-2-2650072; Fax: +39-2-2650693 ; E-mail: argborll@icil64.cilea.it.

Daniel Schwabe developed this work while on sabbatical leave (until January 1991) from Departamento de Informática, Pontifícia 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; E-mail: schwabe@inf.puc-rio.br, and was partially supported by CNPq-Brasil.

The work reported here was sponsored in part by the EEC, in the scope of the HYTEA Project (P 5252) of the ESPRIT II Programme, whose main contractor is ARG SpA, Milano, Italy, and includes among other participants the Dipartimento di Elettronica, Politecnico di Milano.

¹In this report we use the term *hypertext* to denote online documents made up of a web of interlinked pages. We will use the term *hypertext system* for software tools used to create a hypertext. Notecards, KMS,

render it clear and accessible to a wide audience. To control the potential explosion on the number of links, hypertext 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.

The importance of a clean and rational organization of an hypertext application becomes even more evident when designing large hypertext applications, that are significantly larger than a conventional book. Examples of such application areas abound - legislators trying to cope with the massive amount of information related to the bills they must vote on; businesses and governments operating in multi-national, multi-lingual and cross-cultural environments; and authors of technical documentation for sophisticated equipment, such as modern day aircrafts.

Since one view of hypertext is as an information application, it is legitimate to ask why the process of developing a hypertext, and therefore the hypertext design methodology, should be different from the development process of other types of applications. There are two main reasons why design for hypertext needs specific considerations, rather than relying on standard design practices and tools.

The first reason is a technical one. Particular features of hypertexts, such as the complexity of the representation structures, the emphasis on visual aspects of the applications, the emphasis on interactivity and navigation, require specific concepts, methodologies and tools. "Specific", however, does not mean totally different.

The second (and stronger) reason is cultural. The pioneers of the hypertext field have traditionally approached the problem of design on an "autonomous" ground, developing specific notions and methods, with little correlation to practices and methods of other application areas.

From our point of view, in a rational hypertext design approach, a 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 the nodes' contents. This approach is based on the belief that systematic and rational structural decisions about the hypertext can and should be made *before* the actual hypertext is ever written, so that coherent and expressive hypertext webs can be designed-in instead of added-on.

This is very similar to what happens when developing a strongly modularized software system: designing the topology and the interconnections among modules is different than writing the code for the content of the modules themselves. By analogy, 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.

Authoring-in-the-small is very much related to the specific application area, while authoring in the large has common characteristics across many applications in a given application domain. Authoring in the large becomes very critical as soon as the size of the hypertext exceeds a manageable limit (where to put this limit is clearly matter of debate).

Authoring-in-the-small is strongly dependent on the medium (putting the text in a node is much different than putting an animation, or a sound, or a picture). Authoring-in-the-large

Intermedia, Hypergate are examples of hypertext systems. Note that a single hypertext might be published in several editions, each using a different hypertext system.

can be at some extent independent from the medium, i.e. establishing a connection between two nodes is somewhat independent from the representation of what is inside a node².

In a truly non-linear hypertext, most of the “complex” semantics should lay in the connections among the nodes, rather than in the nodes themselves. As a consequence, many of the “deeper” design decisions are made at the authoring-in-the-large level.

A model for authoring-in-the-large should provide the primitives which allow the author to describe the hypertext application in a more concise and clear manner than by simply building a skeleton of that application in a given hypertext system; furthermore, this description should be as much system-independent as possible. This can be achieved by defining schemas that describe overall classes of information elements; capture regular structures and connection patterns between them; and specify visualization, synchronization and traversal properties of such objects. Once a schema has been specified, the model also provides primitives to describe actual instances of information elements, as well as navigational structures connecting them. This process essentially provides a way to describe a complete hypertext application in a system independent manner.

Beyond helping the author conceptualize a given application without too much regard to “implementation details”, and also as a communication language between designers, implementors and users, model based descriptions can potentially be used in a more “operational” way. By providing a suitable mapping from an application instance specification into the implementation structures of a given hypertext system, the model can be used to generate running implementations of the application in that hypertext system. In this perspective, the model is a first step towards an application generator, in a similar way as CASE tools are used in software engineering environments. Schemas can also be reused so that applications with similar structure can make use of the same schema, varying only the particular instances and their contents.

We present here the “Hypertext Design Model” (HDM), which supports a model-based approach to authoring-in-the-large. HDM prescribes the definition of a schema, in which the classes of information elements are defined both in terms of their presentation characteristics, the types of their possible connections, and of their internal organization structure. A schema can be instantiated, allowing the definition of a particular application, by giving instances of information elements and of connections types. Many connections of a schema instance can be automatically inferred from the corresponding schema definition. The dynamic behaviour of an application must be specified using some browsing semantics, and HDM is not closed with respect to any particular browsing semantics. We have specified a default browsing semantics, more suitable for relatively simple card-oriented classes of systems.

Before presenting HDM, we briefly review current approaches to hypertext application design.

1.2 Current Approaches

We are not aware of other approaches in the hypertext field that try, in a more or less formal way, to deal with hypertext design and modelling with a similar “authoring-in-the-large” and application oriented philosophy as HDM.

In the very closely related data base design field, models have played a crucial role in bringing the data base development task from being an “handcrafted” (and sometimes inconsistent) activity into becoming a structured and rational process, based on well defined

²This last statement is perhaps less obvious for span-to-span links, that appear to connect parts of nodes to other parts of nodes.

design methods [Wiederhold 83]. Database models were born as a 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 [Brodie 84, Hull 87]). 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 hypertext's peculiar features.

The work of Lundh et alii [Lundh 89] with the Hybris system can be regarded as a preliminary step towards data models for hypertext. However, as it was proposed, it is still mainly concerned with providing hypertext interfaces to conventional databases, by including the database model in the hypertext, as opposed to developing a data model specific for hypertext applications.

The Object Lens system [Lai 88] provides an object-oriented environment allowing the definition of "templates", which may correspond to high-level abstractions in an application domain. Even though its main objective is not the construction of hypertext applications per se, Object Lens may be used to construct a rudimentary hypertext system as a special case, through the use of the linking mechanism between object instances. Object classes, however, do not control in any way how particular object instances are linked.

The IDE [Jordan 89] and g-IBIS [Conklin 88] approaches attempt to explicitly model the semantics of their application domains adopting predefined structures which reflect such "deeper" semantics. Therefore, they are obviously extremely powerful for applications in their class of domains but, by their very nature, are less general purpose as generic model-based approaches. Nonetheless, it is worth discussing each one of them briefly.

IDE, an extension of NoteCards [Halasz 87a,90], is an interactive design and development system which assists instructional designers with the process of creating complex instruction material. IDE provides representation primitives for describing the substance of a course and the rationale for the course design in terms of hypertext structures. IDE moreover, provides "structure accelerators" that function as macros to create a locally-connected set of related nodes. Starting from the observation that many parts of a disciplined hypertext which models predefined domain knowledge structures would inevitably have many repetitive node/link/node local structures, the "acceleration" facility permits authors to create entire webs of nodes and links in a single operation. Note that these accelerators are defined on a syntactical basis. The template facility in Intermedia [Yankelovich 88] plays a similar role.

g-IBIS 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. It does so by adopting a well defined theory of the design process and by providing a set of specific nodes and link types that specifically represent conceptual objects in the domain model. Thus g-IBIS encourages to share this model while seeking to discourage less disciplined argumentation modes. g-IBIS adopts complex recursively defined structures as its central organization idiom; such structures represent a recursively defined structure of a "rhetorical argumentation" from which design discussion may be composed, thus reflecting the structured, step-wise, but indefinitely recursive process through which systems designs are proposed and refined.

Other modelling 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. Nevertheless, we briefly discuss some of the most important ones.

The goal of the Dexter Hypertext Reference model [Halasz 90] 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 block 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 contents 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 [Garg 88] is a mathematical formalization of hypertext networks viewed as static, syntactic structures; no attempt is made to help the designer capture some domain semantics of a hypertext application. Garg's model (as other similar ones such as [Richard 90]) assumes low level objects as building blocks, such as "standard" nodes (and even node components), and node-to-node links. The model has been designed to illustrate abstraction mechanisms useful to describe or derive static syntactic properties of hypertext networks. Garg provides a mathematical framework to define, for example, named collections of information units (aggregation), or collections of "similar" information units (generalization), or attributes on the information units.

The Trellis model [Stotts 89] is mainly a "behavioural" model for hypertext. Hypertext networks are modelled as Petri nets and several sets of various browsing semantics (that is, how information is to be visited) are discussed in terms of Petri nets computations. The Trellis model completely abstracts from the contents and structure of the nodes, which can be arbitrarily complex information structures (including an entire hypertext). We will use a modified version of it to formalize the run-time behaviour of some hypertext applications.

Tompa [Tompa 89] adopts a hypergraph formalism to model generic hypertext structures and to formalize identification of commonalities in these structures. It is in some sense similar to the Trellis model, but is also able to directly refer to "groups of nodes" having a common link semantics. This mechanism can be regarded as a syntactic device that allows reference to "abstract" objects, although no explicit reference is made to any abstraction in the application domain. A dynamic behaviour 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, sometime without being explicitly advocated by the system's author.

In Hypercard [Atkinson 87], for example, linear structures play a central organization role. Even if each cards 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 [Brown 87,89] also prescribes the extensive use of linear structures to clarify the organization of hyperdocuments. It is significant to note however that while Guide and Hypercard both prescribe that hypertext contain extensive linearity, they differ in their motivation; Hypercard uses linearity to provide a built-in, consistent navigational opportunity, Guide uses linearity to provide continuity between information chunks.

KMS [Akscyn 87] 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, beside purely hierarchical links, KMS allows "special links" that are cross-hierarchical, and induce a more complex topology on the network.

Finally, sharing with model-based approaches the goal of helping the author master hypertext complexity, several existing approaches provide "template" facilities to help designers generate hypertext structures more easily and systematically. Templates allow 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 (as opposed to the foreground, which is specific of each card).

Hypergate style pages [Bernstein 88] and Information Lens templates [Malone 87] also let writers create as many instances of a class of hypertext nodes as they require, while NoteCards' notion of a hierarchy of node types and link types encourages a similar style of node creation [Halasz 87].

Other approaches help to organize and modularize existing hypertext material. Trigg's Guided Tours and Tabletops [Trigg88, Marshall 89], for instance, are extensions of NoteCards which help designers group and visualize existing hypertext material in such a way that it is appropriate for a particular reader. Tabletops are means for collecting a particular set of cards on the screen and for capturing their lay-out. Guided tours are networks of Tabletops; they can be used to model at a high level the domain conceptual structure of a portion of hyperbase, and to provide preferred paths through a NoteCards hyperbase tuned to specific goals of the reader.

As will be seen in the sequel, the model proposed here shares with some of the above mentioned approaches the emphasis on preferred structures as building blocks, and with IDE and g-IBIS the goal of representing a (simplified) semantic model of the domain.

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 also presents HDM in detail; section 3 gives two examples of HDM use in hypertext application design; section 4 presents a formal definition of HDM; and section 5 draws conclusions, both by comparing HDM to other approaches and by discussing future work.

2 A Model-based Approach to Authoring-in-the-Large

2.1 Motivations

As mentioned in the introduction, there are many advantages to having a design model, which we summarize here. The features described below become evident especially for application classes whose application domain exhibit regularities, where many applications in that domain have similar structures, where typical applications are very large, and that have few authors and many more readers. Typical examples of such applications are technical documentation, training and educational material [Jordan 89], and auditing systems [De Young 89].

- Improvement of communication

Design models provide a language in which an application analyst can *specify* a given application. Thus they facilitate the communication between the analyst and the end user (i.e., the client, in most cases); between the analyst and system designer; and between the system designer and implementor, when they are different persons. They can be used to *document* the application. This provides support for users of the application; it helps the maintenance of the system; and it serves as a common language in which to compare applications when desired. At the very least, a basis for discussing the similarities of the applications exists.

To paraphrase Halasz et al. [Halasz 90],

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* [Winter 89], Harvard University's *Perseus*, ACM's compilation *Hypertext for Hypertext* [ACM 87], Eastgate's *Elections of 1912* [Bernstein 89], have in common?" ; "How do they differ?"

- Consistency

An interesting related aspect, which has received little attention in the research literature is the task of proof-reading a hypertext. It is clear that proof readers need to check links as well as text, and that specious or accidental links can be as embarrassing as more conventional typographic errors. This task should be greatly helped by the availability of a design language, preferably with (at least semi-) automated support.

- Reusability

As a matter of fact, the availability of such a language paves the way for (partial) reuse of the back-bone structure of applications, since these models capture the “essential semantics” of applications, and can therefore be reused when the semantics of the two applications are similar enough.

- Development of Design Methodologies and of Rhetorical Styles

Design models provide a framework in which the authors of Hypertext applications can develop, analyse and compare *design methodologies* and *rhetorical styles* of “hyperauthoring”, at a high level of abstraction. This analysis can be done without having to resort to looking at particular visualizations (screen formats and appearances, button functionalities and the such) or to the detailed contents of units of information. At this level, it is possible to analyse the “conceptual” organization of the application domain knowledge represented, and examine its adequacy for the intended uses.

- Providing predictable reading environment

A recurring issue in the hypertext field is the “navigation problem” (for example, see [Landow 87], [Utting 90], [Parunak 89], [Nielsen 90], and many others), in which readers get “lost” when navigating in the hypertext structure. It is clear that tools for understanding and specifying hypertext structures can help readers master complex documents and can also help authors and editors avoid structural inconsistencies and mistakes. An additional expectation is that applications developed according to a model will result in a very predictable representation structure. As a consequence, navigation environment for possible readers should also be predictable, thereby reducing the “disorientation/cognitive overhead” problem.

- Use by Design Tools

Design models can be used by *Design Tools* [Walker 88], much in the same way as application generators are based on languages to specify classes of applications, or as CASE tools have specification languages to describe software (at various levels of abstraction). Such tools support a systematic top-down (“structured”), model-based development process, allowing the designer to work at a level of abstraction which is closer to the application domain, and supporting systematic translation process to the implementation level. Hypertext, however, requires particular primitives which render direct use of existing tools very cumbersome.

To be able to provide the advantages just described, a design model clearly must be *expressive*, allowing the description of concepts at the appropriate level; it must be *complete*, in the sense that it captures the most frequently occurring structures; and it must be *simple*, so that it can be easily used by authors. It is clear that these models never will neither describe *all* applications, nor *completely* characterize a given application (analogously to models for database applications). Still, one may hope to have at least *classes* of applications adequately characterized by design models.

The next sub-section describes HDM which is a first step towards defining a model for hypertext applications. HDM can be considered, in some respects, relatively simple minded;

however, as most design models, it is in constant evolution, being improved as more experience is gathered from using it for concrete applications.

2.2 . HDM Primitives

According to HDM, an application domain is seen as being composed of *Entities*, which in turn are formed out of hierarchies of *Components*. Entities belong to a *Type*. Entities or Components can be connected to other Entities or Components by typed *Links*. Components can be instantiated by one or more *Perspectives* into *Units*. An HDM *Schema* is then a set of Entity and Application Link type definitions. An HDM *Schema Instance* is a particular set of entities and links defined according to a given schema.

Once a Schema Instance has been defined, it can be operationalized via the specification of a particular *Browsing Semantics*, that gives the runtime behavior of the application. Even though we have not provided primitives in which to specify such Browsing Semantics, we have provided a *default* one, that is compatible with card-oriented hypertext systems.

We will next elaborate each of these notions in more detail.

2.2.1 Entities and Components

An *Entity* is a (large) set of information that represents some concrete or conceptual real world object of the application domain; examples of entities are “Law 19/8/89” and “Verdi’s La Traviata”. Typically, an entity will denote something quite complex, whose description usually must be structured to be conveniently described. An entity in its whole can be represented through several individual, pieces of information called *Components* organized into a (possibly ordered) *hierarchy*.

A Component is a piece of information describing something about an Entity. All components of an entity are homogeneous with respect to their possible perspectives (see next section). Examples of possible components (with the corresponding entity from above) are “Article 1” (“Law 19/8/89”), “Overture” (“Verdi’s La Traviata”) (which in turn is also decomposed into smaller pieces).

Many authors have observed that hierarchies are very useful to help user orientation when navigating in hyperdocuments [Acksyn 87, Brown 89]. HDM recognizes this via the notion of entities made up of components organized into hierarchies. There are many criteria (i.e., domain relations) for inducing hierarchies, such as “is-part-of”, “is-abstraction-of”, etc... HDM does not define a-priori any particular semantics to the hierarchy-inducing relations; authors may take advantage of the regularities of hierarchies if one such relation is present in the application domain by structuring entities according to that relation.

Entities and components should help authors organize the application domain concepts in a “natural” way for the user to access by establishing a uniform, consistent navigation context.

2.2.2 Perspectives and Units

One of the important aspects of Hypertext is precisely its “multimediality”, i.e., the capability of presenting information in many forms in an integrated way. It is natural in this context that information be presented in several different ways (e.g., media) - text, graphics, images, sound, etc... Moreover, this information may be expressed in different languages (e.g., English, Italian, Portuguese, etc...) as well as in different rhetorical styles (e.g. discursive, synthetic, schematic, etc...). There is however the notion that in some sense this information refers to the *same* subject.

This heterogeneity of presentation properties associated to the different perspectives introduces a further degree of complexity in the design of hypertext applications (w.r.t. other applications such as database or traditional knowledge base applications). An arbitrary, inconsistent use of media and rhetoric styles would cause confusion and disorientation in the reader, and would reduce the usability of a large hypertext application. The use of perspectives should be therefore disciplined to obtaining a readable, pleasant application, and should be somehow specified in the design phase [Glush89] [Even89].

HDM facilitates this design dimension by having *Perspectives* for components. By the term *Perspective* we mean the way to present a piece of information: the media (e.g., text, graphics, sound, video, etc.); the rhetoric style (e.g., discursive, schematic, formal, semiformal, etc.); the language; or any combination of these, adopted to describe its content. The description of a component according to a given perspective is called *Unit*.

A *Unit* provides a referential context to a piece of information; this information constitutes the *Unit's Body*. For example, suppose there is a "Law" entity whose components are its constituting articles. If "Law" can be seen from the "Official Text" and "Description" perspectives, then for each article there will be one unit whose body is its "Official Text" and another whose body is its "Description". These units provide a context in which to refer to the (texts of the) article as part of the "Law".

As another example of perspectives, we may say that the component "First Movement" (of entity "Beethoven's Ninth Symphony") has a "Textual" and a "Music" perspective, and the corresponding units have bodies that are, respectively, a written musical notation and a played rendering of the "First Movement".

If one tries to relate the concept of *Unit* to the usual hypertext notions, one may conceive of the actual nodes that the reader sees physically as, in HDM terms, units of components. Units might then correspond to the usual hypertext notion of "node" (see section 2.3).

Two *Units* may share the same body. In the "Law" example above, there may be another entity, for instance "Contract", where one of its components, say "Legal Background", represents a literal inclusion of one of the "Law"'s articles, say "Article 3". This component would also have an "Official Text" perspective; the body of the unit corresponding to the "Official Text" perspective of "Legal Background" would be shared with the "Official Text" perspective unit of component "Article 3" in "Law". Note that the context of the reference to the text of the article within the "Law" entity is completely different from the reference in the "Contract" entity, even if the text itself is the same. As will be discussed in section 2.2.7, this is the only type of sharing allowed in HDM.

Since HDM is mainly concerned with authoring-in-the-large, which abstracts from the details of contents of nodes, it purposefully does not further exploit the notion of perspective, and says nothing with respect to the internal structure and visualization properties of bodies. In fact, bodies of units could be, in the general case, compositions of uninterpreted pieces of information, even if in most cases it will be simply a constant value. For example, it could be a piece of text, of music, a collage of short pieces of music taken from a CD, or a composition of video images, etc...

2.2.3 Entity Types

An *Entity Type* groups entities having common properties. In HDM, the properties chosen as relevant are

- “using the same set of perspectives”,
- “being broken into components according to the same criteria”, and
- “being related to other sets of entities in the same way”.

Therefore, when specifying an Entity Type, the author is saying that all entities belonging to that set have the same set of possible perspectives, are split apart into components in the same style, and share the same type of potential outgoing and incoming links to and from other types of entities. Examples of entity types (with the corresponding entity from previous examples) are “Law” (“Law 19/8/89”), “Opera” (“Verdi’s La Traviata”).

The incoming and outgoing links between entity types are potential since, in general, the actual existence of a link will depend on the particular entity instances involved. For example, one may have a link of between entity types “Person” and “Opera” to signify that that person is the opera’s author. Since many persons never composed any operas, there will be many entities of type “Person” that do not have any link to an entity of type “Opera”.

Since the set of perspectives associated to an entity type is indicative only of *possible* perspectives for its entities, it is necessary to have the notion of a *Default Perspective*. The intended meaning of the default perspective is that *all* entities of this type have *at least* this perspective; further significance of this concept will become clearer when we discuss the process of mapping HDM structures into conventional node-and-link structures.

We will defer further discussion about the relevance of these properties regarding entity type definition until we have talked about links, in the next section. As far as perspectives are concerned, the notion of sharing perspectives between members of a type reflects a philosophical point of view with respect to (large) hypertext authoring: it will be less confusing for readers if the same type of concepts be presented in an uniform fashion. For example, that *all* “Opera”s have a “Textual” (the score) description, as well as a “Music” (e.g., CD recording) description, regardless of the particular opera one might be “talking” about.

2.2.4 Link Types and Links Instances

The major advantage of the hypertext paradigm is that one may organize an information base in a non-linear fashion. This means, loosely speaking, that pieces of information can be related to each other through links associated to them. These links may be included inside or “next-to” the information itself, depending on the hypertext model adopted.

A moment of thought will show that the success or failure of a given hypertext application is heavily dependent, among other things, on the appropriate choice of links. It is quite clear that the *meaning* of the link can vary greatly, much in the same way as in semantic networks. The more the meaning approximates the relationships in the application domain, the more the user will be at ease in “using” the corresponding hyperdocument, since links will evoke familiar associations. In addition, besides the representational role (capturing domain relations) links have, they also have a navigational role (capturing navigation patterns). The more these links are designed in a consistent and predictable way, the easier it will be for users to follow them.

One of the benefits of HDM is the identification of classes of links that have fundamentally different nature. Once this is recognized, it is possible to treat them in different ways; for some classes of links a “standard” interpretation can be readily provided, thus helping the authoring process. This will also have consequences even on the way information regarding links is presented to the user, as discussed in section 2.3.

In what follows we describe the three classes of links present in HDM.

Perspective Links

Given the notion of perspectives of components, a class of links is immediately induced - links allowing one to switch between presentations (units) of the same component. For example, to be able to examine the "Text" perspective of "Beethoven's Ninth Symphony" when examining (i.e., listening) to its "Music" perspective. Clearly these links can be derived from the definition of the components; we will have more to say about this in section 2.2.7.

Structural Links

A Structural Link connects components belonging to the same entity, expressing some relation induced by the hierarchical organization of the entity. For these components, traversing hierarchical links has familiar meanings such as "Next Brother", "Previous", "Up" (e.g., to move higher in abstraction level), "Down" (e.g, to get more "details"), etc....Structural links, therefore, provide navigational paths over the structure of a given entity.

HDM does not prescribe any particular applicative meaning to the hierarchy-inducing relation within an entity. Nevertheless, properties of this relation may be used in several circumstances, for instance by specialized processors. Consider for example that a linear description of an entity must be generated. For some kinds of hierarchical relations, we must take the body of the root, followed by the linearization of its sons (according with a predefined strategy, say "from left to right", as in KMS [Aksyn 87]). For other kinds of relations, only the leaves need to be considered, the higher level nodes being seen as just a way to show how to organize the leaves. Note that this procedural use (and any other) of the applicative semantics of the hierarchical relation lies outside HDM per se.

From this previous discussion, we have identified at least two styles of fragmenting and organizing entities into components, which we have called respectively "by-refinement" and "by-continuation". These styles correspond to classes of hierarchy inducing relations. Clearly, these styles are not meant to be all-inclusive, but in the course of our research these were enough to describe the situations encountered. We describe each of these in turn.

The by-refinement style is used in two situations. The first one occurs when an entity has an intrinsically hierarchical structure. An example is provided by entities that stand for concepts which are physical assemblies of several others, each represented by a component. For example, an entity "Car" whose components are the "parts" of the car.

The second situation is when artificial refinement is introduced as a rhetoric device. Assume, for example, that a procedure consists of 3 activities, each one in turn arranged into several steps. An organized description will probably not describe it all in one shot, especially if is very complex, but rather by saying it is made of activities A, B and C, and activity A consisting of step 1, 2 and 3, and so on.

In both situations each sub-component is "part-of" the parent component. The difference is that in the first situation the notion "part-of" is almost physical, while in the second situation the same notion represent a conceptual, artificial refinement.

The by-continuation style is typical of the situation when one is describing a linear document and "breaking it up" into chunks, to be examined in turn. In this case, the links describe possible "continuations" the reader may follow after having read the current chunk. For example, in describing a technical document, it may first describe the introduction (general part), then indicate each possible section to be read in turn, then for each section the first part is described, followed by its successors, and so on.

As a design guideline, the hierarchy inducing relation used to structure entities into components should provide a consistent navigation environment for the reader, which should be used for all entities of a given type.

Application Links

The third class of links in HDM are the *Application Links*. Whereas structural links capture rather “standard” semantics (structure), Application Links embody semantic relationships in the domain. That is, they represent some (arbitrary) relationship between entities that the author deems meaningful, in the sense that this relationship evokes some association between concepts useful to the user of the hyperdocument.

Therefore, by providing an application link, the author will be making it “natural” to the user to access some information which is “related” to the information being read at that point, simply by traversing that link.

It should be quite obvious that, being application dependent, it is not possible to automate a priori the processing of such links. However, the notion of links between *classes* of entities helps the authoring process, since it provides a mechanism to state things at a more appropriate level of abstraction and also imposes a certain discipline on it.

An *Application Link Type* is defined as a set of links instances whose source and destination entities are of the same entity type, respectively. For example, if the author specifies that the “Symphony” entity type has a link (type) to the “Persons” entity type named “Composer”, this means that in principle all instances of “Symphony” (e.g., “Ninth”, “Italian”, etc...) may have a link to a corresponding “Person” (e.g., “Beethoven”, “Mendelsson”) that composed it; “Composer” is the application link type.

2.2.5 Outlines

From the primitives presented so far, it is possible to identify at least two degrees of “stability” (i.e., change in time). The first is at the level of Entity and Link Types - they vary very little, if at all, across instances in the application domain. In a similar sense as in the database field, an HDM schema represents “stable” hypertext application dependent information, shared by *all* applications modelled as instances of this schema. A second degree is at the level of schema instance - the navigation paths specified in it should be shared by most uses of the particular hypertext.

It should be kept in mind that, since navigation is a prime concern for hypertexts, an HDM schema should be designed in a way that captures (at least partially) the more common navigation paths in the hypertext application; the primitives described so far encourage this.

On the other hand, an important part of many applications is providing the initial access to the hypertext, before the user starts navigating at all. More generally, it is useful to envisage navigation patterns (including the starting points in the hypertext) which are *superimposed* onto the network itself, in a spirit similar to Guided Tours [Trigg 88]. These patterns are heavily dependent on both the particular user profile and the task to be performed. As a consequence, there will be several such patterns even for the *same* hypertext, depending on the type of user (e.g., novice, expert, etc...) and on the task to be accomplished when using the hypertext (e.g., perusal, search for specific information, etc...).

HDM recognizes this need by allowing (and encouraging) a different style of usage of the entity type primitive, referred to in this context as an *Outline Type*. An Outline Type is a special type of entity (and therefore has in principle a hierarchical internal structure) whose instances have leaf components which “point to” (are linked by application links to) nodes of the hypertext proper. This usage allows different applications to share the same hyperbase

(i.e., same schema instance), but have different outline types and instances, corresponding to different access paths.

A typical example of an outline is a hierarchical index of the contents of a hypertext. The main difference from including this index into the hypertext itself as a “normal” entity type is the criteria for organizing this index. Typically, the organization of the index is opportunistic, as it is much more based on task and user profile information, than on “stable” application domain criteria. For example, an index for an expert user that must find specific information will certainly be different from another index organized for the training of novice users, even though the information itself may be the same. Further examples of outlines will be described together with HDM application examples.

2.2.6 Derivation of Links

Having hierarchies as primitive concepts suggests that, at design level, an author needs to specify only a minimal set of structural links- those necessary to define the structure of an entity (“parent” and “next-sibling”) - from which a large number of other implicit structural links can be derived. At the authoring level then, only application links are actually put in by the author; other structural links, such as parent-son, to-top, etc... are induced by the ordered hierarchic structure. The same is true for perspective links, which can be automatically derived from the definition of a component.

In a more general fashion, it is possible to utilize some of the semantics of links by regarding them as relations between entities (components). Properties of these relations, such as symmetry and transitivity, when present, can also be used to derive other links. This becomes particularly powerful when used in conjunction with composition of relations. Under this point of view, structural link derivation is simply a particular case of composition and transitivity.

As an example, assume that a section of a contract (component of an entity of type “legal document”) is connected to an article of a law (component of an entity of type “state law”) by a link of type “justified-by”. Then one can also say, at a different level of detail, that the whole contract “is-justified” by the whole law. To represent this, if we assume that the root component of an entity is intended to represent the whole entity (in this case a refinement hierarchy), the author would set a link of type “justified-by” between the root of the first entity to the root of the second one. Fig.1 shows how his link can be derived straightforwardly as simple composition of the link “justified-by” between the two components (“Section 1.1” and “Article 2”), the link “to-top” between “Article 2” and the root of its entity (the “Law”) and the structural links between the “Contract” and “Section 1.1”.

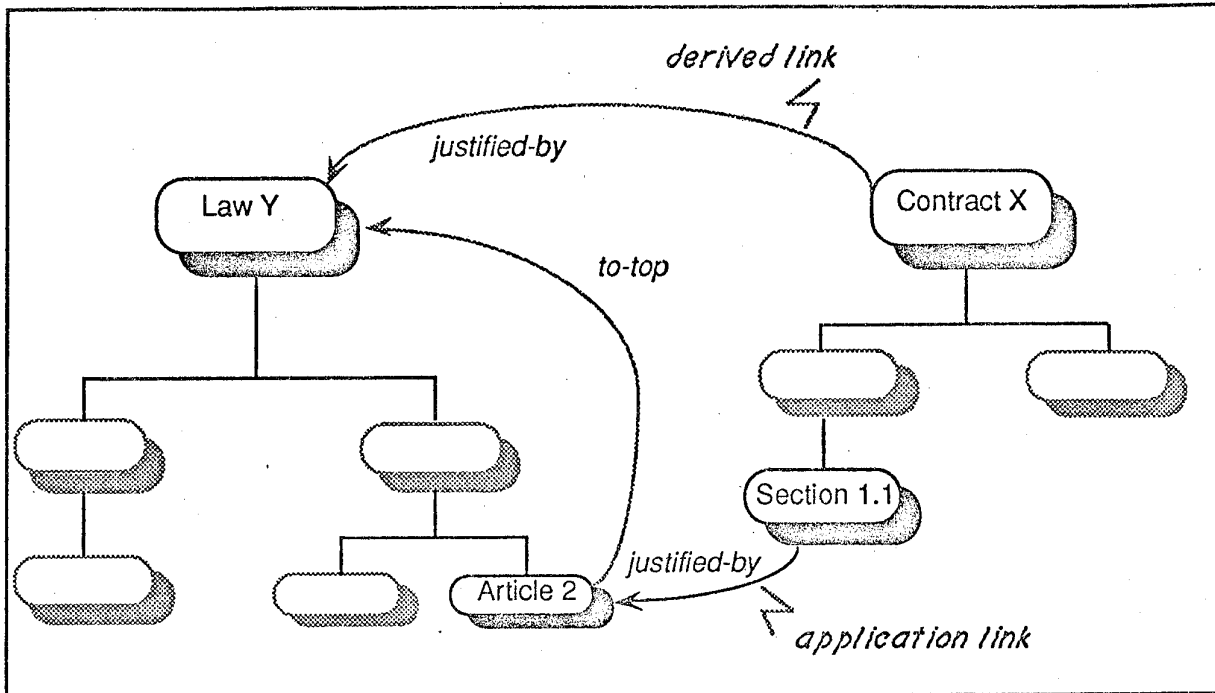


Figure 1- Example of derived link

So far, HDM does not include a language to specify such derivation rules. Once such a language is available, with its corresponding interpreter, HDM can provide a great amount of conciseness to the process of model specification - the author needs to provide a much smaller amount of links than the number of links that will actually be present both at the conceptual level and at the concrete level.

2.2.7 Schema and Schema Instance

A class of hypertext applications in a given domain can be characterized via the notion of *Schema*, which is defined as a set of entity type and link type definitions. A schema characterizes classes of application because it does not specify actual entities and links, but rather general properties of potential entities, and potential links between them.

The discussion on derived links in the previous section indicates that, in reality, it is useful to specify derivation rules at the schema level. In particular, each entity type definition should include also structural link derivation rules, as the instances of each type may require different navigational patterns inside its structure. Furthermore, the derivation rules for application links are naturally specified together with application link type specifications.

A particular application in a given domain is specified by *instantiating* a schema, i.e., by giving entities and links as instances of the types defined in the schema.

The notion of schema and schema instance allows, in many cases, the reutilization of the *same* schema for different applications in the same domain. These applications should share the same *global* structure, but differ in the *particular* instances which are actually present. A further level of reutilization may be obtained when new applications preserve the *local* structure (structural links inside entities and application links between them) in a schema *instance*, but redefine the *bodies* of the units.

A schema instance characterizes the “static” aspects of a specific application. To specify the “dynamic” (runtime) behaviour of an application, one must add a particular browsing

semantics, as will be discussed in section 2.3. This again adds another reuse dimension, in which the same static specification is maintained, but a different browsing semantics is used, generating a different running application. By varying the browsing semantics, it is possible to have the same conceptual application, whose running versions are implemented in several different hypertext systems.

Examples of schemas and schema instances are given in section 3.

2.2.8 Sharing

An important issue in hypertexts is the notion of sharing. Given the large amount of information present in many hyperbases, and the fact that the same information may be used in several different places, it is natural for many authors to simply isolate the shared information into a chunk of hypertext, and set up links to it whenever necessary. This allows the information to be included only once in the hyperbase, and be referred to from all points; in HDM terms, this could be represented either by sharing Components among Entities or by sharing Units among Components or finally by sharing bodies among Units. HDM actually allows only the sharing of bodies among Units, for reasons we discuss next.

Sharing of Components destroys the notion of Entity, since structural links now must be qualified with respect to which entity they refer; analogously, the sharing of Units destroys the notion of Component. Among others, an important point in the notions of Entity and Component is to provide a meaningful context for navigation to the reader, since links have very strong navigational semantics attached to them (this is in fact one of the differences from links in semantic models for conventional databases).

A careful examination of the notion of sharing of bodies between Units will show that it is enough to model the desired degree of sharing, at least for "well-structured" documents. This allows the preservation of the "navigational" contexts provided by Entities and Components, and yet does not require actual duplication of information in the hyperbase. It also helps to clarify the notion of Unit: it is a *referential* context for a piece of information contained in its body. It is clear that any development tools supporting HDM must provide primitives to manage this type of sharing.

2.3. Browsing Semantics

The actual navigational use of a hypertext is largely defined by its *browsing semantics* [Stotts 89]. To summarize, the browsing semantics will determine three further *design* choices:

- 1) What are the objects for "human consumption";
- 2) What are the perceived links between objects;
- 3) What is the behaviour when links are activated.

The first aspect that should be clear is that many times objects present at the authoring level may not be available to readers. Thus, the first point above has a precise meaning in the context of HDM - objects can be either Entities, Components, or Units. The second point refers, in HDM, to how Links are perceived.

The third point deserves some further considerations. Link activation entails a series of effects perceptible to the reader, and these must be specified. However, most of these effects, such as whether the source node remains visible or not, how the body of the destination is presented (it may require processing by some specialized processor), etc..., are authoring-in-the-small concerns, since they regard essentially what happens within the nodes. From the authoring-in-the-large point of view, in HDM terms what must be specified is the activation of one-to-many links.

HDM, as discussed so far, does not prescribe a-priori any particular browsing semantics for hypertexts specified with it. In fact, there are many levels in which this browsing semantics may be specified, from the most abstract one, in which the answer to question 1) says that perceivable objects also include entity types and link types, to the most concrete one in which the answer says that perceivable objects are only units (nodes) and links between them.

Clearly, the particular browsing semantics will dictate the navigation patterns the user will actually be allowed to follow, and is also dependent on the particular hypertext system being user for implementation. Our work on browsing semantic definition and specification at a generalized level is still at a preliminary stage.

However, it is still possible to discuss the implications of choosing a particular browsing semantics with respect to a given HDM specification, and to formalize it (see section 4.2). In other words, we can give (at least partial) answers to the question "what are the further design choices that must be made with respect to a given design specification, once a particular browsing semantics has been chosen"?

In the next section, we exemplify this discussion for a class of simple browsing semantics, that of plain *node-and-links* found in many hypertext systems³. This browsing semantics is considered to be the default semantics for HDM specifications.

2.3.1 Default Browsing Semantics

In this class of browsing semantics, we assume that no abstract objects (entities and components) are visible- only units (corresponding to the usual hypertext notion of node) can be perceived by the readers as concrete objects. As a consequence, readers can see links only among units and so, in the end, actual connections must be established among units. Furthermore, we also assume that only one node is active at any time, and from an active node only one link can be traversed at any time. We will call *concrete links* the connections among units, as distinguished from *abstract links*, which are defined among entities and/or components.

This particular class of browsing semantics raises the issue of how should one translate the links between entities and components into the corresponding (concrete) links between units.

2.3.1.1 Perceivable Objects

There are several possible decisions of the actual concrete links to be included. An obvious requirement is that concrete links should be defined consistently with the specification of abstract links. For example, a simple choice for application links is the idea of having a *default representative* for each abstract object (component or entity).

The default representative for a component is its unit in the default perspective of its type (which, by definition, always exists - see section 2.2.4). The default representative for an entity is the default representative of its root component. This corresponds to saying that the

³ The example discussed has been implemented in a prototype system, using a relational database description of HDM instances, and generating a tabular description that can be imported into Hypercard and manipulated using Hypertalk. This prototype is described in [Schwabe 90a]

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 representatives⁴.

A simple rule for translating component-to-component application links is one in which each link corresponds to a *set* of links connecting each unit of the source component to the default perspective unit of the target component.

To illustrate this rule, consider the situation in which there is a “Ninth Symphony” component, linked to a “Beethoven” (of type “Person”) component, through a “Composer of” application link. Consider further that the “Person” entity type has perspective types “Graphic” (with a picture of the person) and “Text” (with a biography), and “Ninth Symphony” has “Text” and “Music” perspectives, the “Text” one being the default.

In this case, the actual links (corresponding to instances of the link connecting the two components at the abstract level) will connect both the “Ninth Symphony:Text” and the “Ninth Symphony:Music” units to the “Beethoven:Text” unit that is the default perspective for “Person”; this is illustrated in Fig. 2 below.

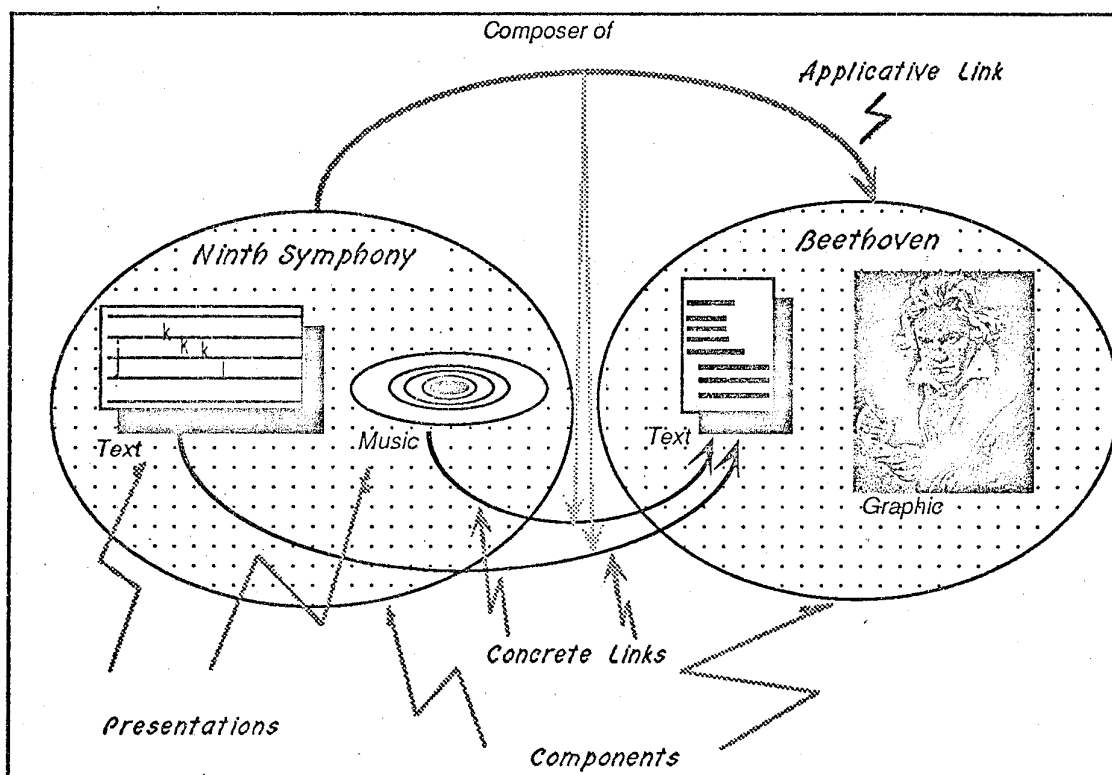


Figure. 2 - Example of concrete link generation

This is an example where one link at the abstract level corresponds to two links at the concrete level. It should be stressed that this is only *one* of the possible choices that can be made, and HDM is not strictly prescriptive in this respect.

A rule for component-to-component *structural* link translation is the following: if a component C1 has a structural link to a component C2, then, for each perspective P, each unit

⁴ Note, however, that the representative mechanism can also be seen as a way of allowing the perception of abstract objects. In other words, the user could still be “thinking” (conceptually) of an entity, even if she is actually perceiving a unit corresponding to this entity’s root component. In this view, one could say that the browsing semantics allows the perception of abstract as well as concrete objects.

of C1 having this perspective should be linked to the unit of C2 having the *same* perspective. This rule expresses a kind of stability criterion w.r.t. the use of perspectives- if the reader is looking, say, at the “Text” perspective of an article of a law, and follows the structural link “next article”, she will see its text perspective (as opposed, say, to the “Graphic” perspective, which might be the default.).

It is also interesting to note that the above rule, in general, cannot be used for *application* links since different entity types will have different perspectives. Nevertheless, a weaker form of the rule can use the notion of “compatibility”⁵ of perspectives, instead of identity. For example “Official Text” of a law and “Description” of a banking procedure may be considered “compatible” since both are textual. In this situation, if the user is looking again at the “Official Text” perspective of a law, and follows the “Affects Procedure” link, she will see the “Description” perspective of the procedure (which might not be the default).

Note that all the rules described so far need not be mutually exclusive- they may be applied in some pre-determined order, until all component-to-component links have been mapped into unit-to-unit links.

Another kind of link introduced (derived) is the *Perspective Link* which connects units of the *same* component. Perspective links allow the user to look at the same object switching between perspectives. For example, switching to the “Text” perspective of a procedure when looking at its “Graphic” perspective.

An important point to stress regarding the translation process is that it actually introduces another design dimension. Each choice made when deciding how to translate abstract links to concrete links affects the final hypertext the user will see. For this reason, it is quite natural to think that these choices can be made according to certain user profiles, therefore allowing the same hypertext design to be used for different classes of users. Other similar aspects are discussed in the next section on perception of links.

2.3.1.2 Perception of Links

In hypertext, connections among pieces of information are usually perceived through “anchors” (or “buttons”). Anchors are well identifiable areas on the screen that show the existence of a connection, and can be selected by the reader in order to get into the link target(s).

We claim that, in a disciplined design, anchors should have, if possible, different visualization properties according to the different meaning of the links they represent, and anchors sharing the same semantics should have similar visualization properties (e.g., similar icons, standard position on the screen, etc.).

This approach suggests the need of a new primitive, which captures the notion of “anchor type”. An anchor type specifies the properties of anchors that represent links of the same type - or of a group of link types. It might be often the case that what the reader actually “sees” in terms of anchors is a restricted set- the author may have chosen to group link of some type under anchors of a common anchor type.

Consider for example, that a “Procedure” entity type may have a link of type “Formal Legal Justification” to a “Law” and a link of type “Informal Justification” to an “Informal Regulation”. The author might want to provide to certain classes of users a single reading link, maybe labeled “Justification”, since the distinction might not be important for readers of that

⁵ The notion of compatibility here may be related, for instance, to the medium or the rhetoric style used. In any case, it is clear that it is not always applicable, as it depends very much on the application domain.

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 possibility for the author is to hide links for a given type for a specific application (since they might be relevant at design and specification level but not at reader level).

Anchor types, therefore, provide a mechanism for the author to present groups of link types together, also renaming or hiding them as well, as desired. By assigning a set of link types to each anchor type, the author can control visibility.

2.3.1.2 Link Activation

From the previous discussion, it is clear that there are many situations in which an anchor actually refers to several possible destination nodes. It must be defined, therefore, what happens when the user activates one such anchor. Clearly, this is determined by the particular browsing semantics being used, which in turn is partly dependent on the particular system being used to implement the hypertext - if it supports multiple active windows, for instance, a possible choice may simply be to show all destinations, each in a separate window. Depending on the particular media the information of the node is stored in, a different effect of this choice is simple to activate all destinations in parallel, as in the case of text and music, for instance.

The previous solution, however, is often not acceptable, and oftentimes not even possible in many systems. One possible approach that can be adopted to deal with this limitation is to introduce the notion of *chooser*. A chooser is a structure associated with a single-source-multiple-target anchor that allows the selection of one of the multiple targets of the anchor. Choosers can be generated automatically from the specification of the concrete links in the hypertext, using any available mechanisms present in the implementation environment (e.g., menus).

3. Examples of Hypertext Modelling with HDM

This section will present two examples to illustrate HDM. The first one describes an application that was developed *starting* with the model. The other one is an existing application which is included to demonstrate the utility of HDM in describing applications even if they were not developed using HDM.

The presentation of each example will necessarily be quite sketchy, since it is not intended to describe them in full detail, but rather to illustrate different aspects of the model. It should also be kept in mind that we are chiefly concerned with hypertext *structure*, and therefore nothing will be said about the appearance of the nodes of the examples.

Finally, the second example should be read with the caveat that, being done "a posteriori", it has an extra degree of arbitrariness in the sense that we will only be describing a *possible* model for that application, chosen from the many other possibilities with the chief criterion of better illustrating HDM concepts.

3.1 Expert Dictionary for Banking Environment

This example describes the information handled by a credit organization⁶. The goal is to have an organized way to access and refer to a large set of information of vastly different (but inter-related) nature.

⁶This example is a subset of a larger prototype application (named Expert Dictionary) [Garzotto 89] developed by ARG SpA and Politecnico di Milano within the european Esprit projects INDOC and SUPERDOC.

This organization manipulates *Documents* according to *Procedures*. The reasons why *Documents* and *Procedures* are the way they are can be explained by looking at *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.

The above state of affairs is captured in the schema described in Fig. 3., where Entity Types and Application Link Types have been specified. Since in this case all application links are by-directional (representing a relation and its inverse) - this is frequently (but not necessarily) the case - we have drawn the links only once.

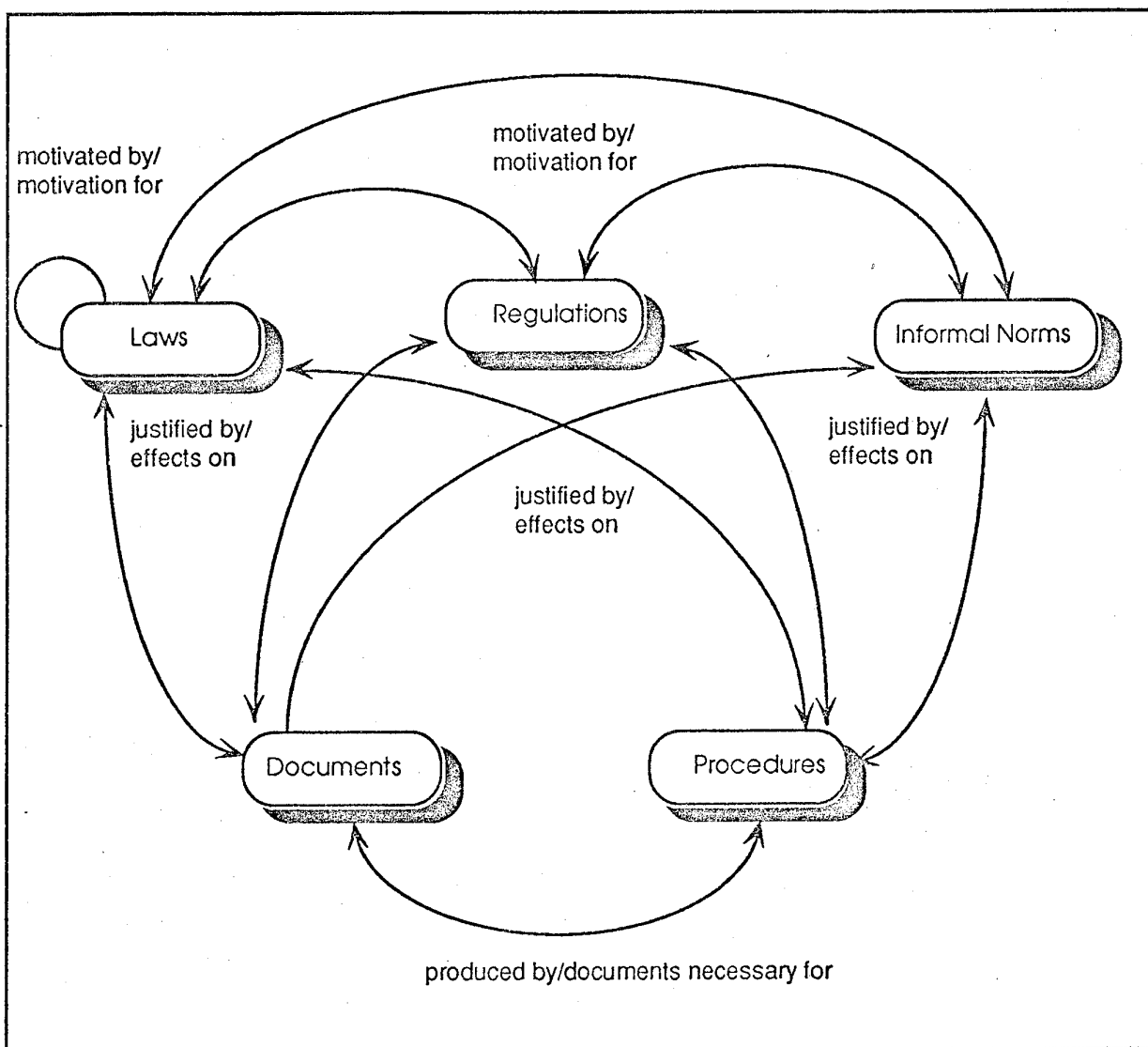


Figure 3. Schema of Entity Types and Application Link Types

Each of these Entity Types has a set of *Perspectives* associated to it. We do not enumerate all of them here, but give a few examples (the one marked with a "*" is the default perspective):

- *Laws and Regulations* have *Structure** and *Official Text*;
- *Documents* have *Structure** , *Official Text* and *Description*;
- *Procedures* have *Flow Diagrams** and *Description*

Let us look now at a fragment of an instance of the schema above, depicted in Figure 4. In the figure, shaded areas denote entity instances whose internal structure has been further detailed.

There is an entity of type *Procedure* named *Mortgage Loan Procedure*, which is made up of several steps. The first step is named *Preliminaries*, and one of the intermediate steps is named *Request Acceptance and Entry*. This step is, in its turn, also made of several steps, the second being named *Verification of Request* and the third being named *Request Data Entry*. As we can see, the *Mortgage Loan Procedure* is really a hierarchy of sub-procedures, each represented by a component.

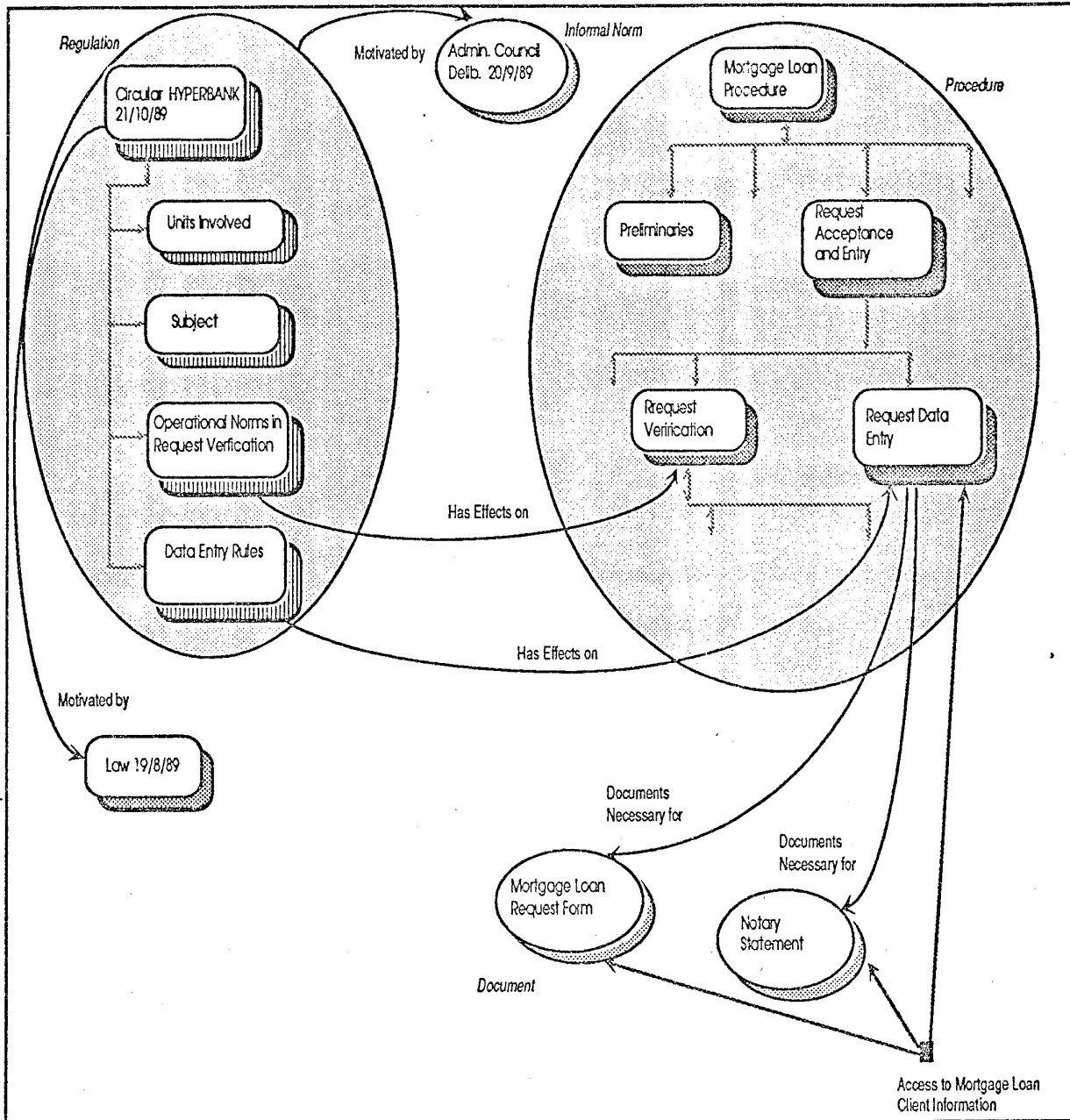


Figure 4 - Instance of Schema in Fig. 3

Another entity present is *Circular HyperBank 21/10/89* of type *Regulation*. made up of 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 *Verification of Request* and *Loading Request Data* are performed; therefore, an application link of type *has-effects-on* is present between them. *Circular HyperBank 21/10/89*, in turn, is *motivated-by* *Law 19/8/89*

Finally, we have included an access structure, i.e., an instance of an Outline, that allows the user to access directly the nodes describing all documents needed for the compilation of a Mortgage Loan Request

From the (schema) instance exemplified above, it is possible to arrive at an implementation using some existing hypertext system. In this section, we show a few examples of screens (corresponding to HDM units of the appropriate components) from the Hypercard implementation of this system. This application was generated using the browsing semantics described in section 2.4.1.

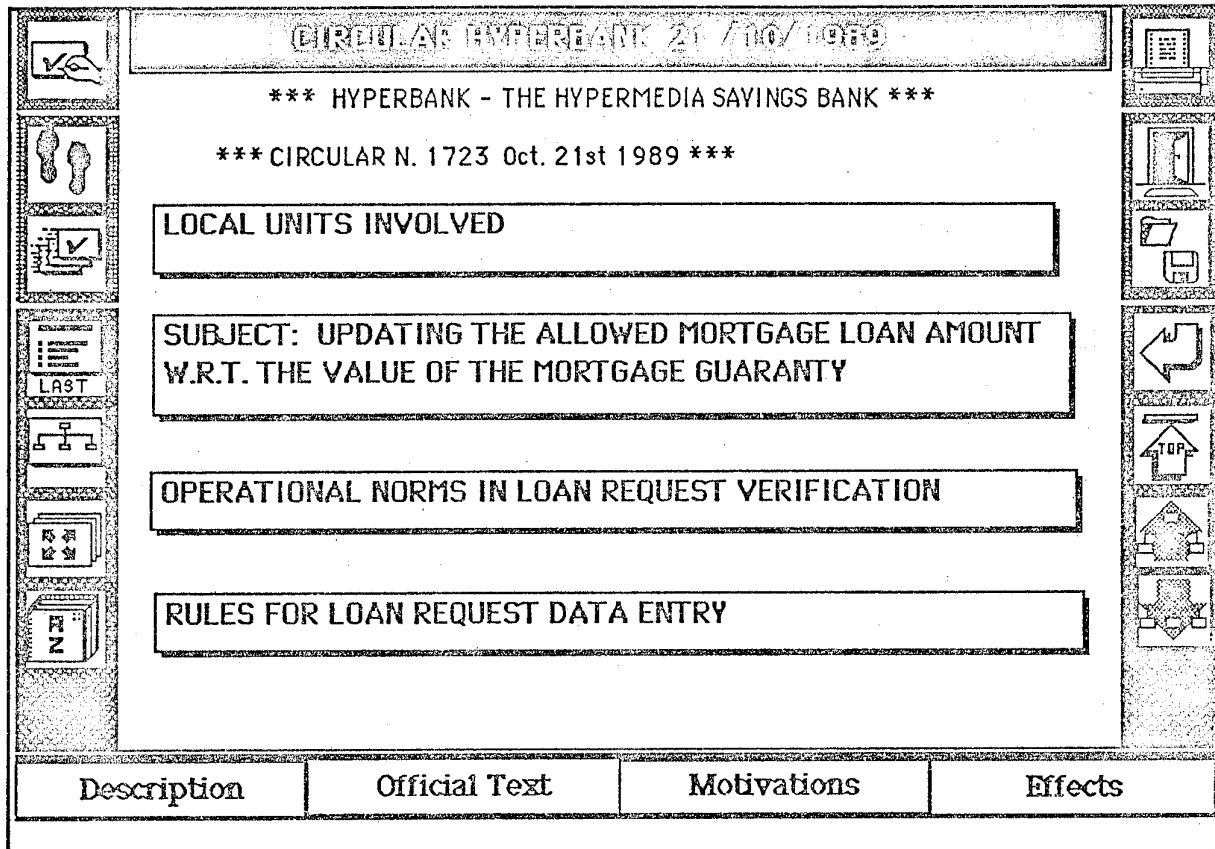


Figure 5 - Structure perspective of a Regulation

In Fig. 5, the user is looking at the "Structure" perspective of the Regulation instance "Circular HyperBank 21/10/89". Notice that all the links of type "Justified by" have been grouped under the anchor (button) "Motivations". To change perspectives and look at the "Official Text", the user presses the button with the same name (in this case), seeing the screen shown in figure 6. Besides the anchors corresponding to the application relations ("Effects", "Motivations") and the perspective links, all of which are at the bottom of the screen, there are other anchors on the sides, some corresponding to other structural links (e.g., "Top"), others corresponding to functions of the system (i.e, are not associated with any link), such as "Mark" and "Trace" at the top right hand.

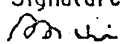
CIRCULAR HYPERBANK Oct 21st 1989			
CIRCULAR HYPERBANK N. 1723 - Oct. 21st 1989			
to: LOCAL UNIT MANAGERS, CLIENT SERVICES DEPARTMENT MANAGERS			
Subject: UPDATING THE ALLOWED MORTGAGE LOAN AMOUNT W.R.T. THE VALUE OF THE MORTGAGE GARANTY			
We inform that the existing rules, to be adopted by the local 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  Dr. M. Bianchi Client Services Dept. Director			
Structure	Description	Motivations	Effects

Figure 6- Official text perspective the Regulation in Fig. 5

Next, the user is interested in seeing the effects of this regulation, by following the application link "Effects On". Since this is in reality a one-to-many link, the user must choose which of the possible destinations she wishes to go to, from the appropriate chooser (see section 2.4.1.2). In this case, the chooser was implemented as a card with buttons inside, corresponding to each possible destination; it could also have been implemented as a menu. Notice that, for some hypertext systems supporting multiple windows, a possible alternative authoring choice at this point would be to activate one window for each destination node (but this is advisable only when there are few of those!). This is shown in Fig. 7.

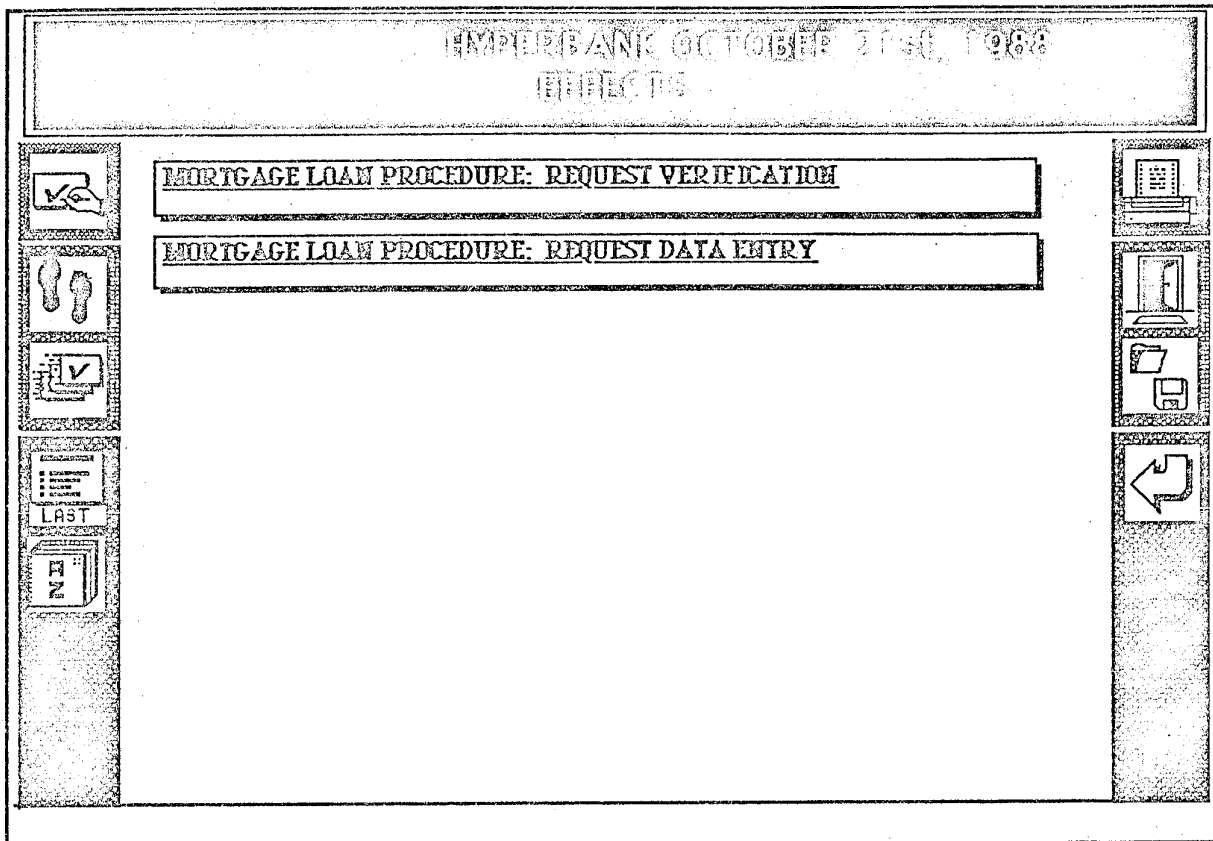


Figure 7 - Chooser allowing to follow the multiple destination "Effects" link of the Regulation in Fig. 6

Once again, notice that multiple target anchors are detected at "compilation time", and a chooser card is mechanically generated without direct intervention of the author.

Supposing the reader has chosen the second alternative, she will see the screen shown in Fig. 8, which shows the "Description" perspective of a procedure step.

MORTGAGE LOAN PROCEDURE LOAN REQUEST PRELIMINARY VERIFICATION					
	<p>The local unit employee who receives a loan request must check the request form and the annexed documents. The purpose of this preliminary check is to ensure the complete and correct compilation of the form, and the completeness and consistency of the annexed documents.</p>				
	<p>In particular, we recommend that the following data be carefully checked:</p>				
	<p>- LENDING PARTY PERSONAL DATA: it must be THE SAME AS those registered at the General Register Office; THE PARTY MUST HAVE OFFICIAL RESIDENCE IN ITALY.</p>				
	<p>- GUARANTY: the guaranty on the loan is a mortgage on any kind of building. Its mortgage value must satisfy the following constraints (w.r.t. the amount of the requested loan):</p>				
	<p>* if the lending party is a physical person and the purpose of the loan is the purchase of his/her legally declared residence, then the requested loan amount CANNOT be higher than 75% of the mortgage value</p>				
	<p>* for any other purpose, the requested loan amount CANNOT be higher than 50% of the mortgage value.</p>				
	Structure	Previous phase	Next phase	Motivations	Docs. Needed
					Docs. produced

Figure 8- Description perspective of a procedure

Supposing now the reader wishes to understand better how this step is related to the rest of the loan procedure, she may request to see the "Structure" perspective, and navigate in it. The next screen (in Fig. 9) shows the "Structure" perspective of the "topmost" node of this perspective for the "Mortgage Loan Procedure"; it can be reached by clicking on the "Top" button, third icon from the bottom on the right hand side of the screen.

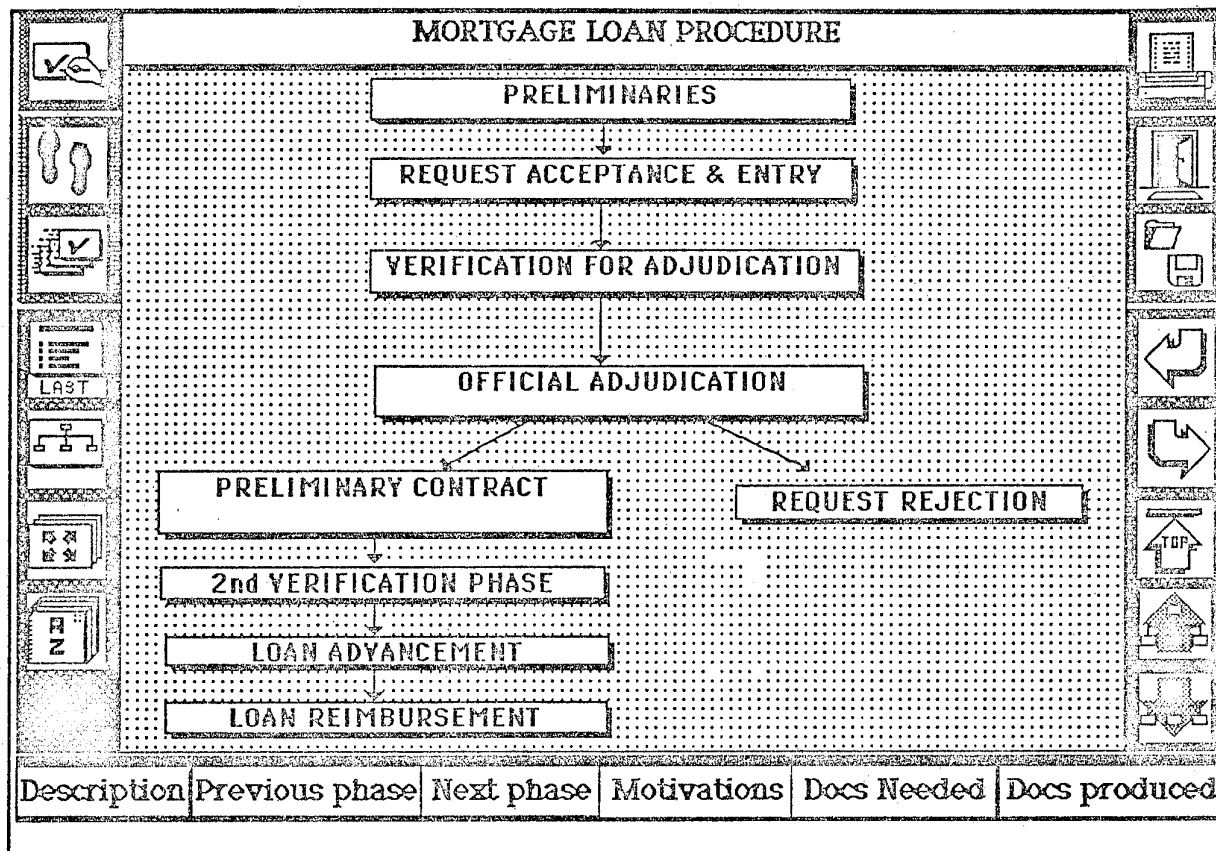


Fig. 9 - Structure perspective of the top node of procedure in Fig. 8

Notice that these last navigation steps were done over the structural links derived from the internal structure of the entity.

3.2 Voyager Company's CD Companion to Beethoven's 9th Symphony

The Voyager Company's CD Companion to Beethoven's 9th Symphony [Winter 89] is a commercial Hypercard application ("stack") that is essentially a listening guide to Beethoven's 9th Symphony. This guide provides several perspectives to that work of music, beyond letting one to listen the symphony in various ways - musical commentary, historical notes, a short introduction to the "art of listening" and even a game (which we do not model).

The "stack" is made of approximately six hundreds cards, containing social and cultural historical notes about Beethoven's time and life, multiple meanings of Beethoven's work, technical comments about the symphony, specific musical components or passages, and general musical concepts. As much as possible, this information is related to actual musical passages or brief motives and themes of Beethoven's work, taken from a compact disc containing an orchestra performance of the symphony. When this is not possible, samples of digitalized music are played directly by the computer.

With this set up, not only can the user read about the description of a given component of the symphony, or a given musical passage or musical concept; with the click of the mouse she can also cause the CD or the Mac to play this precise component or motive.

Before going on to describe a possible HDM model for this application, we would like to emphasize that it reflects a *possible* view of it, which in all likelihood is not shared by its

authors. In other words, it reflects our “understanding” of it (with all its mistakes and misconceptions about it), and is being used solely to illustrate the usage of HDM concepts.

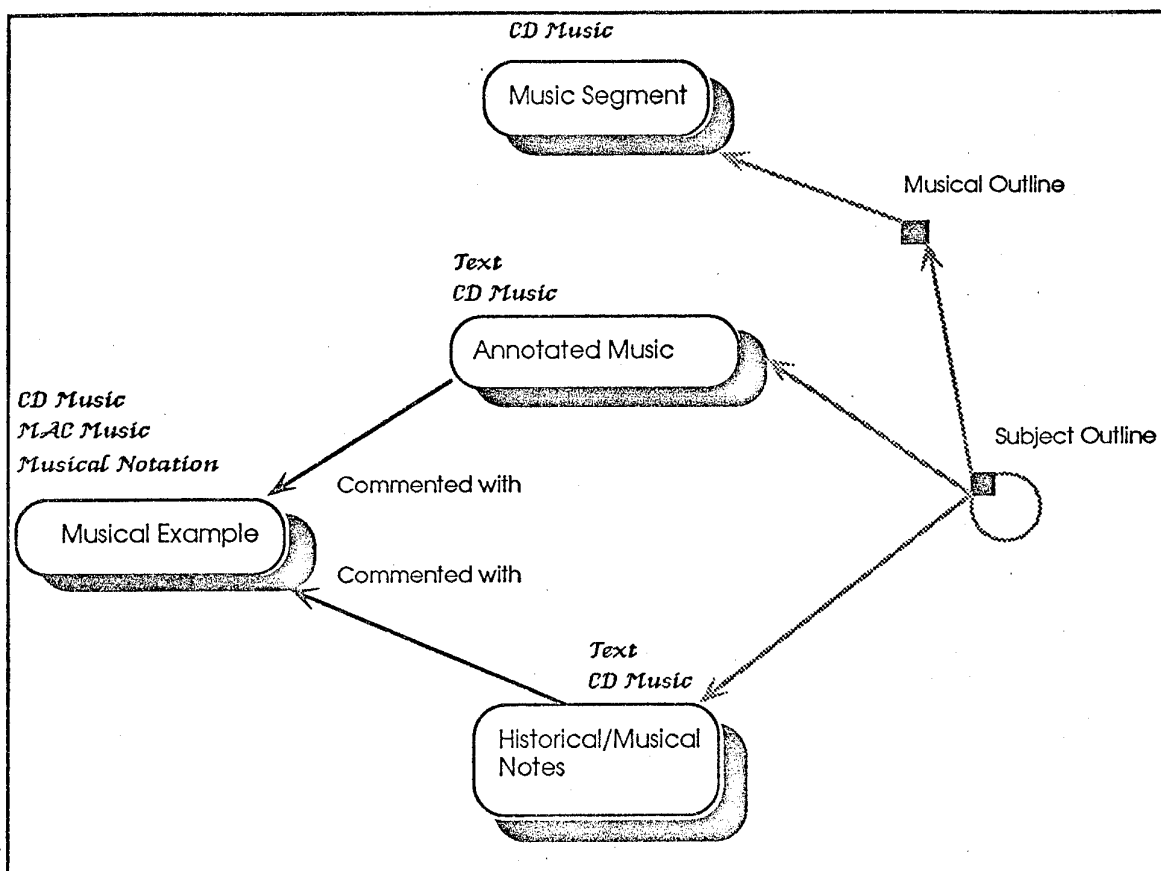


Figure 10 - HDM Schema for Beethoven's 9th. Square nodes indicate Outline types.

Figure 10 shows an HDM Schema for Beethoven's 9th. The “Historical/Musical Notes” entity type is used to describe the information concerning social and cultural history of Beethoven's time and life, multiple meanings of Beethoven's work, and technical comments about the symphony. This information may be seen from two perspectives: “Text” and “CD Music”. Entities of this type are built in the “by-continuation” style.

Beyond the information in “Historical/Musical Notes”, there often is a musical example that is used to illustrate certain points about a note. These examples are represented in the “Musical Example” entity type, which can be seen from the “CD Music”, “Mac Music”, and “Musical Notation” perspectives. The instances have no internal structure (i.e., are “hierarchies” of only one node).

One of the ways to listen to the symphony is by listening to it “passage by passage”, simultaneously looking at comments about that passage, and at eventual textual representations of it (when it is sung). This is represented in the “Annotated Music” entity type, which again can be seen from the “Text” and “CD Music” perspectives. The instances are built using the “by-refinement” style.

Another way to listen to the music is simply by selecting an entire segment - represented by the “Music Segment” entity type, with only the “CD Music” perspective. Instances of this type are built using the “by-refinement” style.

To access the entity instances of the types described above, there are two types of outline available: "Subject Outline", which lets you choose between instances of "Historical/Musical Notes" and "Musical Outline"; "Musical Outline" simply lets you choose between instances of "Music Segment".

A moment of observation of the schema immediately raises the question "How come entity type "Musical Example" only has *incoming* links?". The answer is that instances of this entity are to be taken as "comments" about the instance (of "Historical/Musical Notes" or of "Annotated Music") to which it is attached. Taken as such, one may look at the comment (by following the link), but from there the only possible action is to "go back" (or backtrack) to the original ("commented") node. Therefore, the return link is of a "functional" (navigational, run-time) nature, and does not need to be present; this behaviour is similar to "pop-up" text in Guide.

An interesting point to observe is that, once this schema has been defined, it could be (re) used to describe other musical works, without any change - it is enough to define a new instance of the schema. In other words, the schema captures a particular authoring style to describing symphonies.

Figure 11 shows an instance of the above schema describing the CD Companion to Beethoven's 9th. Only the main instances of entity and outline types are depicted, with the corresponding types shown in italics. Note that we have not shown instances of "Musical Example", since they make sense only when one examines the *internal* structure (i.e., the components) of each of the entity instances mentioned above. We will comment on some of these instances in order to further illustrate HDM modelling features.

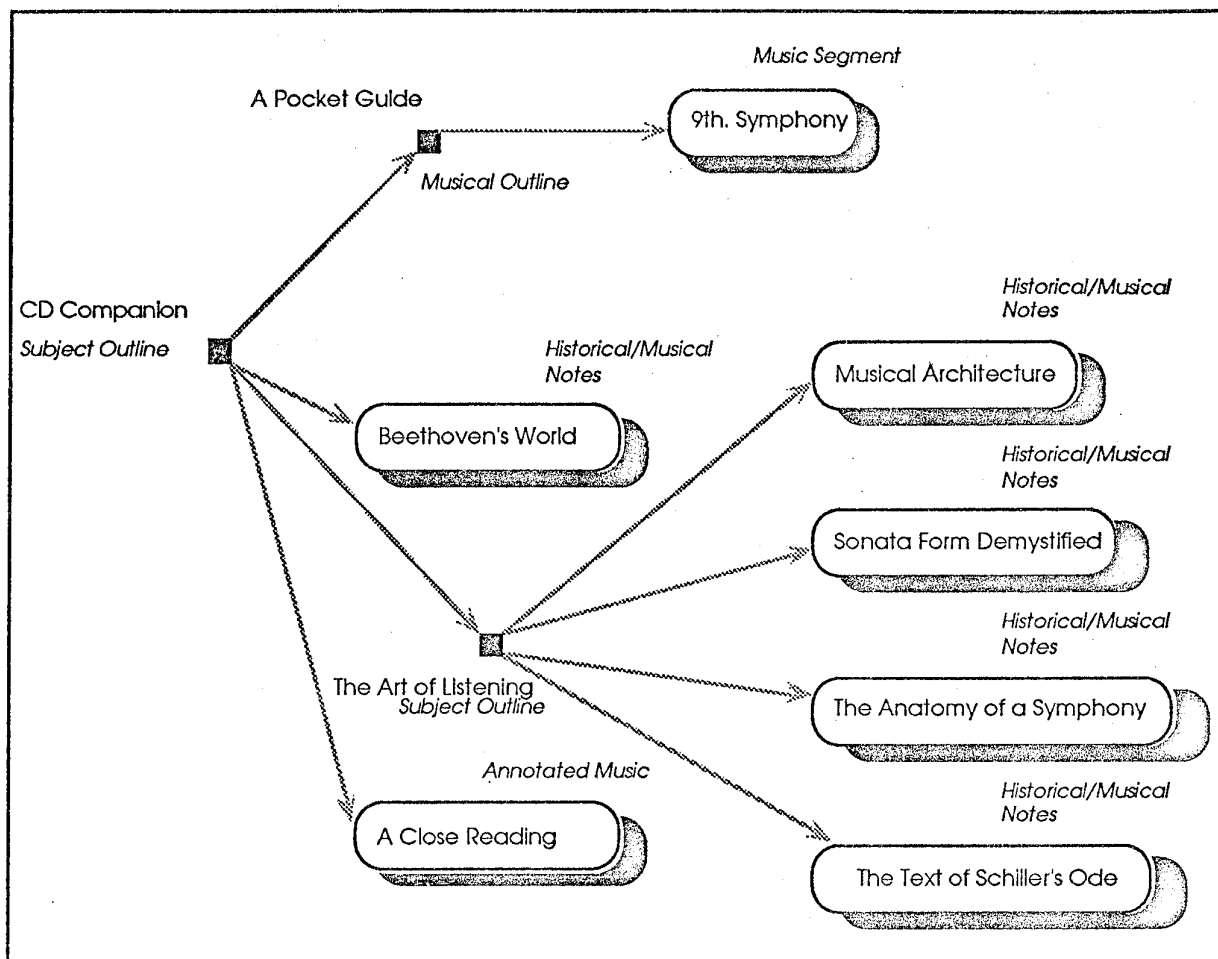


Figure 11 - Outline structure of CD Companion to Beethoven's 9th Hypertext. The type of each element (corresponding to the types in Fig. 10) is indicated in italics next to it.

The "A Pocket Guide" outline simply points to an instance of "Music Segment" corresponding to the symphony itself, whose structure is shown in figure 12.

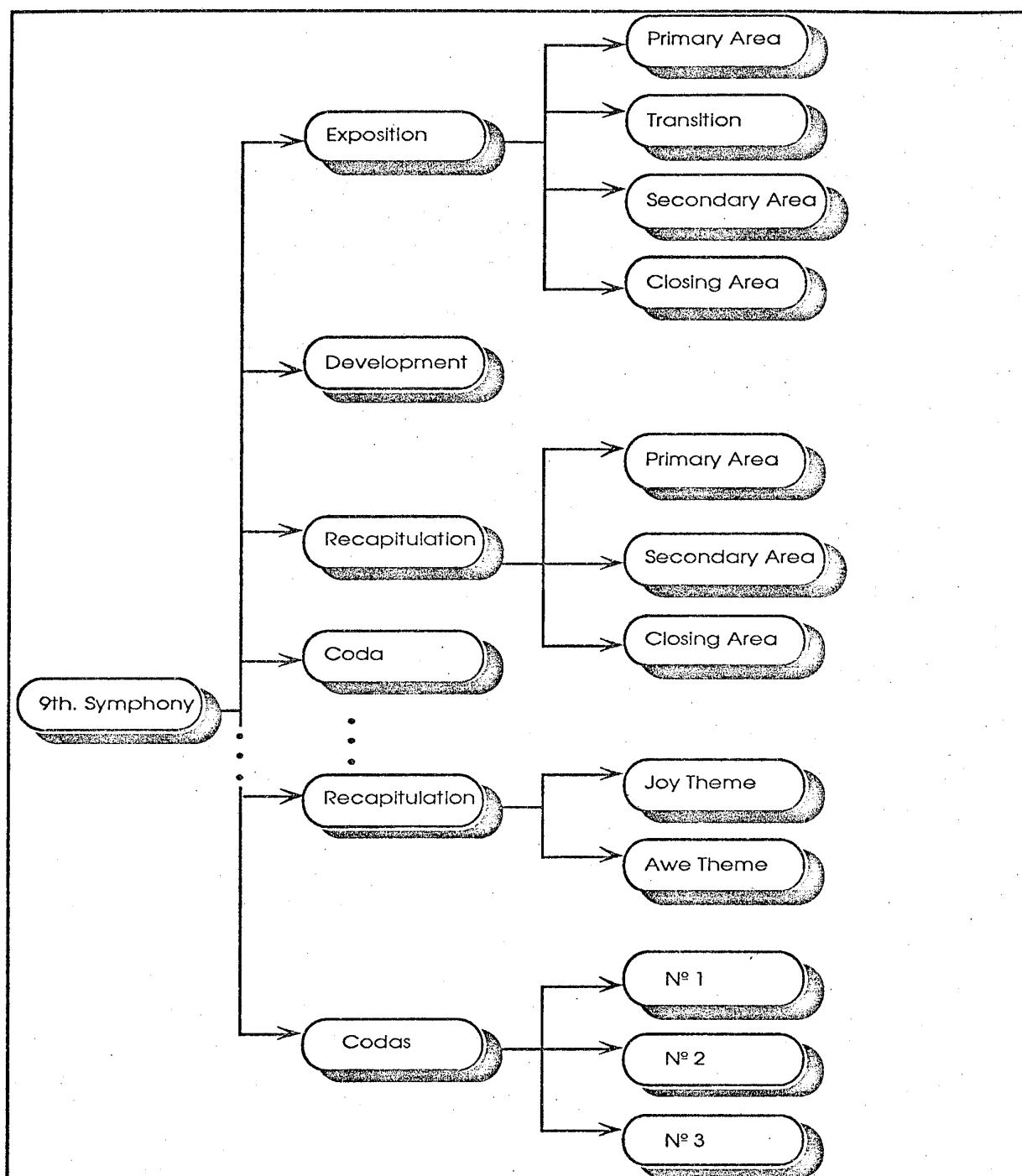


Figure 12 - Internal structure of entity “9th Symphony”, of type “Music Segment”.

Figure 13 shows the single card corresponding to this outline node, with the anchors allowing you to access the corresponding components highlighted. In fact, from this figure it is possible to see that the outline really points not only to the root, but also to all intermediate nodes as well as the leaves of the hierarchy. In this case, accessing an intermediate node of a “by-refinement” structure means that all of the leaves under the node are “displayed” - which is in fact just playing the CD music of each one of them.



LUDWIG VAN BEETHOVEN Symphony No. 9			
A • P O C K E T • G U I D E			
1st MOVEMENT Sonata Form	2nd MOVEMENT Sonata Form [Scherzo]	3rd MOVEMENT Sectional Form	4th MOVEMENT Sonata-Concerto Form
Exposition	Exposition	A-Section	Open. Ritornello
Primary Area	Development	B-Section	Exposition
Transition	Recapitulation	A-Section varied	Horror Recitative
Secondary Area	Binary Form [Trio]	B-Section	Joy Theme
Closing Area	First half	Interlude	Turkish Music
Development	Second half	A-Section varied	Development
Recapitulation	Scherzo da capo	Coda	Recapitulation
Primary Area	Coda		Joy Theme
Secondary Area			Awe Theme
Closing Area			Codas
Coda			Nos. 1 2 3
  INDEX ↑ GLOSSARY ↑			

Figure 13 - Card corresponding to the outline node that points to "9th Symphony", of type "Music Segment". Note that the anchors correspond to nodes in Fig. 11.

Let us look now at the internal structure of entity "Beethoven's World", of type "Historical/Musical Notes". This entity is meant to be a sequence of pieces of information (corresponding to one card), divided into chapters. Each piece may have one or more "Musical Example" entity instances linked to it. The reader may at any point go to the next card, or jump to the beginning of any of the chapters.

In HDM terms, "Beethoven's World" is simply one single entity with a hierarchical component structure. Beside the basic (*parent and next-sibling*) structural links, it also has additional links induced by the structure to allow the navigation described in the previous paragraph. Figure 14 outlines this structure; in it, each chapter is represent as a "column", whose first node is the chapter's name. Some of the induced structural links are represented in gray, to give an idea of the internal structure of the entity; not all of these induced links are included to keep the figure simple. The grayed nodes stand for instances of "Musical Example", which are to be regarded as "commentaries" to the text.

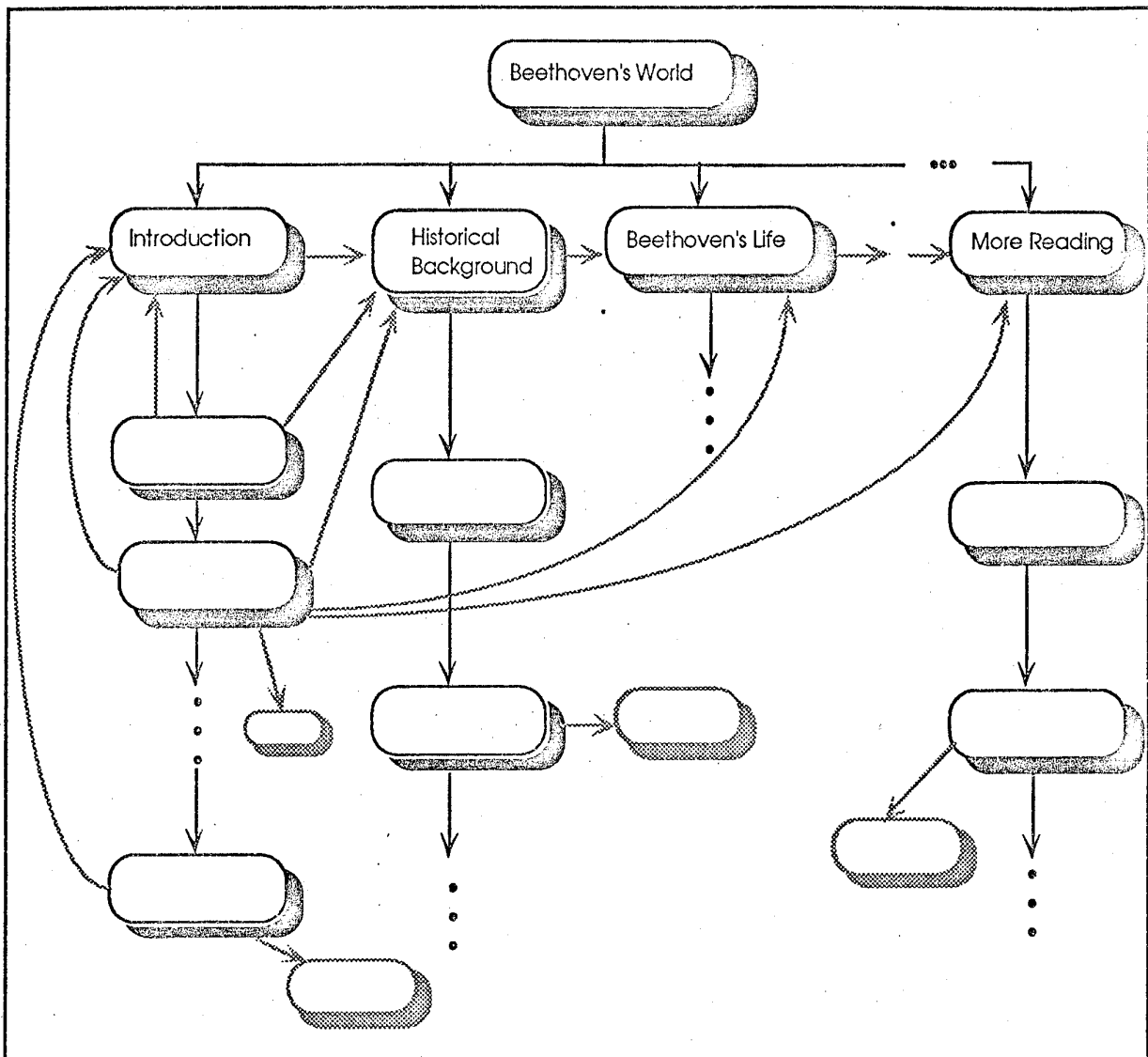


Figure 14 - Internal structure of entity "Beethoven's World", of type "Historical/Musical Notes".

Reading of "Beethoven's World" is then essentially a (depth-first) traversal of this tree. When a "Musical Example" instance is present, the reader may "go to" the node corresponding to it, but from there she may only go back to the text itself.

Beethoven's World

What, then, is a Beethoven sketch? There were actually several kinds. On the one hand, there were simple one-line notations for themes. Here is a typical example of a theme considered for the finale of the Ninth Symphony but not used:

PLAY SKETCH THEME

This idea was later adopted as the main theme for the last movement of Beethoven's String Quartet in A Minor, Op. 132, from 1825.

INDEX ↑ CHAPTERS ↑ GLOSSARY ↑ FIND AGAIN 60 of 124

Figure 15 - Typical card of "Beethoven's World", corresponding to some unit of some component in Fig. 14. The highlighted button allows the access to a "Musical Example" instance.

Figure 15 shows a typical card of the entity "Beethoven's World", corresponding to a unit in its "Text" perspective. In this figure, the button labeled "Play Sketch Theme" is an anchor that stands for the link to an instance of "Musical Example" (see the schema in figure 10). To move inside this entity, along structural links (its internal structure is depicted in figure 14), one uses the buttons (i.e., anchors for structural links) at the bottom: "Index", "Chapters" and the right-hand arrow symbols. "Index" and "Chapters" allow "jumps" into particular intermediary points in the hierarchy, whereas the arrow symbols allow sequential traversal.

The entity "A Close Reading", which is an instance of the entity type "Annotated Music" has internal structure very similar to "9th Symphony" of type "Music Segment" (see fig. 12). In fact, types "Annotated Music" and "Music Segment" are similar, differing basically by the fact that the first has more perspectives. Fig 16 contains an outline of the internal structure of "A Close Reading"

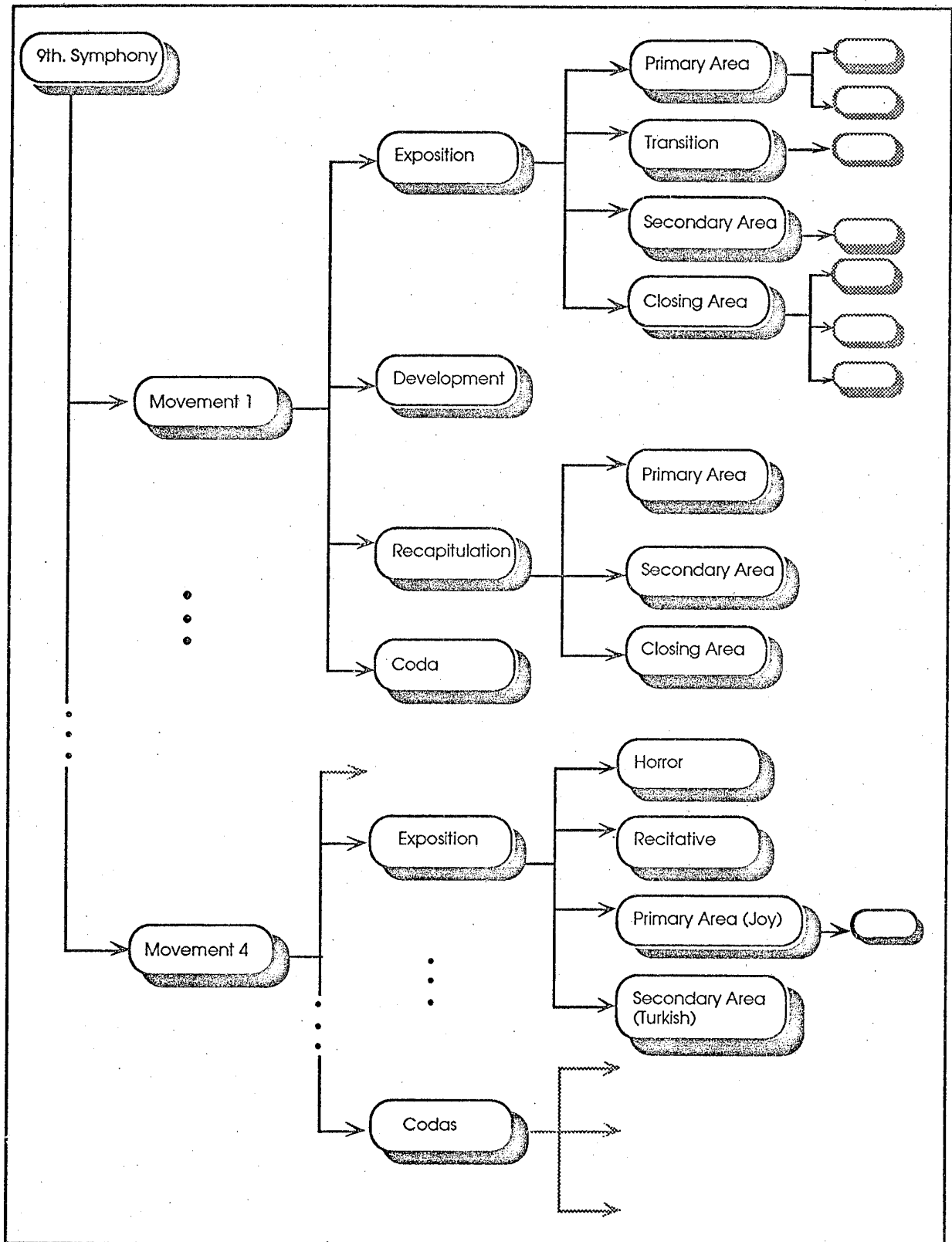


Figure 16 - Internal structure of "A Close Reading". Grayed nodes stand for the smallest piece of music that can be played individually.

In figure 16, the leaf nodes (grayed) represent the smallest pieces of music from the symphony that can be played individually, or that have a comment of their own; for simplicity, we have not shown all of them. Each of these nodes may also have one or more “Musical Example” instances attached to them; these were also not included for simplicity’s sake.

A CLOSE READING

MOVEMENT 4
Allegro assai [Very fast]

Soloists: a a' b a' Chorus: b a'

[313] Just when it does not seem possible to generate more excitement, the b-a' choral entries do just that, enlivened by a series of electrifying orchestral • trills.

Stanza III:
She gave us kisses and the ripe grape, a good friend, loyal to death;

SHOW GERMAN

SONATA-CONCERTO
Opening Ritornello
Exposition
Horror/Recitative
Primary Area (Joy) 1 2 3
Secondary Area (Turkish)
Development
Recapitulation
Coda

PLAY TRILL

INDEX ↑	MOVEMENT ↑	NOTES	10:06	Ⓞ PLAY THROUGH	PAUSE	FIND AGAIN
?	GLOSSARY ↑	10:04 - 10:10		○ PLAY THIS CARD		278

Figure 17 - A card corresponding to a leaf node in Fig. 16.

Each node has a “Text” perspective (the commentary on the piece of music), and a “CD Music” perspective (the actual played music). The user may play the music bit by bit, play it through (a sequential traversal of the hierarchy), or go to any of the major parts (the intermediate nodes in the hierarchy). Figure 17 shows an example card, in this case corresponding to a leaf node.

An interesting aspect to observe is the possible ways to visualize this entity. If one selects the “play this card” option at the bottom, the “CD Music” perspective of the current component is presented (i.e., the music is played). In the “play through” mode, all the leaves of the structure are presented in sequence, with the system automatically cueing the next component according to the duration of the “CD Music” perspective of each one. All of the above applies to both the “Text” and the “CD Music” perspectives, which are perceived contemporaneously. Note that this is a different browsing semantics than that exemplified in section 2.4.1.

To summarize the modelling of this hypertext application, it is quite straightforward to observe that it has a simple structure, with most navigation being either sequential, or short “deviations” from it (the musical examples), or else jumps into other points of the hierarchy. In HDM terms, this application can be entirely modelled with structural links only. The remaining features are nicely captured with the notion of perspectives, and the sharing of bodies (“CD Music”) among units.

4. Formal Definition of HDM

In this section we present a formalization of HDM concepts described so far. HDM primitives that model the more static aspects of hypertext applications are formalized in a relational-set theoretical style. Moreover, we show how the browsing semantics discussed in section 2.4.1 can be formalized using a simplification of the Trellis' Petri Nets based formalism [Stotts 89].

4. 1 Formalization of HDM Primitives

Def. 1 - Perspective and presentation

We assume the existence of a set P of perspective symbols (or, simply, perspectives). A perspective denotes a set of unformatted data (e.g., text, graphics, still images, digital sound recordings, etc...) which can be possibly characterized under a predefined linguistic or rethoric style, or even formatted information (e.g., tables of a database, spreadsheets, pieces of program code, etc...) which can be interpreted outside of HDM through an appropriate interpreter. Perspectives are neither interpreted nor further modelled within HDM (since they concern with "authoring in the small").

Examples of perspectives are: "Text-schematic-english", "Text-schematic-italian" (denoting a set of English/Italian documents described in a schematic style), "Text-Official-Version-English", "Text-Official-Version-Italian" (denoting a set of English/Italian official documents), "Graphics-geometric", "Graphic-artistic" (denoting a set of geometric/artistic graphic images), "Lotus-spreadsheet (denoting a set of Lotus-1-2-3 spreadsheets)".

If p is a perspective, we will write $v \in p$ to mean that v is a value belonging to the set of data denoted by p .

We call a set of perspectives a *presentation*.

Def. 2 - Unit

Given a perspective p , a *Unit* u under p denotes a value $b \in p$. b is named the *body* of that unit, denoted by $body(u)$. We use the notation $perspective(u)$ to refer to p .

For perspectives $p_1 = \text{"Text-Official-Version-English"}$ and $p_2 = \text{"Text-Official-Version-Italian"}$, example of units respectively under p_1 and p_2 are "circular 1723-i-sect. 1" and "circular 1723-e-sect. 1" denoting the official texts (respectively in Italian and English) of hyperbank circular 1723 described in fig. 6 of sect. 3.1.

Def. 3 - Component

A *Component* under a presentation P is a set of units $c = \{u_1, \dots, u_n\}$ such that

- 1) for all $i=1..n$, $body(u_i)$ belongs to some perspective in P ;
- 2) if $u_i \neq u_j$ then $perspective(u_i) \neq perspective(u_j)$.

In the following, we will use the notation c/p to denote the unit of c under the perspective p .

Note that \mathcal{C} does not partition the presentation, as there may be components that do not have units in some of the perspectives of the presentation.

For the example of units above, component “circular 1723-sect. 1” under the presentation $\{p_1, p_2\}$ is $\{u_1, u_2\}$, and the notation “circular 1723-sect. 1”/ p_1 denotes its unit u_1

Def. 4 - Entity

An *Entity* under a presentation P is an *ordered tree* of components under P .

Given an entity e , $\text{root}(e)$ denotes its root component.

An example of entity is “circular 1732”, which is made of components “introduction-c.1723” (the “root”) and the sequence of “sons” <“circular 1723-sect. 1”, “circular 1723-sect. 2”>, each one in turn can be made of a sequence of sub-sections, sub-sub-sections, etc.

Given a set of entities $\{e_1, \dots, e_n\}$, we define $\text{Comp}(\{e_1, \dots, e_n\})$ as the set of all components of the entities in $\{e_1, \dots, e_n\}$: $\text{Comp}(\{e_1, \dots, e_n\}) = \{c \mid c \in e, e \in \{e_1, \dots, e_n\}\}$.

We define $\text{Units}(\{e_1, \dots, e_n\})$ as the set of all units of the components of the entities in $\{e_1, \dots, e_n\}$: $\text{Units}(\{e_1, \dots, e_n\}) = \{u \mid u \in c, c \in e, e \in \{e_1, \dots, e_n\}\}$.

We extend this notation when the set contains only one entity e to be $\text{Comp}(e)$ and $\text{Units}(e)$ instead of $\text{Comp}(\{e\})$ and $\text{Units}(\{e\})$.

From the definition of an entity e , there are two basic binary relations induced over $\text{Comp}(e)$, *parent* and *next-sibling* such that, for $c, c' \in e$

parent (c, c') iff c' is the parent node of c in e ;

next-sibling (c, c') iff there exists $c'' \in \text{Comp}(e)$ such that *parent* (c, c'') and *parent* (c', c'') hold, and c' is the successor node of c in the order relation of e ;

For example, in Comp (“circular 1732”), *parent*(“circular1723-sect. 2”, “introduction-c.1723”) and *next-sibling*(“circular1723-sect. 1”, “circular 1723-sect. 2”) hold.

From the two basic relations, *parent* and *next-sibling*, many others can be defined, such as

first-son (c, c') iff $c' = \min\{c'' \mid \text{parent}(c'', c)\}$.

to-top (c, c') iff $c' = \min\{c'' \mid c'' \in e\}$.

For example, *first-son*(, “introduction-c.1723”, “circular1723-sect. 1”), and *to-top*(“introduction-c.1723”, “circular1723-sect. 1.2.3”).

We call the basic relations *parent* and *next-sibling*, and any relations derived from them *structural relations*.

Def. 5 - Entity Type

An *Entity Type* is a named set of entities sharing the same presentation ρ . Furthermore, there is a distinguished perspective in ρ called the *default* perspective of the type.

An *entity type specification* is then a triple $\langle n, \rho, p^* \rangle$, where n is the type name, ρ is the shared presentation (i.e., set of shared perspectives) and $p^* \in \rho$ is the default perspective. If $ET = \langle n, \rho, p^* \rangle$ is an entity type specification, we will write $e \in ET$ to mean that e is an entity belonging to the set denoted by n .

In the example of the Expert Dictionary (sect. 3.1), some entity types are "Laws" and "Procedures"; the specification for "Law" is $\langle \text{"Law"}, \{ \text{"Structure"}, \text{"Official Text"} \}, \text{"Structure"} \rangle$.

Def. 6 - Links

Let \mathcal{S} and \mathcal{T} be two arbitrary sets, and \mathcal{L} denote a binary relation over $\mathcal{S} \times \mathcal{T}$. Then a *link* over $\mathcal{S} \times \mathcal{T}$ of type $\mathcal{L}_{\mathcal{S}\mathcal{T}}$ is the triple $\langle \mathcal{L}, s, t \rangle$, where $s \in \mathcal{S}$, $t \in \mathcal{T}$ and the relation denoted by $\mathcal{L}(s, t)$ is true. For simplicity, when the type of a link is unambiguous, we will denote it simply by \mathcal{L} instead of $\mathcal{L}_{\mathcal{S}\mathcal{T}}$.

If $l = \langle \mathcal{L}, s, t \rangle$ is a link, we say that s and t are, respectively, the *source* and *destination* of link l , denoted by $source(l)$ and $target(l)$.

The triple $\langle \mathcal{L}, \mathcal{S}, \mathcal{T} \rangle$ is called *basic link type specification*.

Def.7 - Perspective Links

Let T be an entity type, and let $\langle T, P, p^* \rangle$ be its specification, where $P = \{p_1, \dots, p_n\}$

P induces $n(n-1)$ relations, denoted by $p_i:p_j$ ($p_i \neq p_j$), over $Units(T) \times Units(T)$, defined by

$p_i:p_j(u_1, u_2)$ is true iff $u_1 = \langle p_i, b_1 \rangle$, $u_2 = \langle p_j, b_2 \rangle$, and u_1, u_2 belong to the *same* component.

A *Perspective Link* of type $p_i:p_j$ is a link defined over $Units(T) \times Units(T)$ of type $p_i:p_j$. Perspective links connect units of the same component, but of different perspectives.

For perspectives $p_1 = \text{"Text-Official-Version-English"}$ and $p_2 = \text{"Text-Official-Version-Italian"}$, a perspective link of type $p_1:p_2$ is $\langle p_1:p_2, \text{"circular 1723-e-sect. 1"}, \text{"circular 1723-i-sect. 1"} \rangle$.

Def. 8 - Structural Links

If T is an entity type, a link $l = \langle sr, c_1, c_2 \rangle$ over $Comp(T) \times Comp(T)$ is a *structural link* of type sr iff

a) there is an entity e in \mathcal{E} such that c_1 and c_2 belong to $Comp(e)$;

b) sr denotes a structural relation over $Comp(e)$;

Note that structural links are directly induced by the structural relations in the definition of entity. In other words, given an entity definition, the structural links can be automatically derived from it.

$\langle parent, "circular1723-sect. 1", "circular1723-sect. 1.1.2" \rangle$, is an example of structural link of type "parent", and $\langle to-top, "introduction-c.1723", "circular1723-sect. 1.1.2" \rangle$ is an example of structural link of type "to-top".

Def. 9- Application Links

If \mathcal{T} is an entity type, let $D(\mathcal{T})$ be the set of all entities of type \mathcal{T} , and of all their components : $D(\mathcal{T}) = Comp(\mathcal{T}) \cup \mathcal{T}$.

Let \mathcal{T}_1 and \mathcal{T}_2 be two entity types, and let AR denote an arbitrary relation over $D(\mathcal{T}_1) \times D(\mathcal{T}_2)$. An *application link* is simply a link of type AR .

Differently from structural links, the relations corresponding to application links are *not* derived from the structural relations. Moreover, application links can be set between components, entities, or both.

As an example, let "justified-by" denote a relation between entities or components of types "Document" and "Law", such that $justified-by(doc, law)$ is true if the contents of document denoted by doc are justified by the text of the law denoted by law . Then $\langle justified-by, doc, law \rangle$ is a conceptual link over $D(Document) \times D(Law)$ of type "justified-by". The basic link type specification for this type is $\langle justified-by, Document, Law \rangle$

Def. 10- Derived Links

Whenever a relation can be specified *intentionally* by the use of derivation rules, links corresponding to values generated by the such rules are called *derived link*.

From the definition of entity, all derived structural links can be defined in terms of *parent* and *next-sibling*. From properties of application links (e.g. symmetry, transitivity) and from the composition of relations, it is possible to introduce derivation rules for defining other application links. However, we do not include here a language for the specification of such rules.

A *link type specification* is a pair $\langle basic\ link\ type\ specification, derivation\ rules \rangle$. The derivation rules can be specified in any suitable formalism.

As an example, referring to fig. 1 in section 2.3.7, an application link is $\langle justified-by, sect. 1.1, article 2 \rangle$, and a derived link is $\langle justified-by, contract X, Law Y \rangle$, where *contract X*, and *Law Y* are the roots of the entities to which *sect. 1.1* and *article 2* belong. The link type specification for the derived "justified-by" link is $\langle justified-by, Document, Law, "Rule" \rangle$, where "Rule" is any formalization of the rule

"if $\langle justified-by, a, b \rangle$ holds and a and b are components of entities e_1 and e_2 respectively, then $\langle justified-by, root(e_1), root(e_2) \rangle$ also holds,".

Def. 11- HDM Schema

An *HDM Schema* is a 5-tuple

$\langle \mathcal{P}, \mathcal{ET}, \mathcal{SR}, \mathcal{ALT}, \mathcal{PL} \rangle$

where:

\mathcal{P} is a set of perspectives ;

\mathcal{ET} is a set of entity type specifications over \mathcal{P} ;

\mathcal{PL} is a set of perspective link type specifications, for each entity type in \mathcal{ET} ;

\mathcal{SR} is a set of structural link type specifications, for each entity type in \mathcal{ET} ;

\mathcal{ALT} is a set of application link type specifications over \mathcal{BT} .

\mathcal{PL} is a set of perspective link type specifications, for each entity type in \mathcal{ET} .

Note that it is necessary to include structural link type specifications in \mathcal{SR} only for entity types that have derived structural relations, since *parent* and *next-sibling* can be directly derived from the entity definition and are included by default.

Def. 12 - HDM Schema Instance

An *instance* of an HDM schema $\mathcal{S} = \langle \mathcal{P}, \mathcal{ET}, \mathcal{SR}, \mathcal{ALT} \rangle$ is a 4-tuple

$\langle \mathcal{E}, \mathcal{SL}, \mathcal{AL}, \mathcal{PL} \rangle$

where:

\mathcal{E} is a set of entities of type in \mathcal{ET} , such that

$\forall e_1, e_2 \in \mathcal{E}, e_1 \neq e_2 \Rightarrow e_1 \cap e_2 = \emptyset$ (no components are shared among entities);

$\forall c_1, c_2 \in \bigcup_{e \in \mathcal{E}} e, c_1 \neq c_2 \Rightarrow c_1 \cap c_2 = \emptyset$ (no units are shared among components);

\mathcal{SL} is a set of structural links of type in \mathcal{SR} such that for each l in \mathcal{SL} , *source*(l) and *target*(l) belong to *Comp*(\mathcal{E})

\mathcal{AL} is a set of application links of type in \mathcal{ALT} such that for each l in \mathcal{AL} , $source(l)$ and $target(l)$ belong to $Comp(\mathbf{E}) \cup \mathbf{E}$

\mathcal{PL} is a set of perspective links of type in \mathcal{PL} such that for each l in \mathcal{PL} , $source(l)$ and $target(l)$ belong to $Units(\mathbf{E})$;

Note that from the definition above, the only type of sharing allowed is of *bodies* between units.

Def. 13 - Outline

Let $\langle \mathbf{E}, \mathcal{SL}, \mathcal{AL}, \mathcal{PL} \rangle$ be a schema instance. An entity e in \mathbf{E} is called *outline* if

- 1) for each leaf component c of e there is at least one applicative link $\langle l, s, t \rangle$ such that $s=c$ and t does not belong to e ;
- 2) no components but the leaves are sources of applicative links.

Def. 14 - Minimal Schema Instance

An instance $\langle \mathbf{E}, \mathcal{SL}, \mathcal{AL}, \mathcal{PL} \rangle$ of a schema is *minimal* if \mathcal{SL} contains only links of type *parent* and *next*, \mathcal{AL} contains no derived links of any sort, and \mathcal{PL} is empty.

The minimal schema instance is actually the only thing that must be explicitly given by the author; all the other links can be generated from it and from the schema definition once a computational mechanism is available for derivation rules and for perspective links.

4.2. Formal Definition of the Default Browsing Semantics

Let $\mathcal{I} = \langle \mathbf{E}, \mathcal{SL}, \mathcal{AL}, \mathcal{PL} \rangle$ be an instance of a schema $\mathcal{SC} = \{\mathcal{P}, \mathcal{ET}, \mathcal{SR}, \mathcal{ALT}\}$.

The class of browsing semantics discussed in section 2.4.1 can be defined as a 7-tuple

$\langle ep, \mathcal{U}, \mathcal{C}, \mathcal{A}, \mathcal{Ch}, \mathcal{A}^*, \mathcal{S} \rangle$.

where:

- ep is an outline entity in \mathbf{E} , denoting the selected “entry point” of the instance. Note that the choice of different entry points can make accessible different navigation paths, which can be perceived as different “applications” by the reader.
- \mathcal{U} specifies what objects, among entities, components, and units of the instance, are perceived and human-consumable. In our case this is simply the set of units of the entities in the instance; therefore $\mathcal{U} = Units(\mathbf{E})$;
- \mathcal{C} is a set of “concrete connections”. Most HDM links are between entities and components and cannot be perceived by the users in this form, since units are the

only perceivable objects. It is therefore necessary to translate such “abstract” links into connections between units.

If $\mathcal{L} = \mathcal{SL} \cup \mathcal{AL}$ is the union of all structural and application links in the instance \mathcal{U} , the translation is performed by a function $Conn: \mathcal{L} \rightarrow 2^{\mathcal{L}\mathcal{T} \times \mathcal{U} \times \mathcal{U}}$. Then $\mathcal{C} = Conn(\mathcal{L}) \cup \mathcal{PL}$ (perspective links are already between units).

Different definitions of $Conn$ produce different navigation paths. Our definition formalizes the rules discussed in sect. 2.4.:

- If c is an entity or a component, let $[[c]]$ denote the representative of c defined as:

$[[c]] = u/p^*$, if c is a component and p^* is its default perspective

$[[c]] = [[root(c)]]$, if c is an entity

As discussed in 2.4.1, the representative of a component is its unit under the default perspective, and the representative of an entity is the representative of its root component.

- if $lt = \langle lt, c_1, c_2 \rangle$ is an application link type, and c_1 is a component, then $Conn(\langle lt, c_1, c_2 \rangle) = \{ \langle lt, x_1, x_2 \rangle \mid x_2 = [[c_2]] \text{ and } x_1 \in Units(c_1) \}$; an applicative link having a component as source is translated into concrete connections going from each unit of the source component to the representative of the target;
- if $lt = \langle lt, c_1, c_2 \rangle$ is an application link type, and c_1 is an entity, then $Conn(\langle lt, c_1, c_2 \rangle) = \{ \langle lt, [[c_1]], [[c_2]] \rangle \}$; an applicative links having an entity as source is translated into a concrete connection going from the representative of the source entity to the representative of the target.
- if lt is a structural link type, then $Conn(\langle lt, c_1, c_2 \rangle) = \{ \langle lt, x_1, x_2 \rangle \mid x_1 \in Units(c_1), x_2 \in Units(c_2), perspective(x_1) = perspective(x_2) \}$; a structural link is translated into concrete connections between units of the same perspective of the corresponding source and target components.

Moreover, if $cc = \langle lt, x_1, x_2 \rangle$ is a concrete connection, we will say that the type of cc is lt .

Once concrete connections are defined, they are perceived through anchors. Anchoring aspects are modelled by \mathcal{A} , \mathcal{Ch} and \mathcal{A}^* .

- \mathcal{A} is a function which maps link types into the corresponding anchor types. We assume the existence of an anchor universe \mathcal{AU} , partitioned into anchor types \mathcal{AT} , and including one distinct type \mathcal{CAT} called “chooser-anchor”.

$$\mathcal{A}: \mathcal{LT} \rightarrow \mathcal{AT} \setminus \mathcal{CAT} \cup \{\text{nil}\}.$$

\mathcal{A} defines an equivalence relation over \mathcal{C} , defined as

$$\begin{aligned} \text{for all } \langle l_1, c_1, c_2 \rangle, \langle l_2, d_1, d_2 \rangle \in \mathcal{C}, \\ \langle l_1, c_1, c_2 \rangle \approx \langle l_2, d_1, d_2 \rangle \text{ iff } \mathcal{A}(l_1) = \mathcal{A}(l_2) \text{ and } c_1 = d_1. \end{aligned}$$

Therefore two connections are equivalent iff they have the same source and their type is mapped by \mathcal{A} into the same anchor type (or nil).

In our browsing semantics, an anchor can group several connections. As we will see later, the equivalence above is used to map the connections of a given type, with the same source, to a single anchor of a given type. In particular, if $\mathcal{A}(l_1) = \mathcal{A}(l_2)$ then all connections of types l_1 and l_2 from the same source will be perceived through the same anchor (of type $\mathcal{A}(l_1)$). If $\mathcal{A}(l) = \text{nil}$, then the connections of this type are not perceivable, and are called *null connections*.

An equivalence class that does not contain null connections is called an *e-connection* (for extended connections). Let \mathcal{C}^* be the set of extended connections. We define the function target^* on \mathcal{C}^* as $\text{target}^*(c) = \{ t \mid \exists l \in c, \text{target}(l) = t \}$.

- \mathcal{Ch} is a set of choosers. We assume the existence of a universe \mathcal{CU} of choosers (elements distinct from all elements defined so far) which are used to represent the targets of multiple destination anchors.

\mathcal{Ch} is a subset of \mathcal{CU} which contains one distinct element for each $c \in \mathcal{C}$ such that $\|\text{target}^*(c)\| > 1$; we will write $\text{chooser}(c)$ to refer to it.

- \mathcal{A}^* is a set of anchor structures, i.e., the final conceptual representation of connections. An anchor structure has the form $\langle a, x, y \rangle$, where $a \in \mathcal{AU}$ and $x, y \in \mathcal{U} \cup \mathcal{Ch}$. \mathcal{A}^* is formed from \mathcal{C}^* and \mathcal{Ch} by the rules

- 1) For each e-connection c in \mathcal{C}^* such that $\|\text{target}^*(c)\| = 1$, if $c = \{ \langle l, c_1, c_2 \rangle \}$, then the corresponding anchor structure is $\langle a, c_1, c_2 \rangle$.

2) For each e-connection c in C^* such that $\|target^*(c)\| > 1$, if $c = \{ \langle t, c_i, c_j \rangle, i \neq j, i, j = 1..n, n > 1 \}$ and ch is $chooser(c)$ then the corresponding anchor structures are

i) $\langle a, c_i, ch \rangle$;

ii) $\{ \langle a_1, ch, x_1 \rangle, \dots, \langle a_n, ch, x_n \rangle \}$, where a_1, \dots, a_n are distinct anchors of type "chooser-anchor" and x_1, \dots, x_n are all the targets of all links c_j in c .

It can be proved that all anchors generated in 1) and 2) above are distinct among themselves within the same node.

- **S** (simplified Trellis model) is a modified version of the Trellis model [Stott 89]. **S** formalizes the semantics of link activation for the network resulting from the previous steps.

We have (re)used here the idea of representing navigational path through a Petri Net, mapping perceivable objects into places, and links into transitions. The execution semantics of Petri Nets provides the model of browsing the hypertext. The bodies of units map into "document contents" (in the Trellis model sense) of places, and anchors map into buttons (a-la Trellis); link traversal is then equivalent to transition firing in the corresponding Petri-Net.

We have taken out of Trellis all aspects concerning authoring-in-the-small (namely, "logical windows" and "display projection"), as this is not a concern for HDM.

S is a 6-tuple $\langle \mathcal{N}, \mathcal{Ct}, \mathcal{Bt}, \mathcal{Cl}, \mathcal{Bl}, \mathcal{M}_0 \rangle$ defined as follows:

$\mathcal{N} = \langle \mathcal{Pl}, \mathcal{T}, \mathcal{F} \rangle$ is a Petri Net, where \mathcal{Pl} is the set of places; \mathcal{T} is the set of transitions, and \mathcal{F} the set of flow relations.

\mathcal{Pl} and \mathcal{T} are isomorphic to $\mathcal{U} \cup \mathcal{Ch}$ and \mathcal{B}^* respectively. Let $f_1 : \mathcal{U} \cup \mathcal{Ch} \rightarrow \mathcal{Pl}$ and $f_2 : \mathcal{B}^* \rightarrow \mathcal{T}$; be the two functions that perform such isomorphism. \mathcal{F} is defined as $\mathcal{F} = \{ \langle f_1(x), t \rangle, \langle t, f_1(y) \rangle \mid t \in \mathcal{T}, t = f_2(\langle b, x, y \rangle) \}$.

\mathcal{Ct} is the set of Trellis "document contents", $\mathcal{Ct} = \bigcup_{u \in \mathcal{U}} body(u)$, which are the bodies of the units.

\mathcal{Bt} is the sets of buttons, defined as $\mathcal{Bt} = \{ b \mid \exists b^* \in \mathcal{B}^*, b^* = \langle b, x, y \rangle \}$; that is, the set of buttons is exactly the set of anchors in the anchor structures of \mathcal{A}^* ;

\mathcal{CL} is a function mapping of places into contents, $\mathcal{CL} : \mathcal{PL} \rightarrow \mathcal{Ct} \cup \{\text{nil}\}$, such that $\mathcal{CL}(p) = \text{nil}$ if $p = f_1(\text{ch})$, $\text{ch} \in \mathcal{Ch}$, and $\mathcal{CL}(p) = \text{body}(u)$ if $p = f_1(u)$, $u \in \mathcal{U}$. Since choosers do not have bodies as such, as they are merely containers for links, places corresponding to choosers have no contents; all others have as their contents the bodies of the corresponding units.

\mathcal{BL} is the mapping of transitions into buttons, $\mathcal{BL} : \mathcal{T} \rightarrow \mathcal{Bt}$ such that $\mathcal{BL}(t) = b$, if $t = f_2(\langle b, x, y \rangle)$. The effects of clicking on a button correspond to firing the corresponding transition in the net.

\mathcal{M}_0 is the initial marking of \mathcal{N} such that $\mathcal{M}_0(s) = 1$ iff $s = f_1(\text{ep})$, where ep is the entry point of the instance, and $\mathcal{M}_0(s) = 0$ otherwise. This definition of initial marking is consistent with the intended semantics of the entry point since it leaves only the transition corresponding to the entry point initially enabled.

It can be trivially proved that in a hypertext modelled by the current HDM browsing semantics the following property holds:

“there is only one “active”, i.e., perceivable, node (unit or chooser) at any time and each button takes exactly to one node”.

It is easy to see that the Petri Net generated in the model given here is a Condition/Event Net, that is: the weight of all arcs is one, all places have at most one token, and therefore a transition is enabled iff the places in its pre-set have exactly one token and all places in its postset are empty. Moreover, the pre-set and the postset of each transition contains exactly one place. Therefore, there is only one marked place at any time, and the only enabled transitions leave from the place corresponding to it.

Finally, we feel that the formal Petri net based approach seems promising for specifying even more complex navigation behaviours than the one presented here.

5. Conclusions

5.1 Discussion

A model was presented in the previous sections allows the specification of an hypertext at the “authoring-in-the-large” level, focusing on the structure of the hypertext rather than nodes’ contents. The advantages of having such a model were discussed; even with the present simple formulation of the browsing semantics, it is possible to see the power of HDM authoring-in-the-large approach, providing conceptual and formal tools to design, analyze, and verify structural and access properties of a given hypertext at a relatively high level of abstraction. Examples of its use, and a formal specification allowing a precise definition of its semantic were also given.

HDM provides at least two dimensions of indirection. The first dimension is static, provided by the notion of schema (global structure) and schema instance, and by the separation between contents and structure, given by the notion of bodies for units (the latter as in Trellis). The second dimension is dynamic, provided by the separation between the static and runtime aspects, given by the notions of schema instance and its association with some browsing semantics.

By encouraging viewing the hyperbase at a higher level of abstraction than the “lower” level of nodes and links, HDM can be regarded as a “logical” hypertext model, in database terms. On the other hand, by referring to HDM data primitives as “conceptual” objects of the application domain, and by allowing the derivation of links through rules (which necessarily capture some application domain semantics), HDM may also be regarded as a preliminary attempt towards “semantic hypertext modelling”.

HDM shares some apparent similarities with the Entity Relationship (ER) model [Chen 76]. However, there is no ER notion equivalent to HDM perspectives and HDM Entities are much more structured than ER entities, which do not have structural links. Most importantly, whereas in the ER model relations are included for representational reasons, HDM links are included also with the goal of providing *navigational* paths. In fact, in those cases where entities do not have any internal structure, the resulting HDM model will be very similar to an ER model, if one ignores momentarily HDM perspectives; HDM in this sense can be said to be higher level than ER, allowing for more concise specifications of hypertext applications.

HDM differs from IDE and g-IBIS in the fact that it does not fix, a priori, the application domain, and therefore its primitives are more “general”, oriented towards allowing the specification of models in most application domains.

Object Lens classes resemble somewhat HDM entity types. However, there are no constraints imposed on possible connections between object class instances; even if one interprets class fields as link types, they do not impose any real discipline on actual instances. For example, one may have a “Supervisor” field for object class “Person”, and have an instance in which the value of this field is (an instance of) a “Vehicle”.

With respect to other models discussed in section 1.2, HDM differs in the fact that it is aimed at modelling applications rather than systems. Therefore, it is fair to say that its primitives are at a “higher” level of abstraction than the node-and-links level of the other models.

Even though the Trellis model completely abstracts from the contents and structure of the “nodes”, it allows its contents to be arbitrarily complex information structures (including an entire hypertext). Petri-Nets used as in the Trellis model is a simple and elegant formalism in which to formalize browsing semantics. A similar observation can be made with respect to Tompa’s model.

5.2 Future Work

The model presented in this article is still under development; we have described only its first stable version. There are a number of extensions being examined, which we mention briefly.

5.2.1 Constraints

A useful extension to link definition is to include *constraints* in the schema definition. A simple type of constraint is on the cardinality of outgoing or incoming links, for given entity types. A straightforward way to specify these constraints is through an integer, say n (or $n+$) associated with the link type specification, with the intended meaning that in schema instances that link will have exactly n instances (at least n). Note that when $n = 1$ this is simply referential integrity. Design tools can take advantage of these constraints to either enforce consistent hypertext creation, or to check existing designs, or to incorporate them in structure accelerators in a spirit similar to IDE.

A second class of constraint is entailed by the notion of “well-known” entity type *instances*, so that one may state a constraint which specifies that *all* instances of a given type

must be linked to one (or more) well-known instance of another type. A typical example of a “well-known” instance is one containing “help” information about the entity type. This type of constraint actually allows the specification of class links, since they describe links for all elements of a given type at the global, schema level.

5.2.2 Structure Patterns for Entity Types

The current HDM definition says nothing about the internal structure of entities at schema level; only entity types are specified. However, it is useful to be able to specify, still at the schema level, some overall structural pattern that entity type instance must follow. For example, it might be useful to say that “Law” entity type instances all have a common format, for example, that the first component sub-tree is always of type “Introduction”, followed by a variable number of sub-trees, of type “Section”. Being able to allow such distinctions permits for example automatic generation of navigation links such as “link first the Introduction, then each Section”. This mechanism can also be used in a very powerful manner to construct structure accelerators, in the Intermedia template and in the IDE structure accelerator styles.

5.2.3 Links as First Class Objects

Links in HDM are, in a way, “second-class” objects, as they do not contain any information besides their types. A natural extension is the possibility of adding information to links, turning them into “first class” objects (as in [Sherman 90]). Semantically speaking, this means the association of information to *relationships* between entities or components. It should be noticed that this can already be done in HDM as it stands, but requires the creation of a fictitious entity to stand for the relationship, much in the same way as is done in the relational model for databases.

5.2.4 Entity Attributes and Properties

A frequently occurring situation is the one exemplified by the “Musical Example” entity type in the “CD Companion to Beethoven’s 9th” example. The entities and components of type “Musical Example” can be regarded really as “attributes” (in the sense of the Entity-Relation model) of the types they are linked to (e.g. “Historical/Musical Notes”). Said differently, it is often the case where one finds an entity type related to many “attribute” types, the relation simply representing the fact that one is an attribute of the other. The entities of attribute types do not have an existence of their own, but must be accessed always through the instances of the entity type they are an attribute of. This situation corresponds to having an entity whose components are no longer all homogenous with respect to a unique entity type; some components are of “attribute type”. Nevertheless, the semantics of structural links can be extended consistently to allow navigation across the corresponding “attribute” links between the entity and its attributes.

Another useful notion is the possibility of adding “properties” (i.e. formatted meta-information) to entities (components, units). These properties may be used by specialized processors for tasks such as maintenance, automatic document generation, query based retrieval, etc...

5.2.5 Overlays for Navigation

The Outline usage of Entities presented here is clearly limited, as it concentrates more on providing initial access to points of the hyperdocument. We intend to generalize this notion with the concept of *Overlays*. An Overlay is a set of links which is superimposed onto the “stable” schema instance to represent particular navigation patterns which are dependent on specific user classes (profiles) and tasks. As a mechanism, it is more flexible than the currently defined Outlines, which must still be structured as Entities.

5.2.6 Dynamic Primitives

The discussion on browsing semantics indicated that it is necessary to be able to model runtime behaviour properties of hypertext applications. We have shown how Petri-Net based models can be used to formalize the run-time behaviour of an application specified in HDM; the browsing semantics described so far is fairly simple, even though it is the only feasible, or at least natural, in many popular hypertext implementation environments such as Hypercard (versions previous to 2.0) and Guide.

HDM does not yet include primitives for specifying browsing behaviour of applications both at the schema and at the schema instance levels. In fact, work is continuing in this direction, including defining primitives for specifying browsing semantics, visualization aspects, and more generally, operational behaviour of applications.

In particular, HDM should be extended to model true multimedia hyperdocuments, those that allow *interaction* with multimedia; these extensions raise several new modelling requirements, both from static and from dynamic points of view.

5.2.7 Preferred Navigation Patterns

Once a schema is available, it is possible to foresee a number of navigational patterns allowed by the available link types. Very seldom, however, it is written anywhere what are the intended patterns, among all possible ones, that the hypertext author had in mind. HDM schemas could be enriched by a *preferred navigation pattern*, which is intended, at first instance, to be simply a requirement specification that the author states at a very early stage of the design. Once the schema has been completed, it is possible to check it against the preferred paths specifications and see whether the final design allows them to be easily traversed. A second usage of these patterns is at run-time, when a special "compilation" mode could generate a "driver" that "accompanies", upon request, the reader along paths that are instances of such navigation patterns.

5.2.8 Authoring Operations

In the discussion so far, nothing was said about authoring operations. Issues such as interface design, navigation environments, traces, and authoring tools are being examined as part of a large European research project on hypertext design tools, called HYTEA [ARG 90]. This project includes the design and implementation of development tools that support HDM-based design, and are capable of "compiling" such specifications into existing hypertext systems. As a matter of fact, a small implementation prototype of a compiler into HyperCard has been developed; its main ideas are described in [Schwabe 90a].

It should be noted that authoring environment entails investigation on integrating HDM with authoring-in-the-small methodologies and environments. Many of the design choices made at the authoring-in-the-large level have consequences at the authoring-in-the-small level - anchors, windows, etc...

Finally, further into the future, we are planning on investigating the extension (or integration) of HDM which allows representing deeper semantic models (e.g., as in [Schwabe 90b]) which capture more domain knowledge.

Acknowledgement - Many of the ideas described here benefited from discussions with Mark Bernstein of Eastgate Systems, Inc, with Norman Meyrowitz of IRIS, Brown University, and also with John Mylopoulos, Computer Science Dept., University of Toronto. As part of the group that developed HDM, we also acknowledge the contributions of Andrea Caloini, Stefano Mainetti, and Sergio Danieluzzi.

5. References

- [ARG 90] ARG, "HYTEA Technical Annex", Esprit Project P5252, June 1990
- [Aksyn 87] Aksyn, R.; McCracken, D.; Yoder, E.; "KMS: A Distributed Hypertext System for Managing Knowledge in Organizations", Proc. Hypertext '87, ACM, Baltimore, 1987, pp. 1-20
- [Atkinson 87] Atkinson, W.; HyperCard, software for Macintosh computers, Cupertino, Apple Computer Co, 1987.
- [Bernstein 88] Bernstein, M., "The Bookmark and the Compass: Orientation Tools for Hypertext Users", SIGOIS Bulletin 9 (1988) pp. 34-45
- [Bernstein 89] Bernstein M. and Sweeney, E. , The Election of 1912, hypertext for Macintosh computers, Eastgate Systems Inc, Watertown MA, 1989
- [Bernstein 90] Bernstein, M.; "The Navigation Problem Reconsidered", chapter VI, "Hypertext/Hypertext Handbook", Berk, E.; Devlin, J. (Eds.), McGraw Hill, 1990.
- [Bernstein 90a] Bernstein, M.; "An apprentice that discovers hypertext links", in "Hypertexts: Concepts, Systems and Applications (Proceedings of ECHT'90)", A. Rizk et al., eds., Cambridge Series on Electronic Publishing, Cambridge University Press, Cambridge, 1990
- [Bolter 87] Bolter, J.D.; Joyce, M.; "Hypertext and Creative Writing", Proc. Hypertext '87, ACM, Baltimore, 1987. pp.41-50
- [Bolter 90] Bolter, J.D.; Writing Space: The Computer, Hypertext, and the History of Writing, Lawrence Erlbaum and Associates, San Mateo. in press.
- [Brodie 84] Brodie, M.; Mylopoulos, J.; Schmidt, J. (eds.), "On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases and Programming Languages", Springer Verlag, 1984
- [Brown 87] Brown, P.J.; "Turning Ideas Into Products: The Guide System", Proc. Hypertext '87, ACM, Baltimore, 1987. pp. 33-40
- [Brown 89] Brown, P.J.; "Do we need maps to navigate round hypertext documents?", Electronic Publishing-Origination, Dissemination and Design 2, 2 (July 89).
- [Chen 76] Chen, P.P.S. "The entity-relationship approach: toward a unified view of data", ACM Transactions on Data Base Systems 1(1), 1976.
- [Clitherow 89] Clitherow, P.; Riecken, D.; Muller, M.; "VISAR: A system for inference and navigation of hypertext", Proc. Hypertext '89, ACM, Baltimore, 1989. pp. 293-305
- [Conklin 88] Conklin, J.; Begeman, M. L.; "gIBIS: A Hypertext Tool for Exploratory Policy Discussion", ACM Trans. Office Information Systems 6 (1988) 303-331
- [Coombs 87] Coombs, _; Renear, _; DeRose, S.; "Types of markup and the future of scholarly text processing", Communications of the ASCM 11 (1987)
- [Crane 87] Crane, G.; "From the old to the new: Integrating hypertext into traditional scholarship", roc. Hypertext '87, ACM, Baltimore, 1987. pp. 51-5
- [Crane 90] Crane G.; "Standards for a Hypertext Database: Diacronic v.s. Synchronic Concerns", Proc. 1st Hypertext Standardization NIST Workshop, Gaithersburg, MD, Jan. 1990
- [De Young 89] De Young, L.; "Hypertext challenges in the auditing domain", Proceedings of Hypertext'89, Pittsburgh, November 1989.
- [diSessa 86] diSessa, A; Abelson, H.; "Boxer: A Reconstructible Computational Medium", Communications of the ACM 29 (1986)
- [Engelbart 63] Engelbart, D.; "A Conceptual Framework for the Augmentation of Man's Intellect", Computer-Supported Cooperative Work: A Book of Readings,

- Irene Greif, ed., Morgan Kaufmann Publishers Inc., San Mateo. 1988. pp. 35-66
- [Engelbart 68] Engelbart, D.; English, W.; "A Research Center for Augmenting Human Intellect", Computer-Supported Cooperative Work: A Book of Readings, Irene Greif, ed., Morgan Kaufmann Publishers Inc., San Mateo. 1988. pp. 81-106
- [Fairchild 88] Fairchild, K.M.; Poltrock, S.E.; Furnas, G.W.; "SemNet: Three-dimensional graphic representations of large knowledge bases", Cognitive Science and its Applications for Human-Computer Interaction, R. Guindon, ed., Lawrence Erlbaum, 1988.
- [Furuta 90] Furuta R., Stotts D., "The Trellis Reference Model", Proc. 1st Hypertext Standardization NIST Workshop, Gaithersburg, MD, Jan. 1990
- [Garg 88] Garg P.K., "Abstraction mechanisms in Hypertext", Comm.ACM, vol. 31, July 88
- [Garzotto 89] Garzotto F., Paolini P., "Expert Dictionaries: Knowledge Based Tools for Explanation and Maintenance of Complex Application Environments", Proc. 2nd ACM Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Tullahoma, TN., Aug. 1989
- [Garzotto 90] Garzotto F., Schwabe, D.; Paolini P.; Caloini, A. Mainetti, S.; Borroni, S., "HDM - HYPERTEXT DESIGN MODEL", Tech. Report. m 90-41, Dept. of Electronics, Politecnico di Milano, Oct. 1990
- [Goldberg 80] Goldberg, A.; Robson, D.; Smalltalk-80: The Language and its Implementation, Reading, Addison-Wesley. 1983
- [Glushko 89] Glushko, R.; "Design issues for multi-document hypertexts", Proc. Hypertext '89, ACM, Baltimore, 1989. pp. 51-60.
- [Halasz 87] Halasz, F.; "Reflections on NoteCards: Seven Issues for the Next Generation of Hypertext Systems", Proc. Hypertext '87, ACM, Baltimore, 1987. pp. 345-66
- [Halasz 87a] Halasz, F.; Moran, T.P.; Trigg, R.H.; "NoteCards in a Nutshell", Proc. ACM CHI+GI 87 (Toronto, 5-9 April 1987) 45-52
- [Halasz 90] Halasz F., Schwartz M., "The Dexter Reference Model", Proc. 1st Hypertext Standardization NIST Workshop, Gaithersburg, MD, Jan. 1990
- [Hartson 89] Rex Hartson H.; Hix Deborah, Human-Computer Interface Development: Concepts and Systems for Its Management, ACM Computing Surveys 21, 1989
- [Hayes 88] Hayes, P.J.; Knecht, L.E.; Cellio, M.J.; "A News Story Categorization System", Proc. 2nd Conf. on Applied Natural Language Processing, Austin, Texas, February 1988. pp. 9-17
- [Hull 87] Hull, P.; King, R. "Semantic Database Modelling: Survey, Application and Research Issues", ACM Computing Surveys, Sept. 1987
- [Jordan 89] Jordan, D.; Russel, D. "Facilitating the Development of Representations in Hypertext with IDE", Proc. Hypertext '89, ACM, Baltimore, 1989
- [Kapor 90] Kapor, M. et al; "Agenda", Communications of the ACM 33 (1990)
- [Lay 88] Lay, K-Y; Malone, T.W.; Yu, K-C; "Object Lens: A Spreadsheet for Cooperative Work", Transactions on Office Information Systems, Vol. 6 (4), Oct. 88, pp 332-353.
- [Lenat 90] Lenat, D. B.; Guha, R.V.; Building Large Knowledge-Based Systems: Representation and Inference in the CYC Project, Reading, Addison-Wesley, 1990.

- [Lundh 89] Lundh, J.; "HYBRIS - A first step towards efficient information resource management", Technical Report, Swedish Institute for Systems Development, Kista, Sweden, 1989.
- [Malone 87] Malone, T.W.; Grant, K.R.; Lai, K-Y.; Rao, R.; Rosenblitt, D.; "Semistructured Messages are Surprisingly Useful", Computer-Supported Cooperative Work: A Book of Readings, Irene Greif, ed., Morgan Kaufmann Publishers Inc., San Mateo. 1988. pp. 311-34
- [Margolis 89] Margolis, M.; review of "The Election of 1912", Social Science Computer Review 7 (1989) 231-3.
- [Marshall 89] Marshall, C.C.; Irish, P.M., "Guided Tours and On-line Presentations: How Authors Make Existing Hypertext Intelligible for Readers", Proc. Hypertext '89, ACM, Baltimore, 1989
- [Nelson 76] Nelson, T.H.; Computer Lib/Dream Machines, reprinted by Microsoft Press, Redmond WA 1988.
- [Nelson 81] Nelson, T. H.; Literary Machines, privately published, 1981.
- [Nielsen 90] Nielsen, J; "The Art of Navigating Through Hypertext", Communications of the ACM 33 (1990)
- [Parunak 89] Parunak, H. V. D.; "Hypertext Topologies and User Navigation", Proc. Hypertext '87, ACM, Baltimore, 1987. pp. 43-50
- [Richard 90] Richard G., Rizk A., "Quelques idées pour une modélisation des systèmes hypertextes", to appear in Techniques et Sciences Informatiques.
- [Schwabe 90a] Schwabe, D.; Caloini, A.; Garzotto, F.; Paolini, P., "Hypertext development using a model-based approach", Technical Report 90-74, Dipartimento di Elettronica, Politecnico di Milano, Nov. 90. Submitted for publication.
- [Schwabe 90b] Schwabe, D.; Feijó, B; Krause, W.G., "Intelligent Hypertext for Normative Knowledge in Engineering", in "Hypertexts: Concepts, Systems and Applications (Proceedings of ECHT'90)", A. Rizk et al., eds., Cambridge Series on Electronic Publishing, Cambridge University Press, Cambridge, 1990
- [Sherman 90] Sherman, M.; Hansen W.J.; McInerney, M.; Neundorfer, T.; "Building hypertexts on a multimedia toolkit: An overview of the Andrew Toolkit Hypertext facilities", in "Hypertexts: Concepts, Systems and Applications (Proceedings of ECHT'90)", A. Rizk et al., eds., Cambridge Series on Electronic Publishing, Cambridge University Press, Cambridge, 1990
- [Shneiderman 88] Shneiderman, Benjamin, editor, Hypertext on Hypertext, Database & Electronic Product Series, ACM Press, 1988
- [Stotts 89] Stotts, P.D.; Furuta, R., "Petri-Net-Based Hypertext: Document Structure with Browsing Semantics", ACM Transactions on Office and Information Systems, 7(1), January 1989.
- [Tomba 89] Tomba, F., "A Data Model for Flexible Hypertext Database Systems", ACM Transactions on Information Systems, Jan. 1990.
- [Travers 89] Travers, M.; "A Visual Representation for Knowledge Structures", Proc. Hypertext '89, ACM, Baltimore, 1987. pp. 147-58
- [Trigg 88] Trigg R.H., "Guided Tours and Tabletops: Tools for Communicating in Hypertext Environments", ACM Trans. of Office Information Systems, Oct. 88, vol. 6, n.4.
- [Utting 90] Utting K; Yankelovich, N.; "Context and Orientation in Hypertext Networks", ACM Trans. on Information Systems, 7. (1990) pp. 58-84
- [Walker 88] Walker, J. H.; "Supporting document development with Concordia", IEEE Computer 21 (1988) pp. 48-59

- [Wiederhold 83] Wiederhold, G. "Database Design", Second Edition, McGraw Hill, 1983.
- [Winograd 88] Winograd, T.; "A Language/Action Perspective on the Design of Cooperative Work", Computer-Supported Cooperative Work: A Book of Readings, Irene Greif, ed., Morgan Kaufmann Publishers Inc., San Mateo. 1988. pp.623-56
- [Winter 89] Winter R., Ludwig Van Beethoven Symphony Number 9, CD Companions Series, The Voyager Company, 1989
- [Yankelovich 88] Yankelovich, N.; Haan, B.J.; Meyrowitz, N. K.; Drucker, S.M.; "Intermedia: The concept and construction of a seamless information environment", IEEE Computer 21 (199) 91-967.