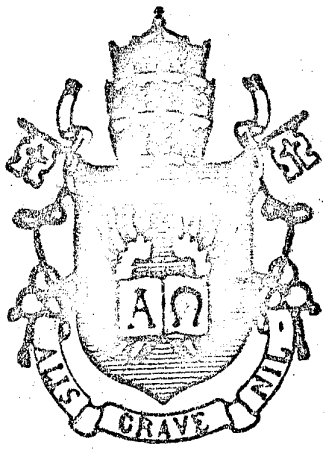

PUC

Series: Monografias em Ciência da Computação,
No. 23/90

NEW PERSPECTIVES FOR HYPERTEXT APPLICATIONS USING MODEL-BASED DESIGN

Franca Garzotto
Daniel Schwabe
Paolo Paolini

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. 23/90
Editor: Paulo A. S. Veloso December, 1990

NEW PERSPECTIVES FOR HYPERTEXT APPLICATIONS USING MODEL-BASED DESIGN *

Franca Garzotto **
Daniel Schwabe
Paolo Paolini **

* 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-90-76.

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

Abstract:

We describe a novel use of hypertext within hybrid applications - those involving the integration of formal and informal knowledge about an application domain. This integration is achieved using a model-based approach to hypertext application design; this model can be used even if no integration with formal knwoledge is needed. As a consequence of this approach, we also propose a structured approach to hypertext application design that aims at defining the "structure" of the hypertext before the actual contents of the nodes are defined.

Keywords:

Knowledge based systems, hypertexts, muitinedia systems, authority models for hypertext.

Resumo:

Neste artigo descreve-se um novo uso para hipertextos, no contexto de aplicações híbridas - aquelas que envolvem a integração de conhecimento informal e formal acercá de um domínio de conhecimento. Esta integração é conseguida através da utilização de um enfoque baseado em modelos para o projeto de aplicações hipertextuais: este modelo pode ser utilizado mesmo quando não é necessária a integração com conhecimento formal. Como consequência deste enfoque, é proposto também uma abordagem estruturada ao projeto de aplicações hipertextuais, que procura definir a estrutura do hipertexto antes da definição dos conteúdos dos nós propriamente ditos.

Keywords:

Sistemas baseados em conhecimento, hipertextos, sistemas multimidia, modelos de autoria para hipertexto.

NEW PERSPECTIVES FOR HYPERTEXT APPLICATIONS USING MODEL-BASED DESIGN

Franca Garzotto, Daniel Schwabe, Paolo Paolini
Politecnico di Milano

Abstract - We describe a novel use of hypertext within hybrid applications - those involving the integration of formal and informal knowledge about an application domain. This integration is achieved using a model-based approach to hypertext application design; this model can be used even if no integration with formal knowledge is needed. As a consequence of this approach, we also propose a structured approach to hypertext application design that aims to define the *structure* of the hypertext before the actual contents of the nodes is defined.

1. Introduction

Traditional hypertexts have been used to describe complex application domains. The complexity is represented mostly through a large number of connections (links) between nodes, and it is commonly argued that it is exactly this richness of connectivity that gives it advantages over traditional, linear text.

It is possible to claim that, in some sense, hypertexts are a form of knowledge representation. Since most hypertext systems know nothing about the contents of the nodes, and are aware only of the particular network topology of the hypertext, this representation is in some sense informal - any additional interpretation is done by the human reader. We refer to information that can only be processed by human beings as an informal representation of the knowledge about an application domain. Informal representation is most useful for domains in which the knowledge relies on some sort of "common-sense" (e.g., legislation), whose formal representation and automatic processing are very difficult.

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: paolini@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.

A different approach to knowledge representation has been taken by the so-called "Knowledge Based Systems" (KBS's) [Frost 89]. Here the application domain is formalized in some computer processable way, and once this formal description is available the system can try to solve specific problems in the application domain by manipulating the formal description. This approach is most successful when it is possible to narrow the problem area to very specific points, in domains amenable to formalization. We refer to information represented in a computer-processable form as a formal representation of the knowledge about the application domain.

Integration of hypertext and "knowledge-based" systems can be done at various levels - see for instances the articles in [Bernstein 88]. To organize the discussion, we could classify these approaches as follows.

1) Hypertext interface - In this class, the domain knowledge is all contained and processed in the KBS formalism; hypertext is used to provide a user-friendly interface to it. This interface may either allow navigational access to the knowledge, or it may be used to enhance the explanation capability of KBS ([Moore 88] contains a survey of such explanations). Examples are the hypertext interface to medical knowledge-bases [Banks 88] and the interface to CYC using hypertext [Travers 89].

2) Knowledge Based Enhancements to Hypertext - This class includes the systems in which knowledge-based techniques are used to enhance hypertext functionality. Such enhancements can help the navigation process, query mechanisms, the computation of dynamic links, etc... Examples are VISAR [Clitherow 89] and certain aspects of Object Lens [Lai 88] .

3) Hybrid systems - Systems in this class represent the knowledge both in hypertextual and using knowledge based techniques. Besides the system described in this article, other examples are the representation of engineering norms described in [Schwabe 90] and the JANUS (architectural) Design Environment [Fischer 89]. Notice also that systems in this class may also have features of kinds 1) and 2) above.

The third category above, hybrid systems, can also be seen as a special case of "semi-formal systems", which can be defined as (see [Lai 88]) "Computer systems that 1) represent and automatically processes certain information in formally specified ways; 2) represent and make it easy for humans to use the same or other information in ways that are not formally specified; and 3) allow the boundary between the formal processing by computers and informal processing by people to be easily changed".

Semi-formal systems are useful when there is an application domain where part of the knowledge is readily formalizable, and part of the knowledge is not. To achieve graceful integration between the two types of knowledge representation, it would be desirable to have some sort of notation which, on

one hand, is formal enough to be mapped onto formal representations, and on the other hand is at the appropriate level of abstraction to also be used by human beings as an aid in accessing the informal knowledge representation.

Hybrid systems, in which the informal part of the knowledge is represented in hypertextual form, open up new perspectives for hypertext applications. Furthermore, the goal of having graceful integration with formal representation may induce thinking about the informally represented knowledge at a more conceptual level, thus leading to a "disciplined" hypertext application design.

We describe in this article an example of a hybrid system, specified using a design model for hypertext applications, called HDM (Hypertext Design Model), which aims at "bridging" formal and informal representations. This model has been fully (also formally) specified in another article [Garzotto 90c].

Independently from the advantages of allowing graceful integration of representations, HDM prescribes an approach to hypertext application design which brings several advantages to the authoring process as well as to the final resulting application. To summarize, it allows a structured, systematic, "authoring-in-the-large" approach, in which the global aspects, as well as the structure of the hypertext are designed independently from content and "appearance" concerns; and it allows achieving a larger degree of independence of the particular hypertext system used for implementation.

The remainder of this article is divided as follows. Section 2 gives an example application domain in which hybrid systems are useful; section 3 discusses the overall architecture of the proposed system to support the activities specified in section 2; section 4 discusses model-based hypertext application design, also introducing HDM; section 5 presents the system specification using HDM, presenting also examples of an implementation; and finally section 6 draws some conclusions.

2. An Application Domain

To illustrate and motivate the use of hybrid systems as discussed so far, we describe a particular application domain, office automation of banking environments, and its requirements.

Consider, for example, the activities involved in a mortgage loan procedure for an (Italian) bank. To obtain a loan, a customer must fill-in an application form, which will be evaluated by a number of departments in the bank.

If the application is accepted, the customer is then requested to provide additional information about the property being bought; a specific department within the bank will check if this property satisfies all of the bank requirements. Assuming a positive evaluation of the property, a preliminary contract is drawn, and the customer, having signed this contract, receives part of the money and

actually buys the property. At this point, a final contract is drawn, in which there is a mortgage on the property defined in favor of the bank, and the customer receives the rest of the loaned amount.

The customer starts paying back the loan, with payments (say) every six months, for at least 10 years. At the end of the payment period, if the customer has kept with his obligations and paid up the mortgage, a new final contract is drawn, stating the customer's full rights to the property. Otherwise, other complex procedures are started to correct the situation.

The description just given is clearly a simplification of the actual case. For example, a simple statement such as "a contract is drawn" is actually an involved procedure, in which the relevant data must be collected and validated, the appropriate "legal" phrasing must be coined, and the proper placement of the data in the text must be defined, all according to several rules. Another example is the case where the bank now wishes to change an existing loan contract, maybe due to new government regulations, or to adapt it to a new type of loan, it is necessary to find all the portions that are affected. Some portions of the contract depend on regulations (government or bank), others depend on the nature of the object of the contract, and still others depend on the nature of the loan itself

The knowledge required to perform such activities in this type of environment is very complex. Some of the types of knowlege in the domain are more readily automatically processed, whereas others are less readily or conveniently processed automatically, since they are typically ill-defined. As a consequence, there will be tasks that are easier or harder to be automated.

Consider for example the activities related to handling contracts. Definition of new contract types requires quite sophisticated expertise, including the hability to interpretat relevant laws and regulations, knowledge about commonly accepted practices, as well as of operational factors that may affect future handling of such contracts within the organization. This type of activity involves a level of knowledge processing which is very difficult to automate, but is more easily done by humans. The definition of a certain type of contract becomes, at the end, a set of operational rules that a lay employee is able to follow in order to produce a contract instance for a specific situation. This last phase may, as a matter of fact, be automated, since the type of knowledge it employs is much more well-defined.

Even an automated activity such as contract generation may benefit from being integrated with less formalizable knowledge, in order to be better managed by human beings. The non-automated tasks (such as definition of new contract types) may also benefit from having access to some organized informal representation of the required knowledge, as a support for human processing.

In fact, much of the knowledge involved in non-automated tasks is *not* recorded explicitly anywhere, and is usually available only through some experts that simply "know" it, but are often unavailable. Even when recorded explicitly, it is oftentimes distributed throughout the organization, and not available in

organized form. As a consequence, such tasks become hard to perform, and in fact most of the time contain inconsistencies.

For example, since it is difficult to determine the rationale behind the form and contents of contracts, new types of contracts are defined by altering portions of existing contracts by addition of new parts, taking care only not to introduce obvious contradictions with other pre-existing parts. Things like redundancy or obsolescence of old parts are almost always not checked, with the end result being unnecessarily complex (and big) documents.This observation can also be made for most office procedures - they tend to evolve from pre-existing ones via small local changes, leading to a gradual loss of overall consistency due to lack of global consistency checking.

As another example, the lack of explicit representation of the rationale behind contracts and procedures also affects training. New employees can, more or less easily, learn by heart how procedures work. However, they have difficulty in dealing with unforeseen and uncommon situations, as they have little understanding for the reasons behind the rules they have learned. "Word of mouth", when available, is the only way apprentices learn about things from older, more knowledgeable employees. One of the side-effects of this situation is that personel turn-over is very disruptive to the institution, since "institutional memory" is stored inside people's minds and gets lost when people leave the organization.

It should be clear, at this point, that hybrid systems are appropriate for supporting activities in the banking environment. As an example, we would like to have a system that supports, among others, the activities related to contract handling described above, and in particular is able to automatically generate documents in this environment, and support the definition and maintenance of contracts, as well as professional training. Of the many aspects of the application domain just sketched, we would like to *formally* represent document generation knowledge and to automatically manipulate it, and *informally* represent the knowledge required for the other tasks, as a support for human processing.[1]

3. An Architecture for Hybrid Systems

3.1 Overview

In light of the discussion of the previous section, hypertextual representations seem to be good candidates for informally representing the part of the domain knowledge that cannot be automatically processed, as they conveniently provide exploratory access and friendly user interfaces, both useful for supporting human processing of such knowledge.

---

[1] It should be noted that this choice of what should be "formalized" and what should be left "informal" is somewhat arbitrary, and we make it here basically to provide an example. A real system could also "formally" represent other aspects of the activity (e.g., certain procedures) just as well.

For the application under consideration and many others, it is convenient to think of the whole knowledge representation, including the formally represented one, as being "hypertext based". This means that certain hypertext nodes contain the formal representation of the knowledge which can be automatically processed, and the user may access this knowledge by activating specialized processors which are capable of formally manipulating it. The particular division between what is kept formal and what is kept informal in each application is in some sense arbitrary, and is determined by the designer as part of the design process.

Figure 1 contains a schematic, informal representation of the proposed architecture of the system. The system is composed of two "engines", one capable of managing hypertextual descriptions, the other capable of processing formal descriptions. The grayed arrows indicate access through activation of specialized processors. Note that, given the complexity of the application domain tasks, manipulating hypertext structures should also include adding functionalities beyond simple navigation primitives, such as query mechanisms, task oriented navigation aids, etc... (as in [Halasz 87]).

For the application under consideration, the document generation task is automatically performed via specialized processors that access and manipulate the formal representation (see section 3.2); other tasks, such as training or exploratory query of the knowledge, are performed by humans mainly by browsing through the informal knowledge represented in the hypertext (e.g., laws, informal norms, etc...), with support by the hypertext engine.

The maintenance task typically needs to access both the formal and the informal representations, browsing through the knowledge represented hypertextually and also taking advantage of the specialized processor's understanding of the formal representation. For example, to integrate a new law about contracts that changes previously existing laws, the user may browse through the informal knowledge representation, looking at existing laws and their effects (which are represented as links in the hypertext) on document generation rules; once the effects are determined, the user may ask the specialized processor to retrieve all generation rules that satisfy certain conditions (specified by the human, depending on the conditions imposed by the new law). In some cases, the retrieved rules should be updated to satisfy the new law; the actual interpretation of the required changes, when needed, is done by humans.

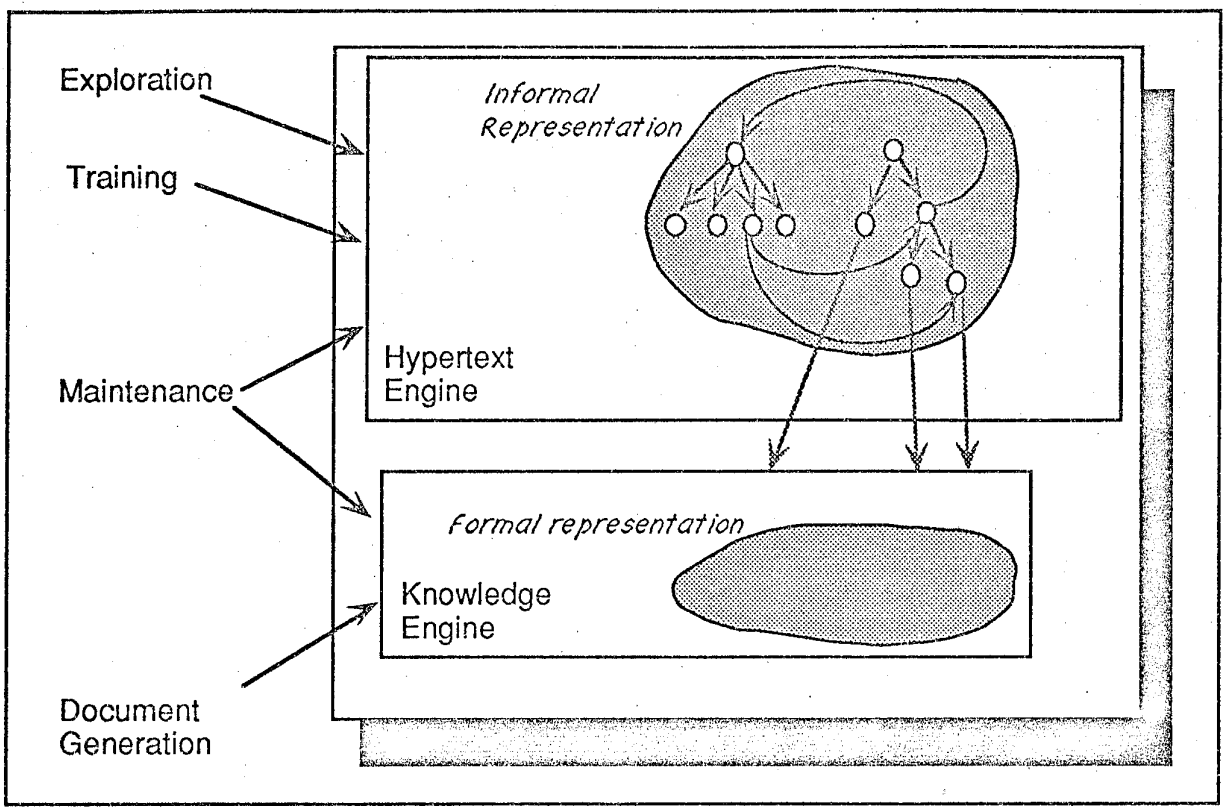

Figure 1. General architecture of the system.

The remainder of the article will concentrate on the hypertextual aspects of the system. For completeness, we briefly describe here the part of the system that manipulates the formal representation to achieve one of the tasks of the application domain, namely document generation[2].

3.2 Automated Document Generation

The Document Generator is a rule driven system that takes as input a set of raw data and produces contracts by choosing the proper variants and text clauses, and properly placing this data in the final text. The document is produced in two steps: generation of a conceptual document and generation of a concrete document from a conceptual document. A conceptual document is a formal specification of the contents of the (final) document, abstracting from its presentation (i.e, it does neither describe specific words that must be used to express particular concepts, nor does it describe their physical appearance).

A conceptual document is generated using a schema, which is simply a set of rules that determine the possible logical structure of conceptual documents. An example of a rule is

---

[2] The document generator described here is a result of the research conducted in the ESPRIT INDOC and SUPERDOC projects. A complete description of the system can be found in [%%%].

"If a mortgage is given as a warranty of a loan, this mortgage must be defined in favour of the lending bank, and the mortgaged property must be described in the contract".

A schema is used to derive a conceptual document by interpreting its rules in a given case. To produce an actual document, a presentation schema must be applied to a conceptual document description. A presentation schema is a set of rules determining the actual appearance of documents - particular terms employed, the division into sections and subsections, and layout aspects. Therefore the same conceptual document can appear in many different forms, depending on the particular presentation schema used.

In order to describe the rest of the system in more detail, the next section will introduce a design model for hypertext applications, which will ease the task of describing the various characteristics of the proposed system. Besides that, it also allows graceful integration of the formal and the informal representations, and has several other advantages, also discussed.

4. Model-Based Hypertext Design

A design model for hypertext applications[3] provides a predefined vocabulary of concepts and primitives which can be used to specify them, with little regard for the contents of the nodes. Hypermedia design models try to identify representation structures (such as nodes, links, anchors, etc...), and operational behaviour, that are common to many existing hypermedia applications in a certain domain, and attempt to provide a common language in which to describe, compare and evaluate the various hypermedia applications.

The use of a model will help to discipline the authoring activity, especially for large (i.e., significantly larger than a book) and complex hypertexts by encouraging the development of the hypertext in a structured fashion, so that its structure is designed before the actual text is actually filled into nodes.

This is very similar to what happens when developing a strongly modularized software application: designing the topology and the interconnections among modules is different from writing the code for the content of the modules themselves. Most hypertext authors agree that hypertext developers face two different (but strongly correlated) tasks: developing a network of nodes and links, and filling in the nodes' content. By analogy with

---

[3]In this article we use the term *hypertext* to denote online documents made up of a network of interconnected pieces of information (nodes). The term *hypertext system* is used for software tools used to create a hypertext. Notecards, KMS, Intermedia, Hypergate, Guide are examples of hypertext systems. Note that a single hypertext might be published in several editions, each using a different hypertext system.

the Software Engineering field we use the terminology *authoring-in-the-large* to refer to the development of the structure of the network and *authoring-in-the-small* to refer to the development of the contents and appearance of the nodes.

It is interesting to summarize the major advantages in having a design model; some are true of any kind of model, others are more specific to hypertext design models.

A design model provides a formal language in which to describe certain regularities of the application domain; this language can also serve as a representation of the domain knowledge. Clearly, this representation is quite "shallow" when compared to other formalisms, but it is at the appropriate level to serve as an interface between a "deeper" level formalization of some domain aspect, and the more informal level provided by hypertext. Being formal, it may provide a handle for "knowledge-based" type of enhancements to hypertexts.

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 and differences of applications exists.

Design models provide a framework in which the authors of hypermedia 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.

Design models can be used by *Design Tools*, 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).

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 spurious or accidental links can be as embarrassing (or even damaging) as more conventional typographic errors [Margolis 89]. This task should be greatly helped by the availability of a specification language.

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 so-called "disorientation/cognitive overhead" problem [Utting 90].

The next section presents a design model that addresses the issues discussed so far.

4.1 The Hypermedia Design Model (HDM)[4]

HDM is a hypertext application model developed at the Politecnico di Milano. 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 can be connected to other Entities or Components by Links which can be either Structural or Application links. Structural links reflect the hierarchical structure of entities; Application links connect Entities or Component to other Entities or Components to reflect application domain relations. Components can be instantiated by one or more Perspectives into Units. Units provide a reference context to information, contained in their bodies. 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 behaviour of the application. Even though HDM does not provide, so far, primitives in which to specify such Browsing Semantics, there is a *default* Browsing Semantics that is compatible with most card-oriented hypertext systems.

4.1.1 HDM Primitives

4.1.1.1 Entities and Components

We refer to an *Entity* as a concrete or conceptual real world object in the domain that is relevant for the application. Typically, an entity will denote something quite complex, whose internal structure may be further decomposed. An entity in its whole can be represented through several individual, smaller-grained pieces of information, that we call *Components*. Examples of entities are "Law 19/8/89", Dante's "Divine Comedy", Gershwins's "An American in Paris".

A Component is a piece of information describing a part of an Entity. Components are grouped into an arbitrary, application dependent, *hierarchy* to form the corresponding entity. All components of an entity are homogeneous.

Examples of possible components (with the corresponding entity from the examples above) are "Article 1" ("Law 19/8/89"), "Paradise" ("Divine Comedy"), "First Movement" ("An American in Paris").

---

[4] This section contains a summary of HDM, which is taken from [Garzotto 90c].

We have chosen hierarchies as the structure of Entities because they are a frequently occurring structure, useful in specific contexts [Brown 87]. 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, but leaves unspecified the criterion to be used when breaking up an Entity into Components since it is very much dependent on the specific application.

4.1.1.3 Perspectives and Units

Components describe pieces of information. Due to the richness of most hypertext reading environments, information may be presented in many different ways. A natural abstraction that can be made in these situations is to allow an author to refer (and think) about the concept that is being represented by a Component independently from the way it is described. In other words, the concept can be thought of as having several "perspectives".

HDM facilitates this abstraction by having *Perspectives* for Components. By the term Perspective we mean the appearance of a piece of information. The description of a component according to a given perspective is called a *Unit*, which has a body containing the information itself. HDM says nothing about bodies, as it is interested in talking about hypertext structure.

The concept of body here is meant to serve as an interface to the environment "outside" the hypertext, in the sense that the information contained in the body can be interpreted, or processed, by specialized processors (even human beings) outside the hypertext per se. The value of the body can, when desired, even be computed from other bodies or from any data outside the hypertext, including data obtained as a result of an interaction with the reader. For example, the body could be a set of document generation rules, interpreted by a generation program activated when the corresponding unit is activated. We will say more about this when we discuss browsing semantics in section 4.1.2.

A Component may have, therefore, one or more Units (corresponding to Perspectives). For example, assume that entity type "Law" has perspectives "Official Text" and "Description". If we structure an entity of type "Law" such that each component corresponds to an article, then we may say that for each article (component) of an entity of type "Law" 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 text or description of) the article as part of the "Law".

Two Units may share the same body. In the "Law" example above, assume furthermore that there is another entity, for instance "Contract" (of type "Document"), 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.

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, allowing the information to be included only once in the hyperbase and be referred to from all points.

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.

4.1.1.4 Entity Types

A common abstraction found in most data models is the notion of type. 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 entity types in the same way".

Examples of entity types (with the corresponding entity from previous examples) are "Law" ("Law 19/8/89"), "Poem"("Divine Comedy"), "Symphony" ("An American in Paris").

In HDM the set of perspectives associated to an entity type is indicative only of *possible* perspectives for its entities, so that 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 be shown when we discuss the process of mapping into node-and-link structures.

4.1.1.5 Links and Link Types

The major advantage of the hypertext model 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. The success or failure of a given application using the hypertext paradigm is heavily dependent, among other things, on the appropriate choice of links. The more the "meaning" of a link 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.

HDM differentiates between three kinds of links, which we discuss next. HDM Structural Links connects components belonging to the same hierarchy. Since entities are intended to provide a kind of "navigation context", following hierarchical links has familiar meanings such as "Next", "Previous", "Up" (e.g.,

to move higher in abstraction level), "Down" (e.g., to get more "details"), etc... The meaning of each of these relations is dependent on the particular criterion chosen to organize the hierarchy, but the reader should nevertheless be aware that she is moving "inside" the particular entity in question.

The second 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. 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, by simply traversing that link.

The third type of link in HDM are the Perspective Links. Given that Components stand for an abstraction of several Perspectives of the same subject, Perspective links allow the reader to move between different Perspectives (i.e, Units) of the same Component.

To be consistent with the notions of Entity and Entity Types, HDM defines Application Link Types as being a set of links instances whose source and destination entities are of the same entity type, respectively.

For example, the author may specify that a link of type "Is-author-of" can connect entities of type "Book" to entities of type "Person": this means that in principle all instances of "Book" (e.g., "Hamlet", "Illiad", etc...) may have a link to a corresponding "Person" (e.g., "Shakespeare", "Homer") that is its author.

4.1.1.6 Outlines

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 hypertext itself.

HDM recognizes this need by allowing (and encouraging) the usage of Entities in a special way, which we call Outline. An Outline is a special usage of types of entity (and therefore Outline instances have hierarchical internal structure) whose instances have leaf components which "point to" (are linked by application links to) nodes of the hypertext proper; the contents of outline components is navigational information. A typical example of an outline is a hierarchical index of the contents of a hypertext.

4.1.1.7 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 (such as parent-child, to-top, etc...) may be derived if desired.

Derivation rules can exist for application links too if we regard them as relations between entities (components). Properties of these relations, such as symmetry and transitivity, when existing, can be used to derive other links. The reader should note that link derivation becomes particularly powerful when used in conjunction with composition of relations.

As an example (see Fig. 1), assume that "Article 2" of a "Bank Regulation" (a component of an entity of type "Regulation") is connected to "Section 1" of a "Contract" (a component of an entity of type "Legal Document") by a link of type "Has Effects on". Then one might argue that that article of the bank regulation "Has Effects on" the whole contract too. To represent this, the author would probably set a link of type "Has Effects on" between the "Bank Regulation" and the root of "Section 1" ("Contract'). This link however can be derived straightforwardly as simple composition of the link "Has Effects on" between the two components and the structural link "to-top" between the second component and the root of its entity.

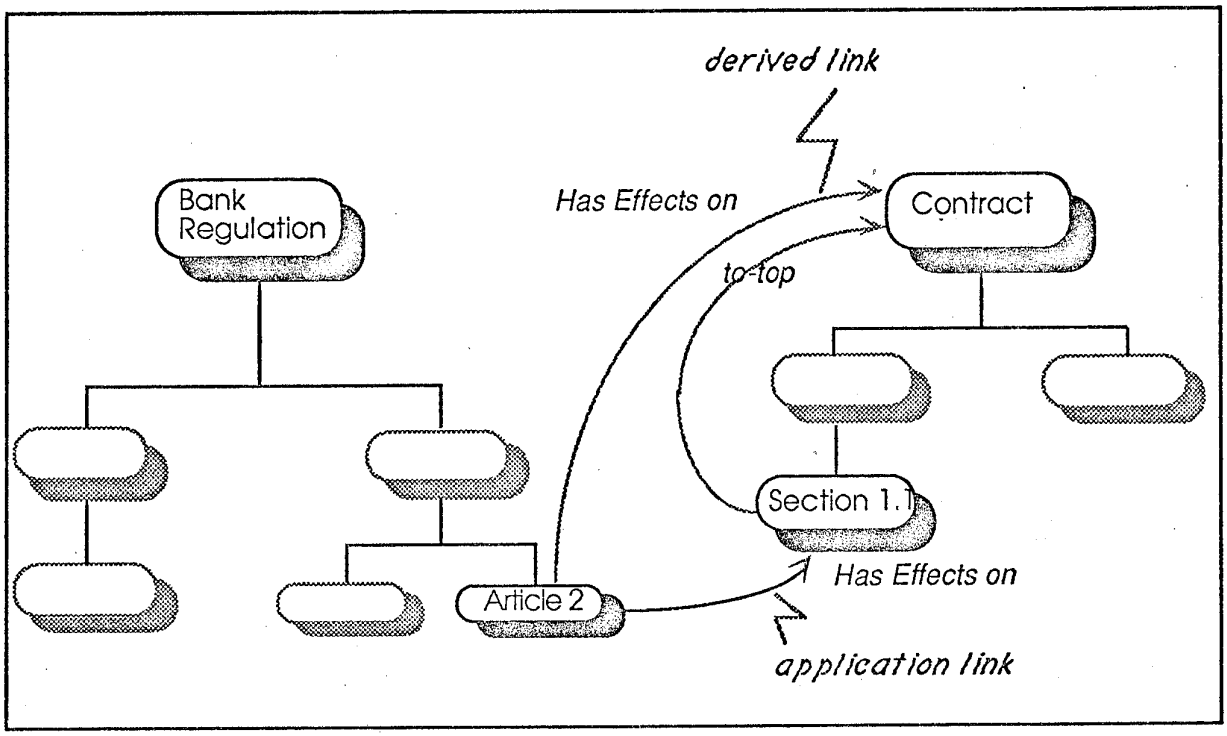

Figure 1- Example of derived link

It should be observed that defining which links are actually derived, for a given application, is a full fledged design step. The same schema (without derived links) can be reused simply by specifying different sets of links to be derived; this usually will take into account delivering the same application to different kinds of users.

So far, HDM does not itself include a language to specify such derivation rules. Nevertheless, it is possible to have such a language in a formalism outside HDM. Given such a language, the use of HDM provides a great amount of

conciseness to the 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 conceptual level and at concrete level.

4.1.1.8 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.

One may consider the entity and link type definitions, and their respective instances for a given application, a form of abstraction of the informal knowledge of the application domain. The abstraction level of the specification can be regarded as the one beyond which the knowledge representation must remain informal. In this sense, derivation rules allow automatic manipulation of this abstraction of the informal knowledge. This capability is another factor that contributes to making HDM useful for specifying hybrid systems.

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.

4.1.2 Browsing Semantics

The actual appearance of a hypertext is largely defined by its *browsing semantics* [Stotts 89]. In HDM browsing semantics will determine three further *design* choices for authoring-in-the-large:

1) What are the objects for "human consumption"; in HDM terms, what can the user perceive: Entities, Components, or Units?

2) What are the perceived links between objects; in HDM terms, which links are visible?

3) What is the behaviour when links are activated; in HDM terms, what happens when one activates a one-to-many link? Also, what happens if the body of the destination must be processed by some specialized processor to be "perceived"?

HDM, as discussed so far, does not specify any particular browsing semantics for hypertexts specified with it. Work on providing primitives for defining such browsing semantics is at a preliminary stage, but simple browsing semantics can be specified with formalisms such as Petri-Nets (see [Stotts 89]); [Garzotto 90c] contains the formal definition of the browsing semantics described in section 4.1.2.1 using a Petri-Net based formalism.

Note also that other aspects of the browsing activity, such as the actual appearance of screens and icons, and other authoring-in-the-small concerns are not discussed at all.

We have defined a simple browsing semantics that is compatible with systems at the *node-and-links* level found in many card-oriented hypertext systems; it is presented in the next sub-section.

It should be noted that, given a particular browsing semantics, it is possible to translate an HDM specification into the implementation structures of some running hypertext system; this translation process actually introduces another design dimension. Each choice made when deciding how to translate HDM links into concrete links affects the final hypertext the user will see. For this reason, it is quite natural to think that these choices should be made according to certain user profiles, therefore allowing the same hypertext design to be compiled for different classes of users. Reference [Schwabe 90a] describes a prototype compiler that allows the translation of an HDM schema instance into HyperCard, using a relational database representation.

4.1.2.1 A Default Browsing Semantics for HDM Specifications.

In this class of browsing semantics, we assume that no abstract objects (entities and components) are directly visible - only units (corresponding to the usual hypertext notion of node) can be perceived by the readers as concrete objects. In 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 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. It is necessary, therefore, to specify how concrete links can be obtained from abstract links.

Structural link translation obeys the following rule: Links between components should be translated into links between the corresponding units in 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 the "Text" perspective of the destination component, as opposed, say, to the "Graphic" perspective, which might be the default.

A simple choice to map (abstract) 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. The default representative for an entity is the default representative of its root component. This corresponds to saying that the root component of an entity (in its default perspective) "stands" for that entity. Given this notion, entity-to-entity abstract links translate into concrete links between their representatives.

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

To illustrate this rule, consider the situation (fig.2) in which there is a "Part of Law" component (belonging to an entity of type "Law'), linked to a "Step of a Procedure" component (belonging to an entity of type "Procedure"), through an "Has Effects on" application link. Consider further that the "Law" entity type has perspective types "Text" (corresponding to an informal commentary) and "Official Text", and entity type "Procedure" has "Text" and "Graphic" perspectives (the latter being a dataflow diagram for example), the "Text" one being the default for both entity types.

In this case, concrete links (corresponding to the link connecting the two components at the abstract level) will connect both the "Part of a Law:Text" and the "Part of a Law:Official Text" units to the "Step of a Procedure:Text" unit (corresponding to the default perspective).

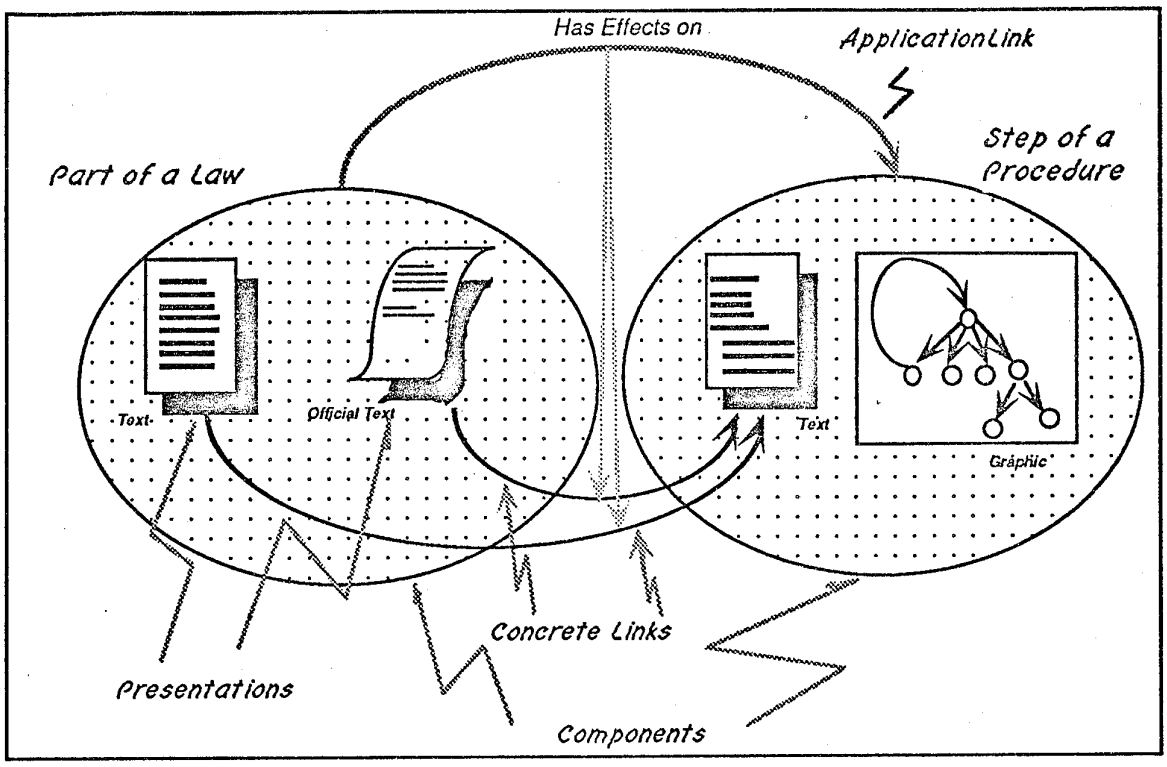

Figure. 2 - Example of concrete link generation

In hypertexts, connections among pieces of information are usually visualized 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).

This approach suggests the need of a new primitive, which captures the notion of "anchor type". Anchor types 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 link visibility.

Consider for example, that entity "Procedure" have a link of type "Formal Legal Justification" to a "Law" and a link of type "Informal Justification" to a an "Informal Regulation". The author might want to provide to the user a single reading link, maybe labeled "Justification", since the distinction might not be important for the reader. This can be achieved by specifying the anchor type "Justification" to be the union of link types "Formal Legal Justification" and "Informal Justification". It should be noted that the anchor mechanism also allows the author to selectively hide some links for some perspectives.

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 a part of the particular browsing semantics being used, which in turn is partially 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. This solution, however, is often not acceptable, and oftentimes not even possible in many systems.

An alternative approach that can be adopted to deal with this situation is to introduce the notion of *chooser*. A chooser is a mechanism associated with a single-source-multiple-target anchor that allows the selection of one of the multiple targets of the anchor. As such, choosers are implementation structures that can be generated automatically from the specification of the hypertext, using any available mechanisms present in the implementation environment (e.g., menus).

5. System Specification in HDM

As indicated in section 2, the HDM specification attempts to abstract the application domain concepts and organize them in a schematic way. The schema generated can be instantiated in many ways, corresponding to several possible applications in this same domain. We will discuss one such alternative.

In this example, the organization manipulates "Generalized Contract"s 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". "Generalized Contract"s describe *classes* of contracts in terms of their parts in common, and of the variant parts specific to each type of contract in the class. A particular contract is defined by a set of conditions that determine its structure; this will be exemplified shortly.

"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.

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**;
- *Generalized Contract* has *Structure** , *Official Text, Formal Specification* and *Description;*

- *Procedures* have *Flow Diagrams**, *Code* and *Description*;
- *Informal Norms* have *Text**.

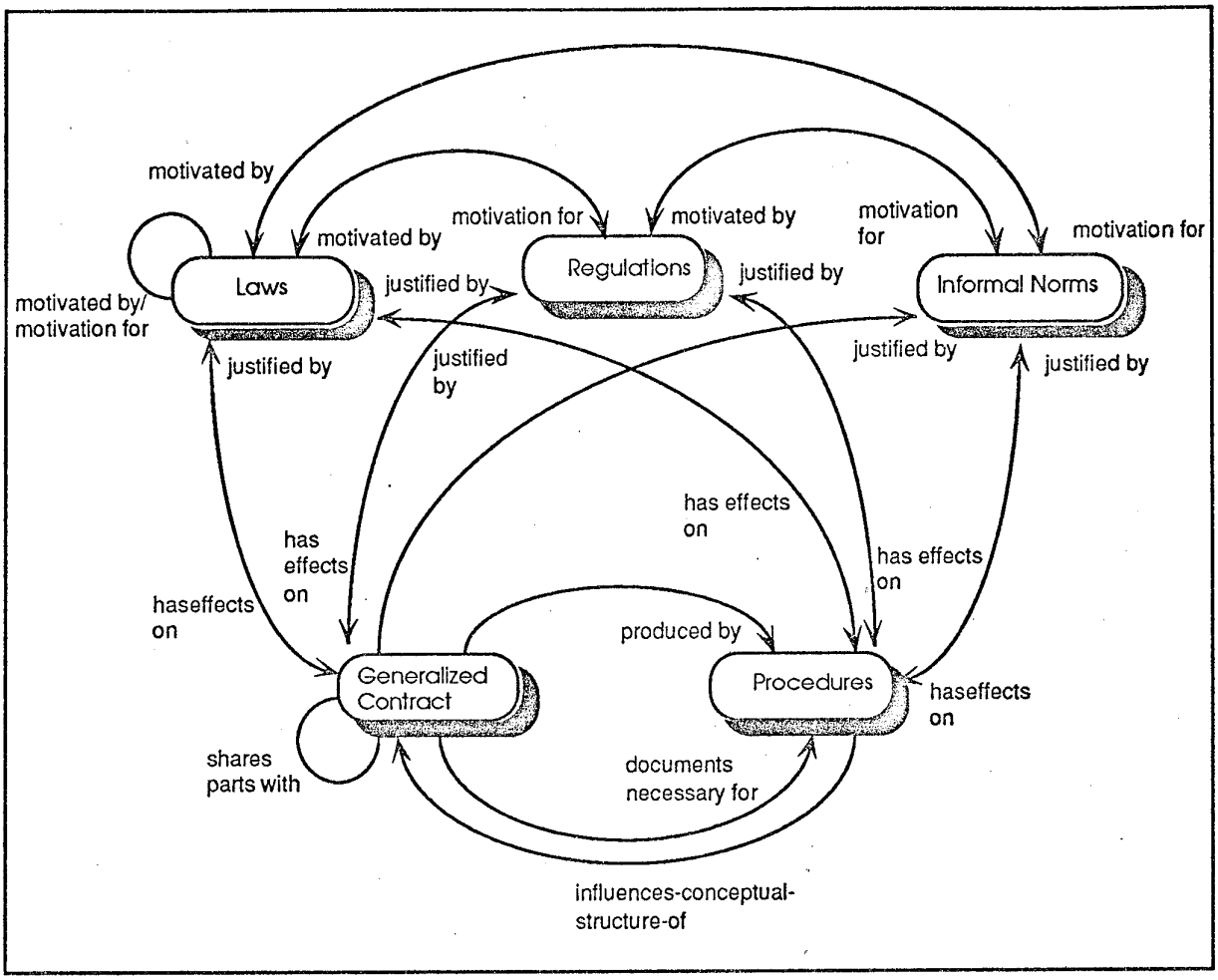

Figure 3 - The schema of the example of banking activities support environment. For each direction, the link types are the labels nearest the arrowhead.

The "Formal Specification" perspective of "Generalized Contract" contains a set of formal rules that define the conceptual structure of the contracts in the class. This perspective is not meant to be processed by humans, but rather by a special processors that are capable of interpreting these rules producing a natural language rendering of the contract. The result of this processing is actually the template for the contract, as it contains blanks (e.g., customer's name) to be filled in with specific data for a specific case.

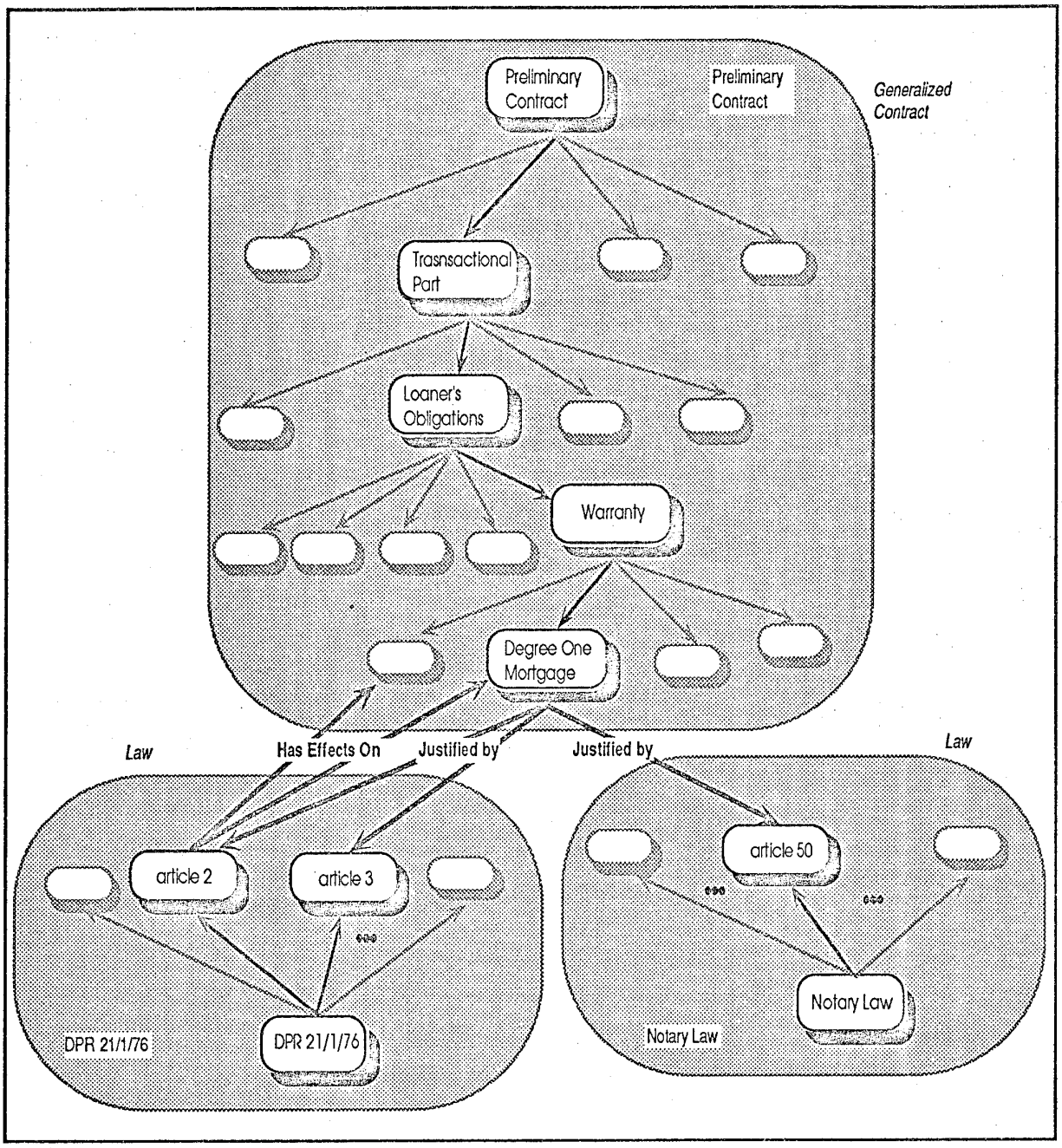

Figure 4 - An instance of the schema in Figure 2. Grayed nodes indicate components not further detailed. Gray background indicates an entity, whose type is specified next to it in italics .

Figure 4 shows a small fragment of an instance of the HDM schema for the support environment. In this example, there is one entity instance of type "Generalized Contract" (indicated by the gray background) called "Preliminary Contract", whose internal structure is outlined. Similarly, there are two instances of type "Law", called "DPR 21/1/76" and "Notary Law". Links within one entity are structural links; links with labels on them, "Has Effects on" and "Justified by", are application link instances.

Using a compiler of HDM schema instances into HyperCard described in [Schwabe 90a], a concrete hypertext was generated. This compiler takes a description of an HDM schema cast as a relational database, and generates a set of tables containing linking information. These tables are then loaded into HyperCard fields; button clicking in HyperCard causes the the corresponding destination to be retrieved from the table and displayed. As far as cards are concerned, the compiler defines only anchoring information, but does not fill-in the card contents (which is regarded as an authoring-in-the-small task). The examples that follow are taken from the resulting system. In the prototype implementation, both the hypertext engine (built in HyperCard) and the knowledge engine for document generation (built in Prolog) are currently running independently; the run time integration between the two environments is currently under development.

Figure 5 shows the "Structure" perspective of component "Preliminary Contract"; it can be observed that it contains essentially the same information depicted in the diagram of figure 4, as far as structural links are concerned. 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.

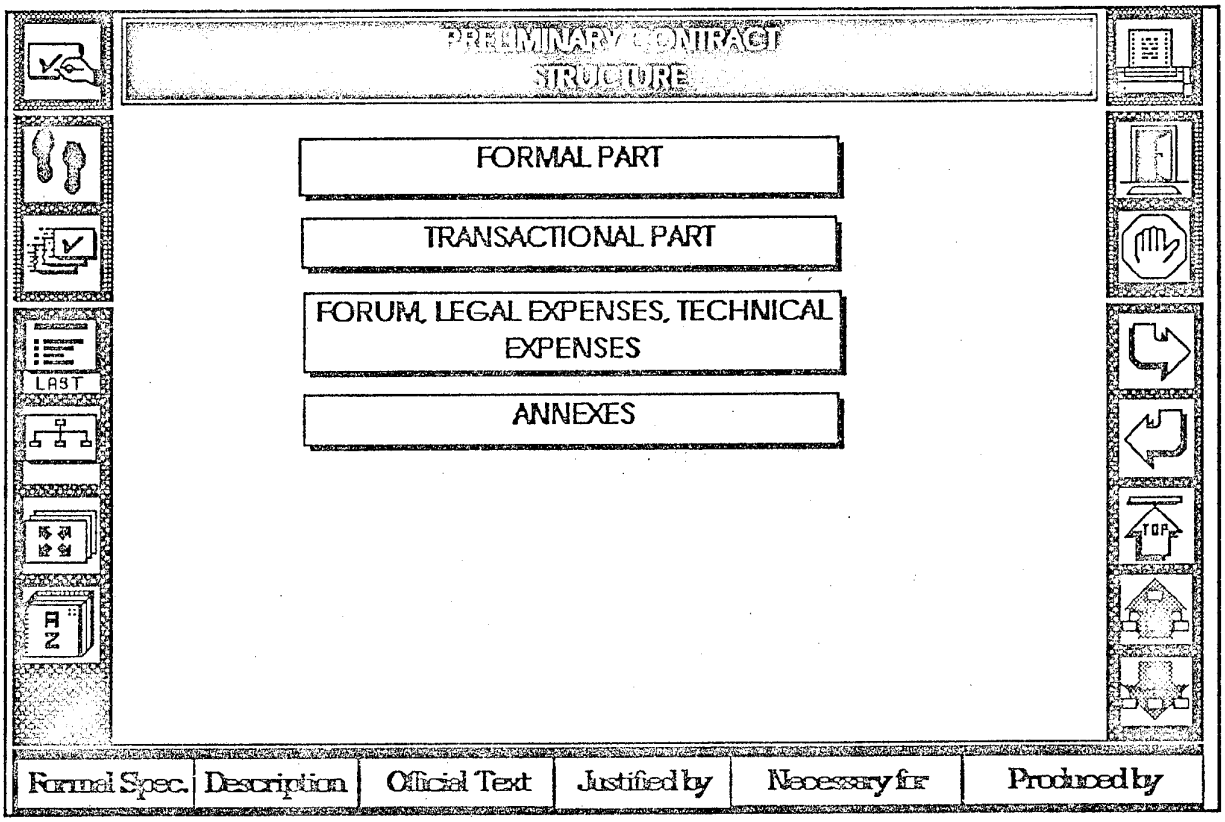

Figure 5 - Structure of component "Preliminary Contract" of the schema instance in figure 4.

Supposing the reader navigates down the structure of "Preliminary Contract", eventually reaching component "Warranty", she will see the screen shown in figure 6. Notice that the children of this component are atomic, in the sense that they are not decomposed any further.

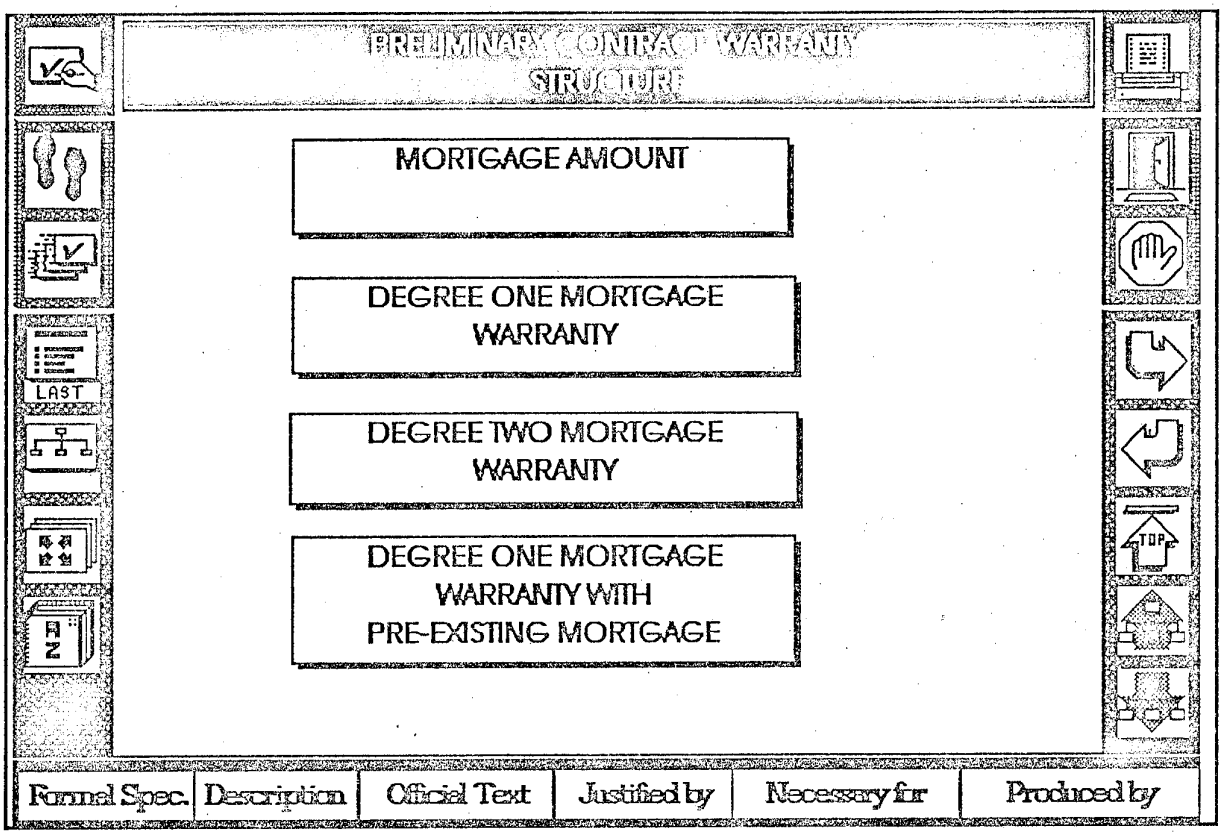

Figure 6 - Structure of component "Warranty" of the schema instance in figure 4.

The reader may wish to look at the "Description" of component "First Degree Mortgage", which can be done by selecting the corresponding field in the card, and clicking on the "Description" button, seeing the screen in figure 7.

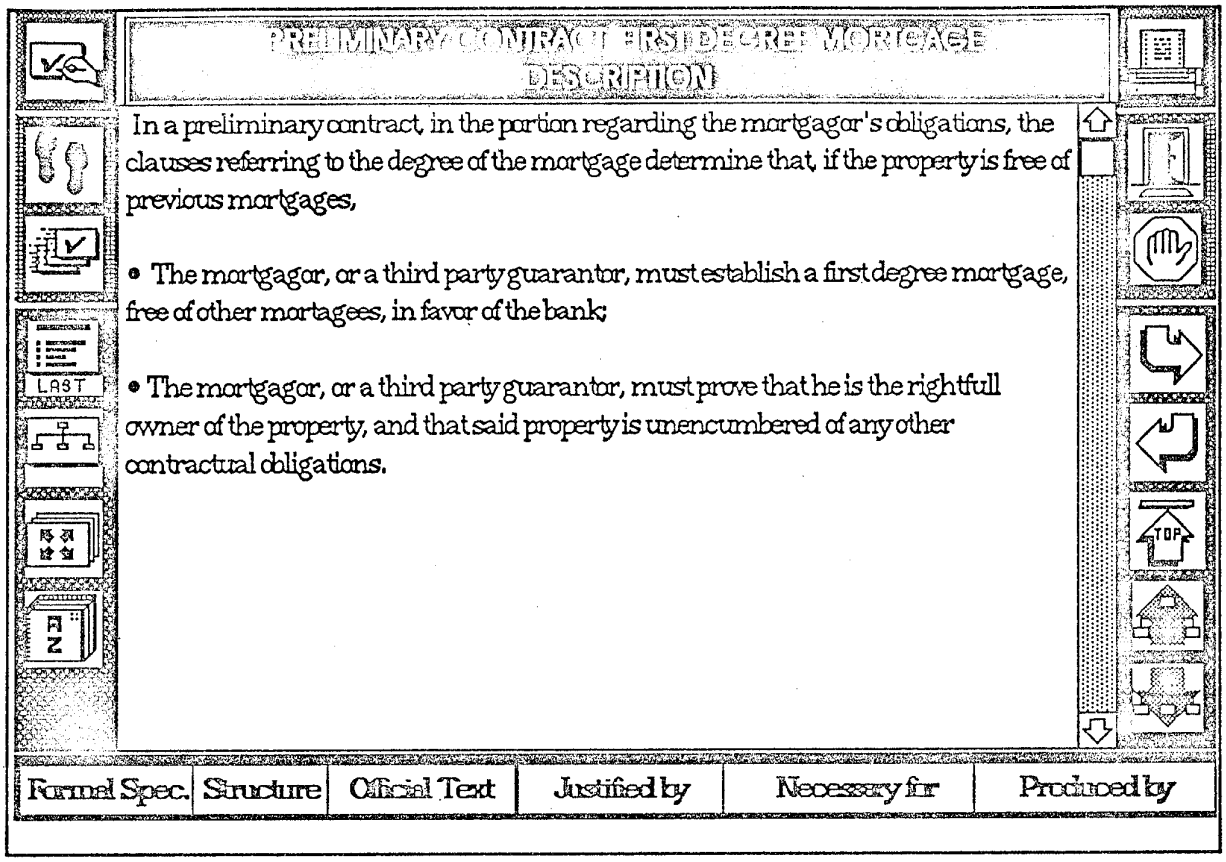

Figure 7 - Description of component "First Degree Mortgage" of the schema instance in figure 4.

From here, it also possible to look at the "Official Text" (i.e., traversing the corresponding "Perspective" link). This entails the activation of the specialized processor that generates the actual template text of the corresponding section of the final loan contract, from the rules represented in the "Formal Specification" perspective. Figure 8 shows the generated template text, and Figure 9 shows the rules that are used to generate it.

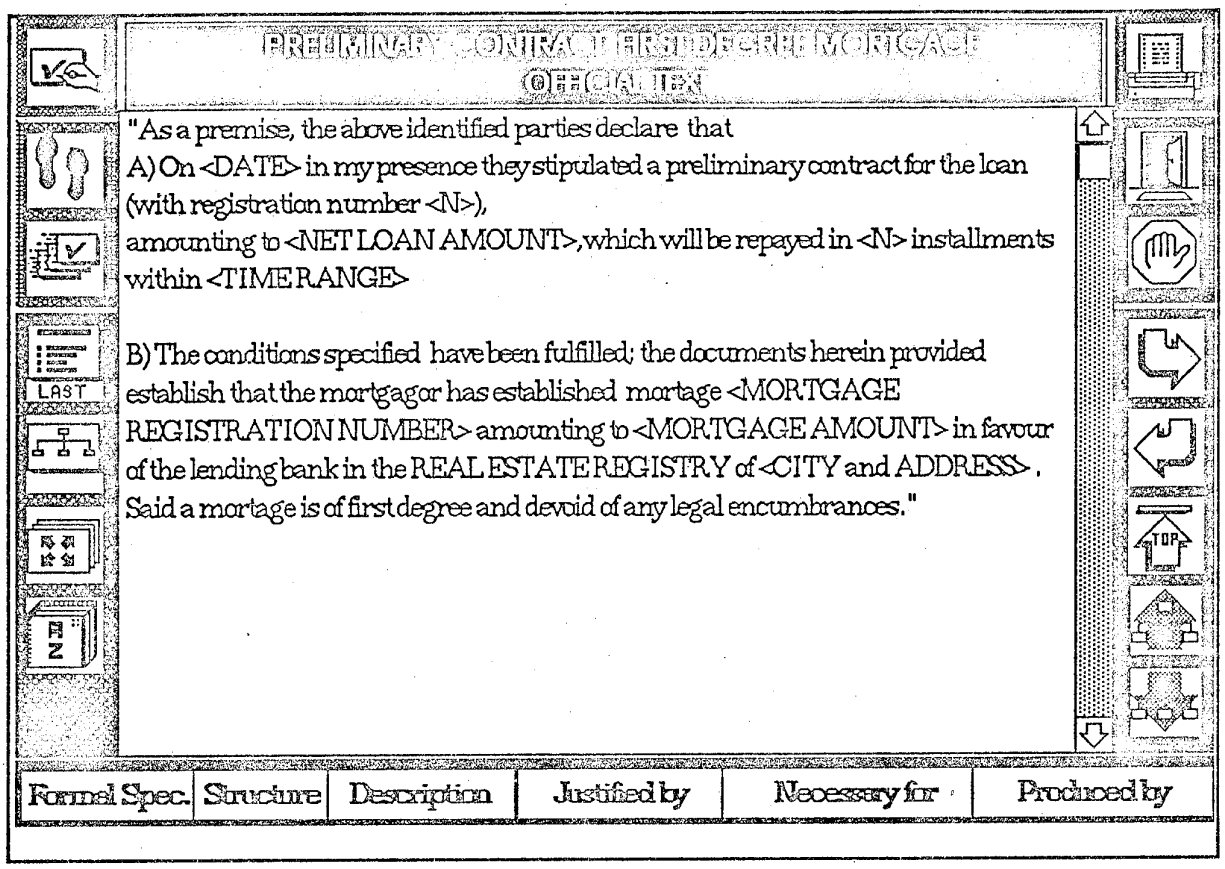

Figure 8 - "Official Text" of component "First Degree Mortgage" of the schema instance in figure 3. This text is actually generated by a specialized processor, from the rules in fig. 9.

An important observation must be made here. The rules presented in figure 9 constitute a textual rendering of the formal knowledge representation about document generation. This textual rendering is meant for human consumption, not for automated processing; therefore, in this sense it is *informal*. At this point, by traversing the link anchored with "Official Text", the user may activate the specialized processors, which interpret the *formal* representation and produce the text exemplified in figure 8.

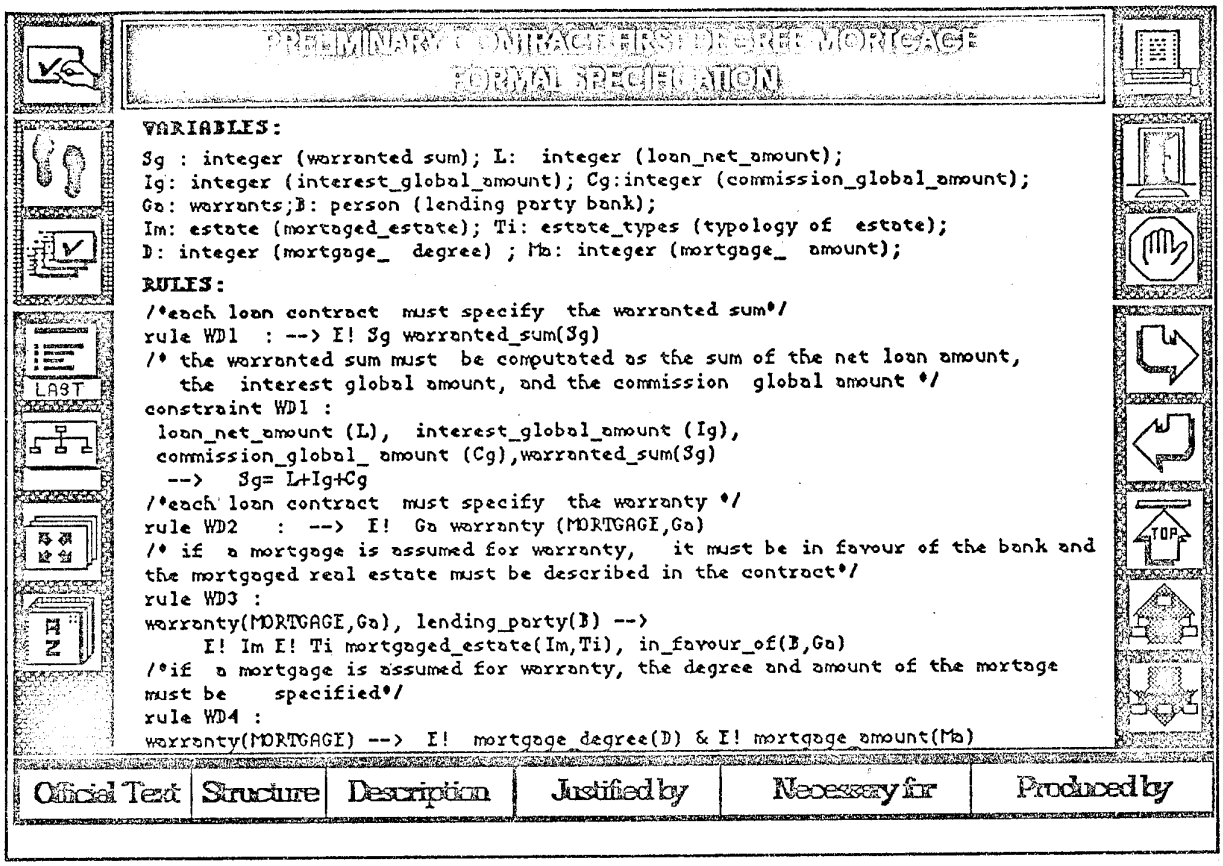

Figure 9 - "Formal Specification" of component "First Degree Mortgage" , containing the rules used to generate the text in figure 8.

The example shown here shows that the hypertext interface provides a further degree of flexibility - it allows the formal knowledge representation to be both processed by specialized processors (accessing "Official Text" perspective) and examined by human beings (accessing "Formal Specification" perspective), in a way that is consistent for the user. As a matter of fact, for most readers of this application the formal rules in the "Formal Specification" perspective would be "hidden", since they are not capable of interpreting them anyhow.

When looking at the "First Degree Mortgage" component of a loan contract, the user might be interested in seeing the legal justifications for its contents; this can be achieved by clicking on the "Justification" button. Since there are several pieces of legislation that are relevant in this case, the reader is presented with a chooser screen (see figure 10), allowing the selection of one alternative among the possible destinations. Supposing the first choice is made, the user will see the screen in figure 11, showing the "Official Text" of "article 2" (of entity "DPR 21/1/76").

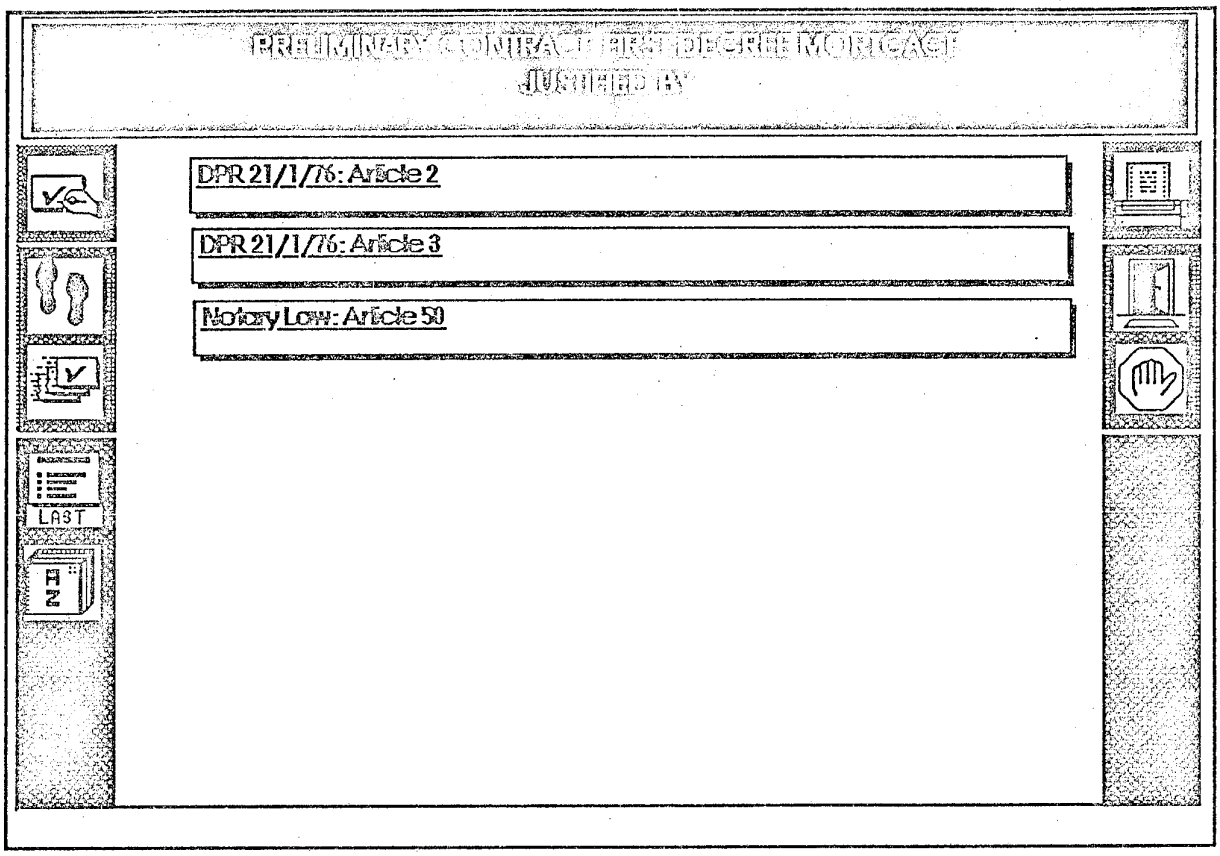

Figure 10 - Chooser allowing the choice among the possible destinations of link "Justified by" in figure 9.

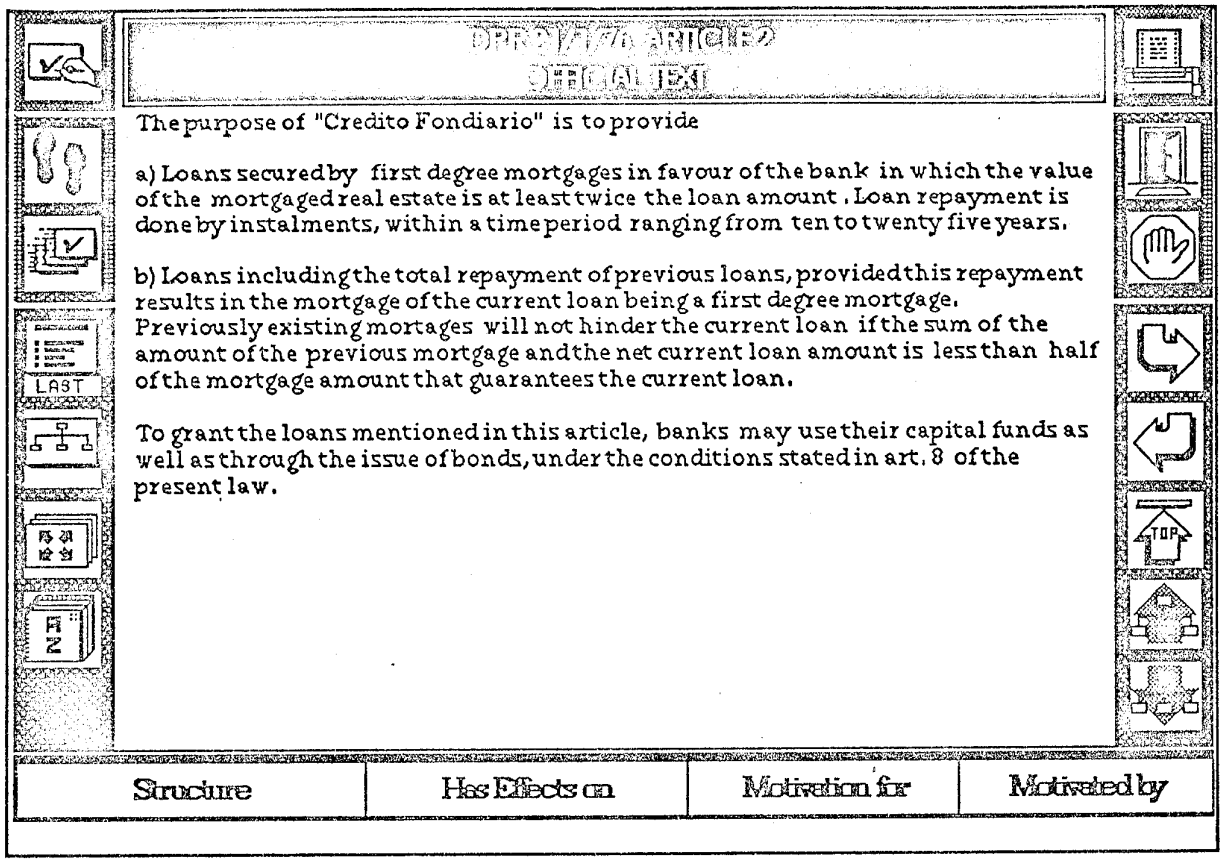

Figure 11 - "Official Text" of component "Article 2" of entity "DPR 21/1/76" , one of the legal justifications for the rule in figure 9.

6. Conclusions

6.1 Discussion

We have discussed a class of systems, called hybrid systems, that attempt to integrate formal and informal representations of knowledge in an application domain. We have argued that the hypertext paradigm provides a good framework in which to informally represent part of the application domain knowledge, facilitating its use by human beings. Furthermore, we have shown how this representation can be integrated with the formal representation of the knowledge in a consistent way, through the use of the primitives provided by our Hypertext Design Model. Hybrid systems developed using this approach constitute a novel class of applications for hypertexts.

As an example, it was discussed how a relatively complex system supporting office automation of banking environments can be specified using HDM. Taken as knowledge representation, this specification can be seen as a very shallow (abstract, high level) representation of human processable knowledge in that domain. Nevertheless, this specification may be used as an aid for humans that must use the system, for instance for understanding all the navigational

alternatives, and for abstracting from presentation aspects. The use of formal representation of machine-processable domain knowledge (the rules for document generation) as being a perspective of components provides an elegant way to interface such "formal knowledge" with the "informal", human-processable knowledge representation (for example, laws and regulations).

From a methodological point of view, HDM allows the specification of an hypertext at the "authoring-in-the-large" level, focusing on the structure and global aspects of the hypertext rather than nodes' contents. Both from the knowledge representation and from the navigation points of view, and especially for complex hypertexts, most of the "deeper" semantics lies in the connections among the nodes, rather than in the contents of the nodes themselves. As a consequence, many critical design decisions are made at the authoring-in-the-large level.

Authoring in the large can exploit common characteristics across many applications in a given domain, and it can be to some extent independent from the medium - establishing a connection between two nodes is somewhat independent from the representation of what is inside the nodes [5]. In addition, management of the topology becomes very critical as soon as the size of the hypertext exceeds a manageable limit (which is quite arbitrary and dependent, among other things, on the available technology).

Even in its present simple formulation, HDM provides conceptual and formal tools to design, analyze, and verify structural and access properties of a given hypertext at a relatively high level of abstraction and of independence from implementation environments. From this, it is possible to see the power of the authoring-in-the-large approach.

HDM specifications allow several degrees of reuse, provided by 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. Along this dimension, a schema may be reused by different applications in the same domain, by giving different instances in each case; in some cases, even the structure of a schema instance may be maintained, and the bodies of the units be replaced.

The second dimension is dynamic, provided by the separation between the static and runtime aspects, given by the notions of schema instance and its assotiation with some browsing semantics. Along this dimension, the same HDM specification (corresponding to a conceptual application) may be used to generate different running versions, by varying the browsing semantics according to different hypertext systems used for implementation.

---

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

6.2 Related Work

From the architectural point of view, the approach described here can be regarded as an alternative to Object Lens for specifying "semi-formal systems". With respect to it, as well as to other systems such as the ones mentioned in [Gaines 88, Garg 88], the major difference lies in our emphasis on hypertextual representation for informal knowledge, as opposed to adding hypertext functionality to traditional knowledge based representation mechanisms.

From a methodological point of view, 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.

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. 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".

The Trellis model [Stotts 89] and the model proposed in [Tompa 89] are mainly "behavioural" models for hypertext. In Trellis, 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. As such, we have adapted it to specify the browsing semantics of HDM specifications. Both models provide some sort of abstraction mechanism for hypertext structures, but these are not related in any way to application domain concepts.

With respect to other models such as the Dexter Hypertext Reference Model [Halasz 90], 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.

6.2 Future Work

Future work will concentrate along several lines. The first is to study and characterize complex application domains where the use of hybrid systems for supporting machine and human processing may be useful.

The second is the extension of HDM in order to include more powerful primitives that make it capable of satisfying the design and representation requirements which emerge from the first line of investigation, as well as from the analysis of other complex hypertextual applications. This includes also extending the formalization of HDM in order to provide a precise overall semantics to hybrid systems.

Finally, a third line of investigation is towards development environments which allow the specification and implementation of hybrid systems in an as integrated fashion as possible.

Acknowledgements - The work repoted here has benefited from discussions, comments and suggestions from Cristina Borelli, Andrea Caloini, Sergio Danieluzzi, and Stefano Mainetti, which worked on parts of it at certain times; we also acknowledge the comments and suggestions on HDM by Mark Bernstein, Norman Meyerowitz and John Mylopolous.

7 References

[ARG 90] ARG, "HYTEA Technical Annex", Esprit Project P5252, June 1990

[Akscyn 87] Akscyn, R.; McCracken, D.; Yoder, E.; "KMS: A Distributed Hypermedia 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.

[Banks 88] Banks, G.; McLinden, S.; Carlos, G.; "Implementation of Medical Knowledge Bases in Hypercard", in [Bernstein 88].

[Bernstein 88] Bernstein, M. (Ed); AAAI-88 Workshop "AI and Hypertext: Issues and Directions", AAAI, 1988.

[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?", Eletronic 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", in Proceedings of Hypertext '89, ACM, Pittsburgh 1989.

[Conklin 88] Conklin, J.; Begeman, M. L.; "gIBIS: A Hypertext Tool for Exploratory Policy Discussion", ACM Trans. Office Information Systems 6 (1988) 303-331

[Fischer 89] Fischer, G.; McCall, R.; "JANUS: Integrating Hypertext with a Knowledge Based Design Environment", Proceedings of Hypertext '89, ACM, Pittsburgh 1989.

[Frost 89] Frost, R. ; "Introduction to Knowledge Based Systems", Collins Professional and Technical Books, London, 1987.

[Furuta 90] Furuta R., Stotts D., The Trellis Reference Model", Proc. 1st Hypertext Standardization NIST Workshop, Gaithersburg, MD, Jan. 1990.

[Gaines 88] Gaines, B. R.; "Integration of Hypermedia with Knowledge Based Systems", in [Bernstein 88].

[Garg 88] Garg, P.; Scachi W. ; "Intelligent Software Hypertext Systems", in [Bernstein 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 90a] Garzotto F., Paolini P., Schwabe,D., Bernstein, M. "Tools for Developers", Chapter 5 of "Hypertext/Hypermedia Handbook", Devlin, J. ; Berk, E. (eds), McGraw Hill 1990.

[Garzotto 90b] Garzotto F., Schwabe,D.; Paolini P.; Caloini, A. Mainetti, S.; Borroni,S, "HDM - HYPERMEDIA DESIGN MODEL", Tech. Report.m 90-41, Dept. of Electronics, Politecnico di Milano, Oct. 1990.

[Garzotto 90c] Garzotto F., Schwabe,D.; Paolini P.;, "HDM - A Model Based Approach to Hypermedia Application Design", submitted to ACM - TOIS, November 1990. Also available as Technical Report 90-??, Dipartimento di Elettronica, Politecnico di Milano, Nov. 1990.

[Halasz 87] Halasz, F. "Reflections on Notecards: Seven Issues for the next generation of hypermedia systems", Communications or the ACM 31,7, July 1987,836-855.

[Halasz 90] Halasz F., Schwartz M, " The Dexter Reference Model", Proc. 1st Hypertext Standardization NIST Workshop, Gaithersburg, MD, Jan. 1990.

[Jordan 89] Jordan, D.; Russel, D. "Facilitating the Development of Representations in Hypertext with IDE", Proc. Hypertext '89, ACM, Baltimore, 1989

[Lai 88] Lai,K.Y.; Malone, T.W.; "Object Lens: A "Spreadsheet" for cooperative work:, Proceedings of the ACM Conference on Computer Supported Cooperative Work, Portland, Oregon, 1988.

[Margolis 89] Margolis, M.; review of "The Election of 1912", Social Science Computer Review 7 (1989) 231-3.

[Moore 88] Moore, J.D.; Swartout, W.D.; "Explanation in Expert Systems: A Survey", Technical Report ISI-RR-88-228, Information Sciences Institute, Marina del Rey, Calif., December 1988.

[Schwabe 90a] Schwabe,D.; Caloini, A.; Garzotto, F.; Paolini, P., "Hypertext Development Using a Model Based Approach", Tech. Report 90-??, Dipartimento di Elettronica, Politecnico di Milano, Nov. 1990. Submitted to "Software Practice and Experience".

[Schwabe 90b] Schwabe, D.; Feijó, B.; Krause, W.G; "Intelligent Hypertext for Normative Knowledge in Engineering", in "Hypertext: Concepts, Systems and Applications" (Proceedings of ECH'90), Rizk,A; Streitz, N.; André, J. (eds)m The Cambridge Series on Eletronic Publishing, Cambridge University Press, 1990.

[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.

[Tompa 89] Tompa, F., "A Data Model for Flexible Hypertext Database Systems", ACM Transactions on Information Systems, Jan. 1990.

[Travers 89] Travers. M.; "Visual Representation for Knowledge Structures", Proceedings of Hypertext '89, ACM, Pittsburgh 1989.

[Utting 90] Utting K; Yankelovich, N.; "Context and Orientation in Hypermedia Networks", ACM Trans. on Information Systems, 7. (1990) pp. 58-84