HYDRA: A Hypertext Model for Structuring Informal Requirements Representations[1]

Klaus Pohl and Peter Haumer

RWTH-Aachen, Informatik V, Germany

e-mail: {haumer, pohl}@informatik.rwth-aachen.de

Abstract: *The ultimate measurement for software quality is the degree to which user needs are satisfied by the system. User needs are an essential input for developing a requirements specification and, in the first place, are most often represented using natural language, pictures, or graphics (informal representations). The consideration of user needs as a driving force throughout the development process is only possible if requirements traceability is assured. Therefore, the specified requirements must be related with their sources, e.g. the user needs. Hypertext offers a technology for enabling this interrelation.*

We propose a formal hypertext model, called HYDRA, for structuring informal requirements information. HYDRA enriches the quasi standard hypertext model Dexter by introducing typed hypertext nodes and links. HYDRA is used to structure informal information during the requirements engineering process by creating formal hypertext objects which refer to the informal representations (or part of it). These formal objects are used to relate informal information with other representations (e.g. entity relationship diagrams, first order logic constraints). Moreover, the formal structure enables situated and selective retrieval of informal information. The creation of the formal objects as well as their relation with other representations is supported by the PRO-ART environment.

1 Introduction

We see requirements engineering as a continuous activity which is responsible for maintaining the requirements of a system over time and across traditional and organizational boundaries (cf. [Jarke and Pohl, 1994]). Due to the economical pressure, organizations must change faster. Thus, conceptual support for change management is becoming the central task for developing high quality systems. The ultimate measurement for software quality is the degree to which user needs are fulfilled by the system. Therefore, user needs must be captured during requirements engineering and considered during the whole system development process. To facilitate the use of user needs and to support goal-directed change in an evolving organization, requirements models must be traceable and must be carried along throughout the life time of a system [Pohl, 1995; Jarke and Pohl, 1993; Gotel and Finkelstein, 1994; Ramesh and Edwards, 1993]).

According to IEEE-830 [IEEE-830, 1984] requirements traceability consists of two parts: (1) enabling the traceability of a requirement back its origin, and (2) facilitating the referencing of

[1] appeared in: Klaus Pohl and Peter Peters, editors, Proc. of the 2nd Int. Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ 95), pages 118-134, Jyvskyl, Finland, June 1995. Augustinus, Aachen, Germany.

each requirement in future development and enhancement documentation. This differentiation between the two kinds of traceability was called backward and forward traceability by Davis [Davis, 1990] and pre- and post-traceability by [Gotel and Finkelstein, 1994].

In contrast to the many definitions proposed in the literature, solutions for the traceability problem are rare. There exist some approaches ([Alford, 1980; Flynn and Dorfmann, 1990; Alford, 1990]) and even commercial tools (e.g. *RT* from Teledyne Brown Engineering, *RTM* from Marconi Systems Technology, *RDD 100* from Ascent Logic Cooperation) which focus on requirements post-traceability. These approaches support traceability within the requirements specification (focusing on the trace of hierarchical decompositions) and between the specification and the implementation. They support the development of high quality systems by helping the requirements engineer to keep track of requirements. Moreover, they ensure that all requirements properly flow down to all levels with no requirement lost and none added in.

Unfortunately, these approaches tend to neglect the traceability of the elicitation and definition of the requirements as well as their evolution; i.e. a particular requirement is not related to its origin and no support is offered for understanding why a particular requirement was defined as it is. This is in contrast to many research and practise results, e.g. the independent surveys of Ramesh & Edwards [Ramesh and Edwards, 1993] and Gotel & Finkelstein [Gotel and Finkelstein, 1994], which elicited that requirements pre-traceability is even more important than requirements post-traceability; especially for systems which are embedded in a continuously changing environment.

For enabling requirements pre-traceability the question " which information must be captured" must be answered. We define the information to be captured according to the three dimension of requirements engineering: representation, specification, and agreement (cf. [Pohl, 1994]).

In this paper, we focus on the representation dimension, in particular the role of informal requirements representations. In section 2 we emphasize the importance of informal representations for understanding, defining, and managing requirements specification. For structuring the informal information we introduce a formal hypertext model, called HYDRA, which enables selective retrieval of informal information and their relation to other representation format (section 3). Tool support for HYDRA is illustrated using an example for structuring and retrieving informal information in section 4.2. Finally the advantages of using a formal structure for structuring and accessing informal information are summarized in section 5.

2 The Role of Informal Representations

2.1 Different Representation Formats

The representation formats used to express knowledge about the system during the requirements engineering process can be classified in three categories. The first category includes all informal representations, such as arbitrary graphics, natural language, descriptions by examples, sounds and animations. The second category subsumes the semi-formal languages such as SA-diagrams, ER-diagrams, SADT etc. The third category covers formal languages such as

specification languages (e.g., VDM [Bjoerner and Jones, 1988], Z [Spivey, 1992]) or knowledge representation languages (e.g., ERAE [Hagelstein, 1988], Telos [Mylopoulos *et al.*, 1990]).

Each of these categories offers some unique advantages. *Informal representations* like natural language are user-oriented. They are well known, since they are being used in everyday life. The expressive power offered by *informal representation* is very high and all kinds of requirements freedom are available (e.g., ambiguity, inconsistency, contradictory; cf. [Balzer *et al.*, 1978], [Feather and Fickas, 1991] for more details). *Semi-formal representations* like SA or ER diagrams are based on a structured graphical visualization of the system. The representations are clear and provide a good overview of the system ("one picture says more than a thousand words"). Additionally they are widely used in industrial contexts as a quasi-standard. In contrast to the informal representation the semi-formal representations come with formally defined semantics, which could be used for reasoning. But the formal semantic of semi-formal languages is very poor, so that most of the represented knowledge has no formal meaning. *Formal representation languages* have a richer, well defined semantic. Therefore reasoning about most of the represented knowledge is possible. Even code can be (partially) automatically generated out of them. Thus, formal representation languages are more system oriented.

The use of a particular representation language in the requirements engineering process has two main reasons. First, the current state of the specification influences the representation format used. In the early states of the process informal languages are preferred to enable the involvement of the users (cf. [Lubars *et al.*, 1993]). In contrast, towards the end of the process more formal languages are used to enable consistency checks and automated reasoning (verification) of parts of the specification. Second, a representation format is used due to personal preference, i.e. different people prefer different representations. For example user of the system most often prefers natural language, whereas the system specialist may stick to formal representation.

Since the ultimate measurement for software quality is the degree to which user needs are fulfilled by the system and the user needs are most often represented using informal representations (natural language, graphics, pictures, scenarios, etc.) informal representation play a crucial role in requirements engineering and therefore in the developmental of high quality systems. As a consequence, requirements pre-traceability must take the informal representations into account, i.e. must relate the informally represented user needs to the more formal specifications gained out of them.

2.2 Need for Structuring Informal Requirements Knowledge

To enable integration of informal represented information for requirements pre-traceability and to prepare the relationship to more formal specification objects the informal knowledge needs to be structured.

First, structuring is used to support the understanding process of the user needs. It gives means to discover and mark contradictions, conflicts and not sufficiently specified requirements, as well as to mark explicitly, inherent relationships in a given text. Due to the structure it is assured that relationships which have once been recognized are never forgotten.

Further, a structure is needed to trace and manage informal requirements in the requirements engineering context. This means that the structure is used for the situation dependent retrieval of information, like declarative or key-based user queries, and the integration (and traceability) of change into the informal representations.

For enabling the definition of relationships between informal representations, the definition of constraints, and selective retrieval of recorded information the structure must be defined in a formal language. The formal definition enables the implementation of constraint checking and reasoning mechanisms.

The need for structuring informal information was also recognized by other researchers. For example the system ISHYS/DIF [Garg and Scacchi, 1989; Garg and Scacchi, 1990] enable the relation of different documents based on hypertext models. However, only documents (not parts of them) can be interrelated by the ISHYS/DIF system.

Another approach is proposed in [Kaindl, 1993]. In contrast to the ISHYS system, the RETH system enables the relation of informal requirements statements and an OOA model on a fine grained basis. However, the system does not provide sufficient support for interrelating different hypertext documents nor for structuring informal information by a set of well defined link types.

This paper proposes an approach for structuring informal requirements information which overcomes the shortcomings of the systems mentioned above. In the following section we first briefly explain the principles of the hypertext technology and characterize the advantages of using hypertext for structuring informal requirements information (cf. section3.1). Than we describe HYDRA a formal hypertext model for structuring informal requirements information (cf. section 3.3).

3 HYDRA: A Hypertext Model for Requirements Engineering

3.1 Hypertext: A Technology for Structuring Informal Representations

> "*The concept of hypertext is quite simple: Windows on the screen are associated with objects in a database, and links are provided between these objects.*" [Conklin, 1987, p. 17]

In other words, the underlying data structure of a hypertext can be defined as a graph, where the information (independent of the representation format used) is represented as nodes and the relationships between the information is represented as links of the graph. Consequently, a hypertext consists of two disjunct objects, namely *nodes* and *links*.

Hypertext nodes are the objects which refer to the information organized in the hypertext. Many hypertext models introduce a set of hypertext nodes for distinguishing between different information formats, e.g. graphics, text, speech, or content, e.g. requirements, modules, comments.

Similar, since links serve different purposes (cf. e.g. [Conklin, 1987]), most existing models provide different kinds of links. For example, links are used to relate a document reference with the referenced document itself, to attach a comment or annotation to a node, or impose

an order of reading to various nodes. Based on this observation Conklin has identified three main classes of hypertext links:

referential links are used to relate hypertext nodes without introducing any closed structure. A referential link can be directed or undirected. These links are associated with a name by which their meaning is described. Examples are `annotates` and `confers-to`.
organizationally links / structuring links are used to define a hierarchical structure of the hypertext. They mostly connect a parent node with its children and thus form a subgraph within the hypertext network graph. Examples for those kind of links are ordering links defining a sequence or sets of nodes.
key links are computed during the use of the hypertext system; most often by textual (syntactical) search or key-word search.

However, most existing hypertext systems provide no formal representation of the hypertext structure and therefore are unable to support consistency checks and selective retrieval.

Nevertheless, hypertext provides the appropriated medium for structuring and interrelating informal requirements information. But for enabling consistency checks, selective retrieval, and support for change propagation, a formal hypertext structure is needed. This structure must offer suitable node types for classifying the recorded information as well as link types for controlled interrelation of these information.

Therefore, we have developed a formal hypertext model, called HYDRA. HYDRA is a specialization of our generic hypertext model (cf. section 3.2). In contrast to the generic model HYDRA offers domain specific node and link types (cf. section 3.3).

3.2 The Generic Hypertext Model: GHYM

The idea to develop a generic hypertext model first was influenced by an observation made on the *Hypertext Standardization Workshop* held by the National Institute of Standards and Technology (NIST) in 1990. The most recognized standard model of this workshop is the *Dexter Hypertext Reference Model* (cf. [Halasz and Schwartz, 1990]). Our generic hypertext model (GHYM) is based on the Dexter Model and the hypertext model proposed by [Eherer, 1993; Eherer and Pohl, 1992]. It refines the two basic hypertext concepts nodes and links as depict in figure 3.1.

According to the Dexter model we distinguish *atomic* nodes and *composite* nodes[2]. Atomics nodes (`HT_Atomic`) are primitives which can contain arbitrary information in arbitrary representations: text, graphic, pictures, spreadsheets, sounds etc. The representation format is expressed by an attribute of atomic node (`HT_Representation`). The abstract atomic`HT_Atomic` node is specialized in two concrete classes of fissile atoms and static ones. `HT_fissile-Atom` can be split into a sequence of new `HT_fissile-Atom` nodes and latter describes nodes which information contents is not dividable, like pictures etc.

Composites (`HT_Composite`) are composed of nodes, atomic and composite (modelled by the link `HT_ConsistsOf`) Composite do not refer to any information themselves. Moreover, a

[2] cf. also [Kilov, 1994] for a discussion on structuring hypertext using composite nodes.

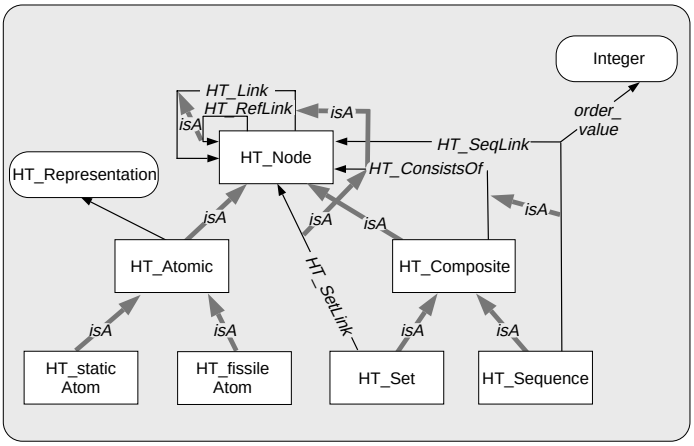

Fig. 3.1 The generic hypertext model.

composite node can not contain itself (neither directly nor indirectly). We differentiate between `HT_Sequences` by which a certain order of arbitrary nodes is represented and `HT_Sets` which are used to represent an unordered relation of nodes. Correct construction of the two specializations is assured by the specialization of the `HT_ConsistsOf` link (`HT_SeqLink`,`HT_SetLink`).

In addition to the structure of the GHYM model we have defined a set of basic operations (cf. [Haumer, 1994]). An example is the split-node operation which allows the creation of a hypertext sequence out of an atomic node. Other examples are the creation of hypertext objects which assigns a new unique object-id, the destruction of hypertext objects[3], or test operations for the determination if an object with a given object-id is contained transitively in a composite object[4].

The generic hypertext model was formalized using O-TELOS [Jarke *et al.*, 1994] (cf. [Haumer, 1994]).

3.3 The Specific Hypertext Model: HYDRA

HYDRA is a hypertext model for structuring informal knowledge using node and link types which are formally defined and enriched by constraints.

One of the main extensions of HYDRA are specific hypertext nodes. Thereby various views on a hypertext document can be created. Moreover, HYDRA provides domain specific hypertext links which are defined as specializations of the referential link type. These extensions are depicted in figure 3.2 and explained below.

Node Types

HYDRA adds further structuring concepts to the generic model. These objects are subsumed by the class `HT_Object`.

The class `HT_Hypertext` enables the definition of different (disjunct) hypertexts. A hypertext`HT_Hypertext` can consists of various `HT_Views` and `HT_Documents`. An `HT_Document` always represents a sequence of `HT_Nodes`. Thus, an `HT_Document` can be used to represent normal textual

[3] Which is refined in the specialization hierarchy depending on which kind of object is destroyed

[4] More operations are described in [Haumer, 1994].

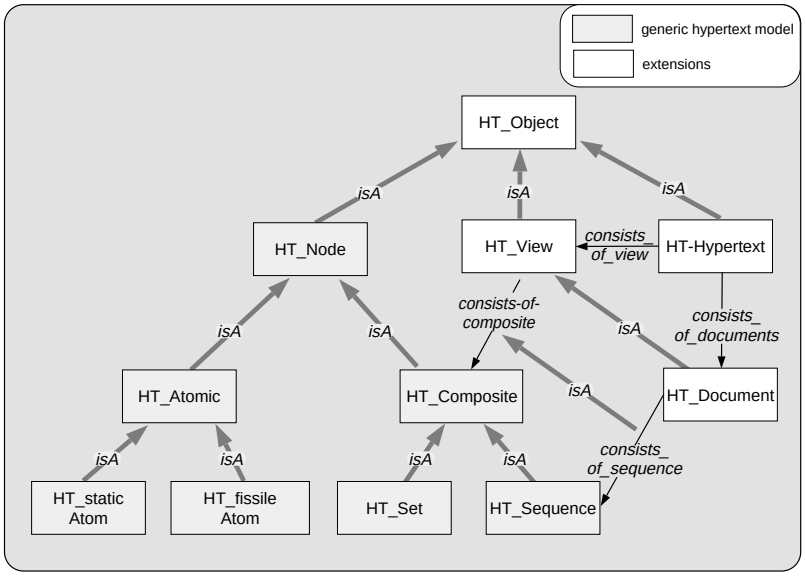

Fig. 3.2 The hypertext model for requirements engineering (structuring extensions).

documents in a linear manner, e.g. a specification according to the IEEE standard. In contrast, an `HT_View` can consists of `HT_Composite`, i.e. it can consists of `HT_Sequence`as well as `HT_Set`. Therefore, an `HT_View` can be created between different documents, e.g. between the *minutes* of a meeting and the result of a *user interview*. The introduction of these concepts goes along with the specification that every `HT_Atomic` node must be related to exactly one document, and that every `HT_Composite` must be related to either an `HT_View` or an `HT_Document` (model as formal constraint). This ensures that there is no isolated node and that a node is always associated to exactly one document.

The notion of hypertext document is further refined (not shown in figure3.2, cf. [Haumer, 1994]) into

- `HT_Interview`, used to represent protocols of user interviews
- `HT_Minutes`, used to represent results of team meetings
- `HT_Userspecs`, used to represent general information collected and acquired during requirements engineering in combination of any informal, semi-formal or formal modeling methodology
- `HT_Notes`, used to represent personal notes about the specifications (specialization of Userspecs)
- `HT_Minispecs`, used to represent the specification of a process of a data flow diagram (DFD) which is not refined (specialization of `HT_Userspecs`)

Each hypertext node of a special hypertext document inherits the node type of the document automatically (Assured by a deductive rule). For example the type `HT_MinispecsNode` will be assigned to a node contained in a document of the type `HT_Minispecs`.

In addition, the actions which can be applied to a document of a special type have been restricted by O-TELOS constraints, e.g. a requirements engineer can change his personal notes, but is not allowed to change a document of the type `HT_Minutes`.

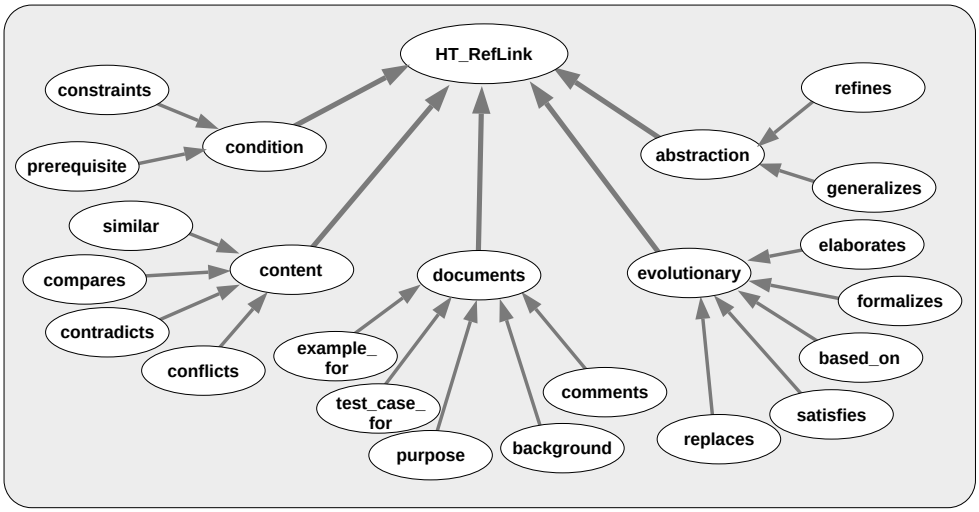

Fig. 3.3 The taxonomy of referential link types

Link Types

In addition to the node types HYDRA offers a set of link types which are based on a comprehensive literature survey (e.g., *requirements engineering*: [Davis, 1990; Dorfman and Thayer, 1990; Ramesh and Dhar, 1992; Ramesh, 1993; Gotel and Finkelstein, 1994; Greenspann *et al.*, 1994], *text analysis*: [Mann and Thompson, 1987; Rösner and Stede, 1993], *knowledge acquisition*: [Neubert and Oberweis, 1992; Neubert, 1993; Hofmann *et al.*, 1990], *recording design rationale*: [Conklin and Begemann, 1988; Conklin and Yakemovic, 1991; Ramesh and Dhar, 1992; Fischer *et al.*, 1989], *hypertext*: [Delisle and Schwartz, 1986; Delisle and Schwartz, 1987; Bigelow, 1988; Garg and Scacchi, 1987; Carando, 1989; Streitz *et al.*, 1989; Streitz *et al.*, 1992; Jarwa and Bruandet, 1990; Garg and Scacchi, 1990; Lafferty *et al.*, 1990; Schuler and Smith, 1990; Wilson, 1990; Cybulski and Reed, 1992; Kaindl, 1993; Kilov, 1994; Schütt, 1993; Wood *et al.*, 1994]).

Figure 3.3 depicts the taxonomy of referential link types which is based on the generic `HT_RefLink` into five disjunct link classes, namely: *condition*, *content*, *documents*, *evolutionary*, and *abstraction* links. All the concrete link types are defined as specializations of these classes. In the following we briefly characterize each link type. A detailed description of the derivation of the various links as well as their formalization in O-Telos can be found in [Haumer, 1994].

3.3.1 Condition Links

Constraint links are used to relate a constraint to a particular object. For example, a functional requirement *"find out which borrower last checked out a particular copy of book"* can be constrained by another informal requirement *"Library staff status is required to find out which borrower last checked out a copy of book"*.

Prerequisite links are used to relate conditions to a requirement which must be fulfilled to enable an implementation of the requirement. For example, if it is known that a particular functional requirement can only be realized, if the underlying hardware has a certain property, the two objects will be related using a *Prerequisite* link.

3.3.2 Content Links

Similar links are introduced between similar objects.

Compares links are introduced between the object which describes the result of a comparison (object *A*), and the objects *O1,...,On* which have been compared, i.e. the object *A* is related to the objects *O1,...,On* using a referential link of type *compares*.

Contradicts links are introduced between requirements which raise an inconsistency, e.g. one requirement states that *"staff status is required to perform function X"*, the other states that *"everybody can perform function X"*.

Conflicts links, in contrast to contradicts links, do not represent an inconsistency in the current specification. A conflict link between two objects indicates that fulfilling one requirement has negative influence on the other one. However, conflicts links may be replaced by contradicts links later on, e.g. when a deeper understanding of the system is gained.

3.3.3 Documents Links

Example_for links are used to relate examples (scenarios) to an arbitrary requirement.

Test_Case_for links are used to relate test cases to the corresponding requirement. These test cases must be used to validate the implementation of the particular requirement during the test and implementation phase.

Purpose links relate a description of the purpose, e.g. an informal text, to an arbitrary requirement, e.g. a functional requirement.

Background links are used to relate information to an arbitrary requirements object which provides some more details about the requirements; e.g., if a certain function should consider a standard *Z*, a description of the standard or a reference to the standard may be related to the functional requirement by a referential link of the type *background*.

Comments links are used to relate arbitrary information to a requirement object.

3.3.4 Evolutionary Links

Replaces links are used to express that a particular requirement (source object) was replaced by the target object.

Satisfies links between two requirements objects express that if the target object is reached by the final system the source object is also satisfied. Assume the requirement *"(R1) journals should always be in the library"* and the requirement *"(R2) journals cannot be checked out"*. If the requirements engineering team decides that fulfilling requirement *R2* will lead to the satisfaction of requirement *R1*, a *satisfies* link is introduced between the two requirements.

Based_on links are used to express that the source object has influenced the definition of the target object, e.g. a particular decision has caused the statement of new requirements.

Formalizes links are used to express that the target object is defined more formally than the source object, e.g. a constraint may be expressed in natural language by the user and formalized by the requirements engineer using first order logic. The *formalizes* link type is a specialization of the *based_on* link type.

Elaborates links relate a more comprehensive description gained later in the development process to the original requirement. For example, a user may name a new requirement within an interview which will be elaborated later on. The result of the elaboration will be linked to the original requirement using *elaborates* link types.

3.3.5 Abstraction Links

Generalizes links are used to represent that the target object represents a generalization of the source object(s).

Refines links are used to define that a particular requirement (target object) is defined in more detail by another requirement (source object). For example, the requirement *"record the persons having lent a book"* can be refined by the requirement *"record the address (consisting of the village, the street, the postal code) and the social number of all persons having lent a book."*. In contrast to the *elaborates* link defined above, the target object of a *refine* link expresses the source object in more detail, e.g., it cannot include a new requirement or a description of some aspects to be considered by transforming the source object into a design object.

3.3.6 Representation of Referential Link Types

Every link type introduced above is defined in O-TELOS as a specialization of the generic `HT_RefLink`. For example, the *comments* link is represented as

```
Individual HT_Node!HYDRA_Comments in MetaClass isA HT_Node!HT_RefLink with
end
```

In addition to the definitions of the link types, consistencies between possible institutions are ensured by constraints; an example of an O-TELOS frame is depicted in figure 3.4. The constraint `no_reflexive_relations` guarantees that an object cannot be related to itself using any referential link type. The constraint `only_one_link_of_linkclass_between_objects` ensures that only one referential link of a certain type can be created between two objects. In addition, no cycle between objects can be defined using directed link type[5], e.g. if object *A* comments object *B* and object *B* comments object *C*, there can be no transitive comments relation between object *C* and *A*. This is expressed by the constraint `no_cycle_of_one_linktype`. The constraint uses the deduced attribute `linkedtranswithsametype` which is computed by the rule `trans_to_rule`.

The taxonomy of the link types outlined above can of course be extended by additional link types or constraints which ensure a certain consistency of possible instantiations, e.g. to restrict the possible sources and destinations for a particular link type.

4 Tool Support and Example

4.1 A HYDRA based Hypertext Editor

Based on the HYDRA model we have implemented a hypertext editor which allows the user to create and access the formal hypertext structure. For persistence data storage we use the O-TELOS implementation ConceptBase [Jarke *et al.*, 1994]. The hypertext editor is based on

[5] There are only three bidirectional link types, namely *similar, contradicts, conflicts*.

```
MetaClass HT_Node with
attribute
 HYDRA_Constrains_Link     : HT_Node; {omitted other types}
 HYDRA_Comments_Link       : HT_Node;

 linkedtranswithsametype   : HT_Node

rule
 get_linkedtranswithsametype :
  $ forall nd1/HT_Node nd2/HT_Node
    (exists rl1/HT_Node!HT_RefLink
     ((From(rl1,nd1) and To(rl1,nd2)) or
      (exists nd3/HT_Node
       rl2/HT_Node!HT_RefLink
       (From(rl2,nd1) and To(rl2,nd3) and
        (rl2 in HYDRA_LinkType([rl1/link])) and
        (not(rl1 in HT_Node!HYDRA_Objectrelation_Link)))
        and (nd3 linkedtranswithsametype nd2)))) ==>
    (nd1 linkedtranswithsametype nd2) $

constraint
 no_reflexive_relations :
     $ forall nd1/HT_Node nd2/HT_Node
        (nd1 HT_RefLink nd2) ==> (not(nd1==nd2)) $;
 only_one_link_of_linkclass_between_objects :
        $ forall nd1/HT_Node nd2/HT_Node
           rl1/HT_Node!HT_RefLink
           rl2/HT_Node!HT_RefLink
    ((((rl1 in HYDRA_Documents_Link) and
       (rl2 in HYDRA_Documents_Link)) or
      ((rl1 in HYDRA_Constrains_Link) and
       (rl2 in HYDRA_Constrains_Link)) or
      ((rl1 in HYDRA_Objectrelation_Link) and
       (rl2 in HYDRA_Objectrelation_Link)) or
      ((rl1 in HYDRA_Evolutionary_Link) and
       (rl2 in HYDRA_Evolutionary_Link)) or
      ((rl1 in HYDRA_Abstraction_Link) and
       (rl2 in HYDRA_Abstraction_Link))) and
     (From(rl1,nd1) and To(rl1,nd2) and
      From(rl2,nd1) and To(rl2,nd2)) or
     (From(rl1,nd1) and To(rl1,nd2) and
      From(rl2,nd2) and To(rl2,nd1))) ==> (rl1==rl2) $;
   no_cycles_with_one_linktype :
    $ forall nd1/HT_Node
       not(nd1 linkedtranswithsametype nd1) $

end
```

Fig. 3.4 An example of the formalization of links & constraints of the HYDRA model.

an abstract hypertext machine which was implemented according to the architecture proposed by Neptune's HAM [Delisle and Schwartz, 1986] system. The abstract hypertext machine provides the generic hypertext operations which are defined for the HYDRA model through a C++ programming interface, e.g. creation of nodes, splitting nodes (cf. [Haumer, 1994] for

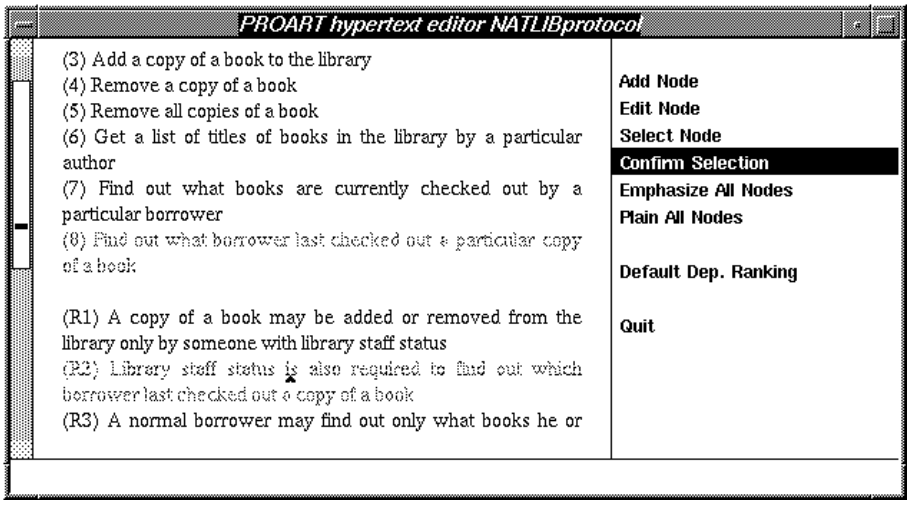

Fig. 4.5 The PHEd hypertext application displaying a document.

details). Thus, the abstract machine encapsulates the HYDRA model and the hypertext editor (shown in figure 4.5) creates and changes a concrete hypertext by using the operations provided by the abstract machine. The user hypertext editor is implemented in C++ and *Andrew* (a multimedia toolkit) [Sherman *et al.*, 1990; Borenstein, 1990].

A hypertext is displayed similar to a text in a normal text editor. By clicking into the text the surrounding hypertext node (atomic) is highlighted. By selecting the *Emphasize All Nodes* menu option all hypertext nodes are highlighted (cf. figure 4.6). Changes made in the displayed text are automatically propagated to the formal hypertext structure. In addition, new hypertext nodes can be created and hypertext links can be introduced. Moreover, selective retrieval of nodes, even from different documents, is enabled (cf. example below).

In addition to the advantages outlined above, the integration of the editor in the PRO-ART environment [Pohl, 1995] enables the interrelation of the formal hypertext objects with all other representations, e.g. objects of an ER diagram (like entities or attributes), of a SA DFD or data dictionary, constraints in a formal language etc. For example a hypertext node describing the user need to manage books in a library system can be related to an actual entity book in a ER diagram.

4.2 An Example

Suppose we start the requirements engineering process with the well known informal requirements statement for a library system proposed by (cf. figure 4.5). When the text is loaded into the hypertext editor for the first time it is represented as a single formal hypertext node, i.e. no hypertext structure exists.

Browsing through the text main relationships can be detected. To assure, that a relationship which was once detected is not forgotten over time, the requirements engineer records the relationships known to him using hypertext links. As an example, he recognizes that the statement *(8) Find out what borrower last check out a particular copy of a book* is restricted by the statement *(R2) Library staff status is also required to find out which borrower last checked out a copy of a book*. Therefore he relates the two statements by introducing a hypertext link of

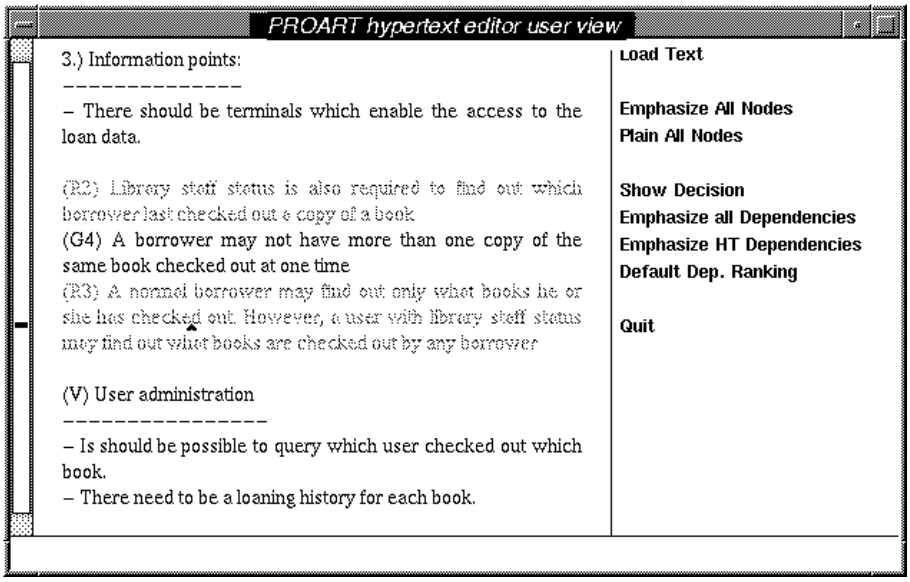

Fig. 4.6 Hypertext view with all nodes highlighted.

the type `constraint`.[6] In the following, we explain the introduction of the link from two views. On the one hand, we illustrate first the view of the requirements engineer using the hypertext editor; i.e. the user view. On the other hand, we explain how the formal hypertext structure is automatically extended by the tool.

The user of the tool chooses the menu option *Create Hypertext Link* for the creation of a link between two or more text fragments. Each text fragment is selected and the menu option *Add Node* is chosen. This causes a splitting of the original hypertext explained below. After creating a hypertext node for each text fragment (this is not necessary, if the selected text corresponds to a already existing node) the user must chose the link-type via a selection menu. Confirming the selection causes the creation of a hypertext node is in the knowledge base.

From the knowledge base viewpoint, the creation of a new node for a marked text fragment creates three new atomic hypertext nodes. Thereby the original node is transformed into a sequence which consists of these three new nodes. The information of the old node, is now distributed among the new nodes. One of them holds the information before the selection mark, another the text after and the third the selected text.

Suppose, the user know wants to create an additional link between the same hypertext nodes after a certain time. In this case the creation of the link is denied due to the `only_one_link_of_linkclass_between_objects` constraint. This constraint formally defined in the HYDRA model assures that there are no two relations between the same hypertext nodes, i.e. since the hypertext link types are disjunct only one relation between two nodes is allowed.

Beside assuring the correct instantiation of the HYDRA model the formalization of the model, among others, enables selective retrieval of parts of the hypertext structure. This has proven

[6] A hypertext link can also be created between two or more documents. For example, if the above function is restricted in a future requirements meeting to “boss-status”, the statement captured by the minutes of this meeting can also be related to the statement (8).

a very useful feature especially when the informal documentation consists of many folders. Using the hypertext editor the user has the possibility to retrieve all nodes (which can belong to different documents) which are related to a particular node. Moreover, he can restrict the selection of the nodes by a set of hypertext links. For example, if the user wants to retrieve all nodes related to the statement *(8) Find out what borrower last check out a particular copy of a book* he selects the node and chooses the menu option `Emphasize all dependencies`. By this interaction the nodes are selectively retrieved from the knowledge based and displayed in a new window as shown in figure 4.6. Note, that the nodes shown in figure 4.6 belong to different documents which where entered into the system, e.g. to the original statements shown in figure 4.5, three minutes of meetings, etc.

5 Conclusions

The main task of requirements engineering is to elicit user needs which are most often represented using informal representations. Thus, during the requirements engineering process a whole bunch of informal representation is created and normally organized using folders. Therefore access to the informal information can only be gained by browsing through the folders. On the one hand, this is very time consuming, on the other, important information may be missed. Therefore, the need of structuring and interrelating this information exists.

We have proposed to use hypertext as a principle medium for structuring informal represented requirements information. But, hypertext technology alone is not enough. In addition, a formal structure is needed which enables situated retrieval of the recorded information, propagation of changes, and assures consistency. For establishing a formal structure we have defined a generic hypertext model which provides the main structuring concepts and is based on the quasi standard model Dexter. This model was enriched by a set of node and link types. The definition of these types was based on a comprehensive literature survey and provide a rich set of concepts for structuring informal requirements information. The so gained formal hypertext structure is totally hidden to the user by appropriated tool support; our unique hypertext editor. On the interface side, the hypertext editor behaves like a normal text editor, e.g. provides insert, delete, cut and paste functions. But in addition, the tools provides the functionality for creating new hypertext nodes, relating nodes, or selective retrieval of information; also across document boundaries.

Thus, HYDRA and the hypertext editor provide the basis for structuring informal requirements information during the requirements engineering process. Thereby, almost invisible for the user, a formal hypertext structure is instantiated with formal objects which refer to the hypertext nodes and links displayed to the user. This structure enables the relation of informal information to any other representation format used during the RE process and therefore enables the integration of informal information in requirements pre-traceability; which is important for developing high quality products.

However, experiments have shown that the current informal representation formats offered (natural language and graphics) are not sufficient for capturing all kinds of user needs. We intend therefore to extend our approach by providing formats for representing and interrelating videos, speech and pictures (or part of the three). This extensions will then provide the basis for

capturing, using, and integrating real world scenarios in the requirements definition process and thereby providing a better medium for structuring and accessing user needs will be established.

References

[Alford, 1980] Mack W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Two. In *Proc. of the 4th Intl. Computer Software & Applications Conference*, pages 866–874, New York, NY, 1980. IEEE Computer Society Press.

[Alford, 1990] Mack W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Eleven – Requirements Driven Design. In Peter A. Ng and Raymond T. Yeh, editors, *Modern Software Engineering*. Van Nostrand Reinhold, 1990.

[Balzer *et al.*, 1978] R. Balzer, N. Goldman, and D. Wile. Informality in program specifications. *IEEE Transactions on Software Engineering*, 4(2):94–103, March 1978.

[Bigelow, 1988] James Bigelow. Hypertext and CASE. *IEEE Software*, 5(3):23–27, May 1988.

[Bjoerner and Jones, 1988] D. Bjoerner and C. B. Jones. *VDM'87 VDM-A Formal Method at Work*. Number 252 in LNCS. Springer-Verlag, 1988.

[Borenstein, 1990] Nathaniel S. Borenstein. *Multimedia Applications Development with the Andrew Toolkit*. Prentice Hall, 1990.

[Carando, 1989] P. Carando. Shadow: fusing hypertext with AI. *IEEE Expert*, 4(4):65–78, Winter 1989 1989.

[Conklin and Begemann, 1988] Jeff Conklin and Michael L. Begemann. gIBIS: A hypertext tool for exploratoy policy discussion. *ACM Transactions on Office Information Systems*, 6(4):140–151, October 1988.

[Conklin and Yakemovic, 1991] E. Jeffrey Conklin and K.C. Burgess Yakemovic. A process-oriented approach to design rationale. *Human-Computer Interaction*, 6(3–4):357–391, 1991.

[Conklin, 1987] Jeff Conklin. Hypertext: An Introduction and Survey. *IEEE Computer*, 20(9):17–41, 1987.

[Cybulski and Reed, 1992] Jacob L. Cybulski and Karl Reed. A hypertext based software-engineering environment. *IEEE Software*, 9(3):62–68, March 1992.

[Davis, 1990] Alan M. Davis. The analysis and specification of systems and software requirements. In Thayer R.H. and M. Dorfman, editors, *Systems and Software Requirements Engineering*, pages 119–144. IEEE Computer Society Press – Tutorial, 1990.

[Delisle and Schwartz, 1986] N. Delisle and M. Schwartz. Neptune: A hypertext system for CAD applications. In *Proceedings of ACM SIGMOD '86, Washington, D.C.*, pages 132–142. ACM, New York, 1986.

[Delisle and Schwartz, 1987] Norman M. Delisle and Mayer D. Schwartz. Contexts – a partitioning concept for hypertext. *ACM Transaction on Office Information Systems*, 5(2):168–186, April 1987.

[Dorfman and Thayer, 1990] Merlin Dorfman and Richard H. Thayer. *Standards, Guidelines and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press – Tutorial, 1990.

[Eherer and Pohl, 1992] Stefan Eherer and Klaus Pohl. Wissensbasierte Unterstützung bei der Erstellung von Hypertexten. In *Workshop Expertensysteme und Hypermedia, SEKI Working paper SWP-92-01*. Universität Kaiserslautern, November 1992. (in German).

[Eherer, 1993] Stefan Eherer. *A Software Environment for the Cooperative Construction of Hypertext*. PhD thesis, Technical University of Aachen (RWTH), Germany, 1993. (in German).

[Feather and Fickas, 1991] M. S. Feather and S. Fickas. Coping with Requirements Freedom. In *Proc. of the Intl. Workshop on the Development of Intelligent Information Systems*, pages 42–46, Niagara-on-the-Lake, Ontario, Canada, April 1991.

[Fischer *et al.*, 1989] Gerhard Fischer, Raymond McCall, and Anders Morch. JANUS: integrating hypertext with a knowledge-based design environment. In *Proc. of Hypertext '89*, pages 105–117, Pittsburgh, Pennsylvania, November 1989.

[Flynn and Dorfmann, 1990] R.F. Flynn and D. Dorfmann. The automated requirements traceability system (ARTS): An experience of eight years. In Thayer R.H. and M. Dorfman, editors, *Systems and Software Requirements Engineering*, pages 423–438. IEEE Computer Society Press – Tutorial, 1990.

[Garg and Scacchi, 1987] Pankaj K. Garg and Walt Scacchi. On designing intelligent hypertext systems for information management in software engineering. In *Proc. of Hypertext '87*, pages 409–432, Chapel Hill, North Carolina, November 1987.

[Garg and Scacchi, 1989] Pankaj K. Garg and Walt Scacchi. ISHYS. Designing an intelligent software hypertext system. *IEEE Expert*, Fall 1989:52–62, 1989.

[Garg and Scacchi, 1990] Pankaj K. Garg and Walt Scacchi. A hypertext system to manage software life-cycle documents. *IEEE Software*, 7(3):90–98, May 1990.

[Gotel and Finkelstein, 1994] O. Gotel and A. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings First International Conference on Requirements Engineering (ICRE)*, pages 94–101, Colorado Springs, Colorado, April 1994. IEEE.

[Greenspann *et al.*, 1994] Sol Greenspann, John Mylopoulos, and Alex Borgida. On formal requirements modeling languages: Rml revisited. In *Proceedings of the 16th International Conference on Software Engineering*. IEEE, Computer Society Press, September 1994.

[Hagelstein, 1988] J. Hagelstein. Declarative Approach to Information Systems Requirements. *Knowledge Base Systems*, 1(4):211–220, 1988.

[Halasz and Schwartz, 1990] F. Halasz and M. Schwartz. The dexter hypertext reference model. In Judi Moline, Dan Benigni, and Jean Baronas, editors, *Proceedings of the First NIST Hypertext Standardization Workshop*, pages 95–133, 1990.

[Haumer, 1994] Peter Haumer. A hypertext system for structuring and integrating informal requirements. Master's thesis, Technical University of Aachen (RWTH), Germany, 1994. (in German).

[Hofmann *et al.*, 1990] M. Hofmann, U. Schreiweis, and H. Langendörfer. An integrated approach of knowledge acquisition by the hypertext system concorde. In *Proceedings of the European Conference on Hypertext (ECHT–90). Hypertext: Concepts, Systems and Applications*, pages 166–179, Paris, 1990.

[IEEE-830, 1984] IEEE-830. Guide to Software Requirements Specification. , 1984. ANSI/IEEE Std. 830.

[Jarke and Pohl, 1993] M. Jarke and K. Pohl. Establishing visions in context: Towards a model of requirements processes. In *Proc. of the Int. Conference on Information Systems*, Orlando, Florida, December 1993.

[Jarke and Pohl, 1994] Matthias Jarke and Klaus Pohl. Requirements Engineering in the Year 2001. *Software Engineering Journal*, November 1994.

[Jarke *et al.*, 1994] Matthias Jarke, Stefan Eherer, Rainer Gallersdörfer, Manfred A. Jeusfeld, and Martin Staudt. ConceptBase – a deductive object manager for meta data bases. *Journal of Intelligent Information Systems*, 3:1–27, 1994.

[Jarwa and Bruandet, 1990] Sahar Jarwa and Marie-France Bruandet. A hypertext database model for information management in software engineering. In *Database and Expert Systems Applications, Proceedings of the International Conference*, pages 69–75, Wien, 1990. Springer Verlag.

[Kaindl, 1993] Hermann Kaindl. The missing link in requirements engineering. *SIGSOFT Software Engineering Notes*, 18(2):30–39, April 1993. ISSN: 0163-5948.

[Kilov, 1994] Haim Kilov. On understanding hypertext: Are links essential? *ACM SIGSOFT Software Engineering Notes*, 19(1), January 1994.

[Lafferty *et al.*, 1990] Larry Lafferty, Albert M. Koller, Greg Taylor, Robin Schumann, and Randy Evans. Techniques for capturing expert knowledge: An expert systems / hypertext approach. In M. Trivedi, editor, *Applications of Artifical Intelligence VIII, Proceedings*, pages 181–191, Orlando, Florida, USA, 1990.

[Lubars *et al.*, 1993] M. Lubars, C. Potts, and C. Richter. A Review of the State of the Practice in Requirements Modeling. In *Proc. of the 1st Intl. Symposium on Requirements Engineering*, San Diego, CA, January 1993. IEEE Computer Society Press.

[Mann and Thompson, 1987] William C. Mann and Sandra A. Thompson. Rhetorical structure theory: A theory of text organization. Technical Report ISI/RS-87–190, Information Sciences Institute, University of Southern California, 1987.

[Mylopoulos *et al.*, 1990] John Mylopoulos, Alex Borgida, Matthias Jarke, and Manolis Koubarakis. Telos: Representing knowledge about information systems. *Transactions on Information Systems*, 8(4):325–362, 1990.

[Neubert and Oberweis, 1992] S. Neubert and A. Oberweis. Einsatzmöglichkeiten von HyperText beim Software Engineering und Knowledge Engineering. In R. Cordes and N. Streitz, editors, *Informatik aktuell: Hypertext und Hypermedia 1992*, pages 162–174. Springer Verlag, 1992.

[Neubert, 1993] S. Neubert. Model construction in MIKE (model based and incremental knowledge engineering. In *Current Trends in Knowledge Acquisition – EKAW '93, 7th European Knowledge Acquisition Workshop*, Toulouse, France, September 1993.

[Pohl, 1994] Klaus Pohl. The three dimensions of requirements engineering: A framework and its applications. *Information Systems*, 19(2), 1994.

[Pohl, 1995] Klaus Pohl. *A Process Centered Requirements Engineering Environment*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen (RWTH), Germany, 1995.

[Ramesh and Dhar, 1992] B. Ramesh and V. Dhar. Group support and change propagation in requirements engineering. In M. Jarke, editor, *Database Application Engineering with DAIDA*, chapter 7, pages 221–242. Springer Verlag Berlin Heidelberg, 1992.

[Ramesh and Edwards, 1993] B. Ramesh and M. Edwards. Issues in the Development of a Requirements Traceability Model. In *Proc. of the 1st Intl. Symposium on Requirements Engineering*, San Diego, CA, January 1993. IEEE Computer Society Press.

[Ramesh, 1993] B. Ramesh. A model of requirements traceability for systems development. Technical report, Naval Postgraduate School, Monterey, California, September 1993.

[Rösner and Stede, 1993] Dietmar Rösner and Manfred Stede. Zur struktur von texten – eine einführung in die rhetorical structure theory. Technical Report FAW-TR-93005, Forschungsinstitut für anwendungsorientierte Wissensverarbeitung (FAW), Ulm, March 1993.

[Schuler and Smith, 1990] W. Schuler and B. Smith. Author's argumentation assistant (AAA): A hypertext-based authoring tool for argumentative texts. In *Proceedings of the European Conference on Hypertext (ECHT–90). Hypertext: Concepts, Systems and Applications*, pages 137–151, Paris, 1990.

[Schütt, 1993] Helge Schütt. *Datenbankserver-Unterstützung für kooperative Hypermediasysteme*. PhD thesis, Rheinisch-Westfälische Technische Hochschule Aachen, 1993.

[Sherman *et al.*, 1990] M. Sherman, W.J. Hansen, M. McInerny, and T. Neuendorfer. Building hypertext on a multimedia toolkit: An overview of andrew toolkit hypermedia facilities. In *Proceedings of the European Conference on Hypertext (ECHT–90). Hypertext: Concepts, Systems and Applications*, pages 13–24, Paris, 1990.

[Spivey, 1992] J. M. Spivey. *The Z Notation — A Reference Manual*. International Series in Computer Science. Prentice Hall, 2. edition, 1992.

[Streitz *et al.*, 1989] N. A. Streitz, J. Hannemann, and M. Thüring. From ideas and arguments to hyperdocuments: Travelling through activity spaces. In *Hypertext '89 Proceedings*, pages 343–364. ACM Press, 1989.

[Streitz *et al.*, 1992] Norbert Streitz, Jörg Haake, Jörg Hannemann, Andreas Lemke, Wolfgang Schuler, Helge Schütt, and Manfred Thüring. SEPIA: A cooperative hypermedia authoring environment. In Dario Lucarella, Jocelyne Nanard, Marc Nanard, and Paolo Paolini, editors, *Proceedings of the ACM Conference on Hypertext (ECHT–92)*, pages 11–22. ACM Press, 1992.

[Wilson, 1990] E. Wilson. Links and structures in hypertext databases for law. In *Proceedings of the European Conference on Hypertext (ECHT–90). Hypertext: Concepts, Systems and Applications*, pages 195–211, Paris, 1990.

[Wood *et al.*, 1994] David P. Wood, Michael G. Christel, and Scott M. Stevens. A multimedia approach to requirements capture and modelling. In *Proceedings First International Conference on Requirements Engineering (ICRE)*, pages 53–56, Colorado Springs, Colorado, April 1994. IEEE.