A Model-Based Approach to Hypermedia Design

Wilfried Lemahieu

Department of Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven
Belgium
tel: + 32 16 32 68 86
fax: + 32 16 32 67 32
e-mail: wilfried.lemahieu@econ.kuleuven.ac.be

Abstract
This paper introduces the MESH approach to hypermedia design, which combines established entity-relationship and object-oriented abstractions with proprietary concepts into a formal hypermedia data model. Uniform layout and link typing specifications can be attributed and inherited in a static node typing hierarchy, whereas both nodes and links can be submitted dynamically to multiple complementary classifications. In addition, the data model's support for a context-based navigation paradigm, as well as a platform-independent implementation framework, are briefly discussed.

1 Introduction
1.1 A brief history of the hypermedia concept
The term hypermedia denotes an approach to computer data organization in a manner similar to the functioning of the human brain. In essence, human cognition is organized as a semantic network in which related concepts are linked together. New information we come across is integrated into our mind’s semantic structures of existing knowledge. These structures allow for the stored information to be accessed by association.

A precursor of current hypermedia systems was mentioned as early as in 1945, i.e. long before the introduction of the modern computer, by [Bush, 1945]. He described an imaginary device called Memex as “a sort of mechanized private file and library […] in which an individual stores all his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory”. One of the key concepts of Memex was said to be its ability to link items together such that they could be accessed by association, rather than through indexing.

In 1965, Nelson came up with the term hypertext, which he defined as "a body of written or pictorial material interconnected in a complex way that it could not be conveniently represented on paper. It may contain summaries or maps of its contents and their interrelations; it may contain annotations, additions and footnotes from scholars who have examined it” [Nelson, 1965].

Generally, the concept of hypertext can be seen as the structuring of standard text with the addition of links that allow for navigation through this text in a non-linear order; each portion of the text can anchor a link that leads to a related text fragment when the anchor is ‘stimulated’. In parallel to the human brain, hypertext organizes data (i.e. text fragments) into a network structure, with semantic relationships being established through links. These links allow for navigating through and accessing data by association. Hence the purpose of the links is not only to model data interrelations, they also represent a navigational path throughout the resulting network structure. Therefore hypertext differs from other data organization techniques in that directives about how to navigate through the information space are included within the data themselves.
Most so-called first generation hypertext systems were implemented on mainframes and were strictly text-only [Halasz, 1988]. Generally, the anchors were represented by underlining the relevant text portion, with the stimulus being provided by ‘clicking’ the anchor. As PC’s inundated the computer market in the eighties, the term hypermedia became a synonym for hypertext, emphasizing the said data organization methodology being enhanced with multimedia capabilities. In such second generation hypermedia systems, the chunks of data not only consisted of text, but also pictures, animations, video and audio fragments or even virtual reality objects. As a consequence, link anchoring and the stimuli to provoke link access have become more diverse, befitting the corresponding media type. The principle, however, remains the same, according to a more up-to-date definition by [Smith & Weiss, 1988]: “an approach to information management in which data is stored in a network of nodes connected by links. Nodes can contain text, graphics, audio, video as well as source code or other forms of data”.

The components of current hypermedia systems comprise a user interface, an authoring environment to create and manage both node content and structure and a hypermedia engine with an associated storage system for the possibly heterogeneous multimedia data.

The appeal of hypermedia is based upon its ability to store complex, cross-referenced bodies of information, which can be browsed according to the user’s personal preferences. The latter, along with the resemblance to human cognition, makes hypermedia highly suitable as a tool for end user exploring and learning. Or, as put in [Bieber, 1993]: “Hypertext systems provide a non-sequential and entirely new method of accessing information unlike traditional information systems which are primarily sequential in nature. They provide flexible access to information by incorporating the notions of navigation, annotation, and tailored presentation”.

Many hypermedia systems have been conceived, some of which did not even outgrow the stadium of obscure experimental systems in research labs. Some were special-purpose built to be applied in a clear-cut field, others were general-purpose ‘shells’ to accommodate for various areas of data. Among the most publicly known (commercial) implementations are certainly Hypercard, which comes free with every Macintosh computer sold since 1987 and the Microsoft Windows Help System. However, the environment that really brought hypermedia to the public eye is undoubtedly the World Wide Web [Berners-Lee & Cailliau, 1994], promoting the hypertext paradigm as the primary access mode to all Internet-connected networks across the globe. Unfortunately, the latter WWW presents a genuine enlargement to many shortcomings that exist to some degree in all current hypermedia implementations.

1.2 Where current hypermedia applications fall short

Indeed, along with increasing popularity and worldwide adoption of hypermedia, limitations and downright deficiencies became painfully apparent. The concepts inherent to hypermedia led to inconveniences that prohibited satisfactory information retrieval by the end user as well as adequate hyperbase maintenance. Whereas non-linear navigation resulted in disoriented end users, the disorderly network of links involved an unacceptable amount of ‘manual’ work to keep the hypermedia structure up-to-date [Ramaiah, 1992].

1.2.1 User disorientation

Users navigating in a hypermedia environment are confronted with questions such as “Where am I?”, “Where do I go?” and “How do I get there?” [Rivlin et al, 1994]. The problems surrounding hypermedia navigation have been thoroughly discussed in literature, e.g. [Nelson, 1987]; [Nielsen, 1990b]; [Bernstein, 1991]. The explorative, non-linear nature of hypermedia navigation imposes a heavy processing load upon the end user. This phenomenon is known as cognitive overhead [Ramaiah, 1992]. If the freedom and flexibility become “too much” to the end user, the latter is distracted from his initial focus of attention [Hammond, 1993]. This process of cognitive overhead effecting into user disorientation and losing one’s chain of thought is referred to as the ‘lost in hyperspace’ phenomenon [Nielsen, 1990a].

1.2.2 Limited maintainability

A problem often obscured by the one described above, but nonetheless at least as stringent is the maintenance problem. The latter was certainly less than a sinecure in the pioneering hypermedia implementations.
A heavy burden upon hyperbase maintainability is the fact that, due to the absence of workable abstractions, many hypermedia systems implement links as direct references to the target node’s physical location (e.g. the URL in a WWW environment). To make things worse, these references are embedded within the content of a link’s source node [Davis, 1995]. As a result, moving a single node demands heavy maintenance efforts to restore hyperbase integrity; all nodes’ bodies have to be searched for a reference to the now-obsolete location and all found references have to be adapted. Hyperbase maintenance has become a synonym for manually editing the nodes’ contents.

Whereas manually created links already reduce maintainability to a great extent, they also have a disastrous impact upon consistency and completeness [Ashman et al., 1997]. The inability to enforce integrity constraints and submit the network structure to consistency and completeness checks, results in a hyperbase with plenty of dangling links. Needless to say that the consequences of inferior maintenance will also frustrate the end user and effect into additional orientation problems.

1.3 Objectives of this chapter

The MESH hypermedia framework as deployed in [Lemahieu, 1999] proposes a structured approach to both data modeling and navigation, so as to overcome said maintainability and user disorientation problems. MESH is an acronym for Maintainable, End user friendly, Structured Hypermedia. Its fundaments are a solid underlying data model and a context-based navigation paradigm.

The data model is based on concepts and experiences in the related field of database modeling, taking into account the particularities inherent to the hypermedia approach to data storage and retrieval. Established entity-relationship [Chen, 1976] and object-oriented [Rumbaugh et al., 1991; Jacobson et al., 1992; Meyer, 1997; Snoeck et al., 1999] modeling abstractions are coupled to proprietary concepts to provide for a formal hypermedia data model. While uniform layout and link typing specifications are attributed and inherited in a static node typing hierarchy, both nodes and links can be submitted dynamically to multiple complementary classifications. The MESH data model provides for a firm hyperbase structure and an abundance of meta-information that facilitates implementation of an enhanced navigation paradigm.

This context-based navigation paradigm builds upon the data model to reconcile navigational freedom with nested, dynamically created guided tours. Indeed, the intended navigation mechanism is that of an “intelligent book”, which is to provide a disoriented end user with a sequential path as a guidance. Such guided tour is not static, but is adapted dynamically to the navigation context. In addition, a node is able to tune its visualization to the context in which it is accessed, hence providing the user with the most relevant subset of its embedded multimedia objects.

These blueprints are translated into a high-level implementation framework, specified in an abstract and platform independent manner. The body of this chapter is dedicated to the MESH data model. Thereafter, the context-based navigation paradigm and the implementation framework are briefly discussed. A last section makes comparisons to related work and formulates conclusions.

2 A model-based approach to hypermedia application development

2.1 Orientation and comprehension in hypermedia

In [Thüring et al., 1995], a distinction is made between hypermedia systems that are destined for being wandered through, picking up information here and there, and the ones that are specifically aimed at deep understanding. It is argued how especially the second kind benefits from a structured approach. In this way, two factors are denoted as being crucial in hypertext readability and comprehensibility: coherence as a positive influence and cognitive overhead as a negative one.

2.1.1 Coherence

Coherence was already described in an earlier effort by the same authors [Thüring et al., 1991]. A coherent hypermedia document would enable the reader to construct a mental model that represents the objects and relations described in its content. Coherence should exist both on the level of a single node
and of the whole hypermedia structure. At the latter level, it can be increased by explicitly representing semantic relationships between nodes, to indicate what these nodes have to do with each other. A second measure can be to provide information about the context in which a node is displayed. This conveys a sense of continuity across separate nodes and reduces the impression of information fragmentation. Other remedies include aggregation and providing overviews of the information space.

2.1.2 Cognitive overhead

Cognitive overhead according to [Conklin, 1987] is “the additional effort and concentration necessary to maintain several tasks or trails at one time”. [Thüring et al., 1995] claim that “Every effort additional to reading reduces the mental resources available for comprehension. With respect to hyperdocuments, such efforts primarily concern orientation, navigation and user-interface adjustment.”

As a solution, they suggest how the hypermedia environment should offer the reader maximal support to identify his current position within the hypermedia structure and to reconstruct the way that led to this position. Moreover, it should make the selection of the next step as easy as possible.

2.1.3 Improved orientation through increased comprehensibility

Moreover, it is claimed how “memory for content and memory for spatial information are different aspects of the same mental representation, i.e. the reader’s mental model”. This is said to explain the close correlation between comprehension and memory for location: both orientation difficulties and difficulties in understanding the hypertext are symptoms of the same disease. As such, every feature that facilitates the construction of such a model by reducing mental effort or increases a model’s quality by improving completeness and consistency, affects both comprehension and orientation.

The authors suggest how readability of hyperdocuments, hence also orientation, can be improved by supporting the construction of a mental model in terms of a dual approach based on both increased document coherence and reduced cognitive overhead. Therefore they suggest eight design principles, which will also feature prominently in the MESH framework:

- Typed link labels that allow for understanding semantic relations between information units and reduce fragmentation
- The indication of equivalencies between information units also reduces the impression of fragmentation
- The preservation of the context in which information units are displayed further reduces fragmentation
- Higher-order information units should be available, e.g. composite nodes, to induce a stronger sense of structure
- Visual information about the hypertext structure should be available as overviews, maps, etc.
- The user should be provided with cues about his current position and available navigational options
- Navigation facilities should cover aspects of direction and distance
- A stable screen layout diminishes cognitive overhead

2.2 Advantages of a formal hypermedia data model

Whereas the design principles above already hint at the need for hypertexts to be structured according to a conceptual model, similar to the ones applied in database modeling, other authors support this vision with partially similar and partially complementary arguments.

2.2.1 Consistency

In [Garzotto et al., 1995], consistency is regarded as one of the most important evaluation criteria of hypertext systems: “treat conceptually similar elements in a similar fashion and conceptually different elements differently”.

[Nanard & Nanard, 1995] insist that tools must enable the designer to work both at the abstract model level and at the level of instances. Therefore, they advocate the use of a conceptual model. Abstract semantic types should offer a means for handling an actual structure both at a global and a local level.
This would enforce consistency of the hypertext structure, increase modularity and allow users to recognize similarities. Both node and link types would model similar semantic properties across different entities and enforce the regularity of structure. Moreover, even a few node instances would allow for evaluating the global design and implementation.

2.2.2 Abstractions

[Rivlin et al., 1994] describes the importance of hierarchies and aggregations to navigation. A well-defined hypermedia structure greatly facilitates end user orientation. It has been proven that insight into the underlying abstractions is a key condition to orientation in a hypermedia environment [Halasz, 1988].

[Botafoho et al., 1991] explicitly refer to the object-oriented paradigm as a means for structuring hypertext and providing meaningful abstractions so as to reduce the complexity of large numbers of nodes and links. Therefore, nodes and links are to be collected into more abstract structures, both through aggregation and generalization. This allows for dealing with a set of nodes and links as a single (higher-level) object, which reduces cognitive overhead.

[Garg, 1988] also describes the usefulness of abstractions in hypermedia. They yield richer information structures and more natural specifications of domain knowledge. Also, the expressive power of queries is increased. Moreover, they allow for a whole collection of information units to be denoted by a single reference. Finally, support for collaboration and versioning is facilitated.

In [Mayes, 1994], a distinction is made between hyperspace and conceptual space. The former refers to the hypermedia structure itself, whereas the latter involves the actual concepts and interrelations represented in the hypermedia system. A close correlation between hyperspace and conceptual space is claimed to significantly advance comprehension and orientation. Therefore, the objects in the hypermedia structure are to reflect the concepts from the domain model as accurately as possible.

2.2.3 Typed links

Arguably the most significant abstraction of all, at least in the context of hypermedia, is the link type. The importance of a conceptual data model with typed links to support navigation was already emphasized by [Halasz, 1988]. Whereas [Thüring et al., 1991] stress the influence of typed links upon the coherence of hyperdocuments, [Knopik & Bapat, 1994] advocate the use of both typed nodes and links. They should provide the user with hints at what awaits him in the next node, such that he can make a well-founded decision about his next move. Therefore, they plead for those types not to be “technical” as is the case in many implementations, e.g. implicit versus explicit or internal versus external. Rather, they should reflect the semantic relationship between source and destination node, i.e. as specified in the application domain. Moreover, link typing should not be limited to attaching labels, but should also influence browsing behavior, allow for displaying properties in different contexts, and enforce semantic constraints.

2.2.4 Authoring advantages

The advantages to the author of a formal design model are described in [Garzotto et al., 1993]: first, it improves the communication between analyst, end user and system designer and allows for complex constructs to be discussed on an application domain independent level. Moreover, design methodologies can be tested, analyzed and compared at a high level of abstraction, independently of individual nodes. This permits certain constructs and components to be reused in different applications as well. Furthermore, a formal data model allows for powerful design tools to support authoring in a systematic, structured way. Another very important factor is that it enables these tools to enforce consistency and completeness constraints and predictable representation structures, which in turn will be of benefit to the end user and reduce disorientation.

2.3 E.R. and O.O.-based hypermedia models

The first conceptual hypermedia modeling approaches such as HDM [Garzotto et al., 1993] and RMM [Isakowitz et al., 1995]; [Isakowitz et al., 1998] were based on the entity-relationship paradigm. Object-oriented techniques were mainly applied in hypermedia engines, to model functional behavior of an application's components, e.g. Intermedia [Meyrowitz, 1986]; [Haan et al., 1991], Microcosm [Davis et
al., 1992]; [Hall et al., 1992]; [Beitner et al., 1995]. Hyperform [Wiil & Leggett, 1992]; [Wiil & Leggett, 1997] and Hyperstorm [Bapat et al., 1996]. Along with EORM [Lange, 1994] and OOHDM [Schwabe et al., 1996]; [Schwabe & Rossi, 1998a]; [Schwabe & Rossi, 1998b], MESH is the first approach where modeling of the application domain is fully accomplished through the object-oriented paradigm. The following section presents MESH’s data model in detail.

3 MESH’s object-oriented hypermedia data model

3.1 The basic concepts: node and link types

On a conceptual level, a node is considered a black box, which communicates with the outside world by means of its links. External references are always made to the node as a whole. True to the O.O. information-hiding concept, no direct calls can be made to its multimedia content. However, internally, a node may encode the intelligence to adapt its visualization to the navigation context, as discussed in section 5.

Nodes are assorted in an inheritance hierarchy of node types. Each child node type should be compliant with its parent's definition, but may fine-tune inherited features and add new ones. These features comprise both node layout and node interrelations, abstracted in layout templates and link types respectively.

A layout template is associated with each level in the node typing hierarchy, every template being a refinement of its predecessor. Its exact specifications depend upon the implementation environment, e.g. as to the Web it may be HTML or XML based. Node typing as a basis for layout design allows for uniform behavior, onscreen appearance and link anchors for nodes representing similar real world objects.

A link represents a one-to-one association between two nodes, with both a semantic and a navigational connotation. A directed link offers an access path from its source to its destination node. Links representing similar semantic relationships are assembled into types. Link types are attributed to node types and can be inherited and refined throughout the hierarchy. Link type properties allow for enforcing constraints upon their instances and can be overridden to provide for stronger restrictions upon inheritance. This mechanism is discussed in full in section 3.3.

E.g. whereas an artist node can be linked to any artwork through a has-made link type, an instance of the child node type painter can only be linked to a painting, by means of the more specific child link type has-painted.

3.2 The use of aspects to overcome limitations of a rigid node typing structure

3.2.1 Definition of aspect descriptor and aspect type

The above model is based on a node typing strategy where node classification is total, disjoint and constant. The aspect construct allows for defining additional classification criteria, which are not necessarily subject to these restrictions. Apart from a single “most specific node type”, they allow a node to take part in other secondary classifications that are allowed to change over time. Although we deliberately opted for a single inheritance structure, aspects can provide an elegant solution in many situations that would otherwise call for multiple inheritance.

An aspect descriptor is defined as an attribute whose (discrete) values classify nodes of a given type into respective additional subclasses. In contrast to a node’s “main” subtyping criterion, such aspect...
descriptor should not necessarily be single-valued or constant over time. Aspect descriptor properties denote whether the classification is optional/mandatory, overlapping/disjoint and temporary/permanent.

Each aspect type is associated with a single value of an aspect descriptor. An aspect type defines the properties that are attributed to the class of nodes that carry the corresponding aspect descriptor value. An aspect type's instances, aspects, implement these type-level specifications. Each aspect is inextricably associated with a single node, adding characteristics that describe a specific “aspect” of that node.

A node instance may carry multiple aspects and can be described by as many aspect descriptors as there are additional classifications for its node type. If multiple classifications exist, each aspect descriptor has as many values as there are subclasses to the corresponding specialization. Its cardinalities determine whether the classification is total and/or disjoint. As opposed to node types, aspects are allowed to be volatile. Hence, dynamic classification can be accomplished by manipulating aspect descriptor values, thus adding or removing aspects at run-time. Aspect types attribute the same properties as nodes: link types and layout. However, their instances differ from nodes in that they are not directly referable. An aspect represents the same real-world object as its associated node and can only be visualized as a subordinate of the latter.

E.g. to model an artist that can be skilled in multiple disciplines, a non-disjoint aspect descriptor discipline defines the painter and sculptor aspect types. Discipline-specific node properties are modeled in these aspect types, such that e.g. the Michelangelo node features the combined properties of its Michelangelo.asPainter and Michelangelo.asSculptor aspects.

3.2.2 Delegation of node properties to aspect descriptors
Node type properties (i.e. layout and link types) can be delegated to aspect descriptors, such that they can be inherited and overridden in each aspect type that is associated with one of the descriptor’s values.

An aspect type’s layout template refines layout properties that are delegated to the corresponding aspect descriptor. Link types delegated to an aspect descriptor can be inherited and overridden as well. In addition, each aspect type can define its own supplementary link types. The inheritance/overriding mechanism is similar to the mechanism for supertypes/subtypes, but because an aspect descriptor can be multi-valued, particular care was taken so as to preclude any inconsistencies. Further details are provided in section 3.3.3.

3.2.3 Inheritance of aspect types throughout the classification hierarchy
Aspect types themselves are node type properties that can be inherited and overridden across the node type hierarchy. The aspect descriptor is used as a vehicle for the inheritance of aspect types. This ability yields the opportunity to use aspects as real building blocks for nodes. Link types and layout definitions pertaining to a single “role” a node may have to play, can now be captured into one aspect type. If the corresponding aspect descriptor is attributed at a generic level in the node hierarchy, the aspect type can be inherited where necessary by more specific node types. This allows for the modeling of a similar ‘aspect’ in otherwise completely unrelated node types. E.g. the aspect type art-collector could be defined at root level and inherited by both museum and private-collector, modeling the fact that both node types can behave as owners of artwork. Node types can be ‘assembled’ by inheriting the proper aspect types, complemented by their own particular features. In this way, different aspects
associated with the same node instance can have different editing privileges, such that updating multimedia content can be delegated to different parties.

![Diagram of node instances](image)

### 3.3 Link typing and subtyping

#### 3.3.1 Introduction

In common data modeling literature, subtyping is invariably applied to objects, never to object interrelations. If additional classification of a relationship type is called for, it is instantiated to become an object type, which can of course be the subject of specialization. However, as for a hypermedia environment, node types and link types are two separate components of the data model with very different purposes. It would not be useful to instantiate a link type into a node type, since such nodes would have no content to go along with them and thus each instance would become an ‘empty’ stop during navigation.

This section demonstrates how specialization semantics can be enforced not only upon node types, but also upon the link types. A sub link type will model a type whose set of instances constitutes a subset of its parent’s, and which models a relation that is more specific than the one modeled by the parent.

#### 3.3.2 Definition and domain of a sub link type

A link instance is defined as a source node - destination node tuple \((n_s, n_d)\). Tuples for which this association represents a similar semantic meaning are grouped into link types. A link type defines instances that comply with the properties of the type and is constrained by its domain, its cardinalities and its inverse link type.

The domain of the link type is the data type to which the link type is attributed. This can be either a node type or an aspect type. The domain casts a restriction upon which nodes are valid as source nodes of the tuples represented by the link type:

\[
(n_s, n_d) \in L \Rightarrow n_s \in \text{Dom}(L)
\]

If \(L_c\) is a sub link type resulting from a specialization over \(L_p\), the set of \((n_s, n_d)\) tuples defined by \(L_c\) is a subset of the one defined by \(L_p\). As a consequence, \(L_p\)’s domain should be the same as or define a subset of (i.e. a sub node type or an aspect type) of \(L_p\)’s domain.
A vertical link specialization is the consequence of a parallel classification over the links’ source nodes, in either node subtypes or aspect types. The term denotes that each sub link type is attributed at a ‘lower’, more specific level in the node typing hierarchy than its parent; since $L_c$’s domain is a subset of $L_p$’s domain and they both model similar semantics, the respective associated sets of tuples will be subsets too.

$$L_c \subseteq L_p \Rightarrow ((n_s, n_d) \in L_c \Rightarrow (n_s, n_d) \in L_p) \Rightarrow \text{Dom}(L_c) \subseteq \text{Dom}(L_p)$$

If the domains are the same, we speak of a sub link type resulting from a horizontal specialization. If the sub link type is inherited in a sub node type or aspect type, we speak of a vertical specialization.

### 3.3.3 Cardinalities of a sub link type

A link type’s cardinalities determine the minimum and maximum number of link instances allowed for a given source node. If the domain is an aspect type, the cardinalities pertain to a node’s corresponding aspect.

$$\text{MinCard}(L) = 1 \iff \forall n_s \in \text{Dom}(L): \exists (n_s, n_d) \in L$$
$$\text{MaxCard}(L) = 1 \iff [\forall n_s \in \text{Dom}(L): (n_s, n_d1) \in L \land (n_s, n_d2) \in L \Rightarrow n_d1 = n_d2]$$

Upon overriding link type cardinalities, care should be taken so as not to violate the parent’s constraints, particularly in case of a non-disjoint classification. The following tables present feasible combinations, respectively in function of a horizontal and a vertical specialization over a given parent link type. Note that a ‘-’ sign stands for “not inherited”. Moreover, the mechanism for link type inheritance in a “real” node subtype is similar to delegation to a (1,1) aspect descriptor, as represented in the rightmost column of the vertical link specialization table.
3.3.4 Inverse of a sub link type

The inverse link type is the most specific link type that encompasses all of the original link type’s tuples, with reversed source and destination. There are two possibilities. If the ‘inverse-of’ relationship is mutual, we speak of a particular inverse, notation: \( L \leftrightarrow \text{Inv}(L) \). If this is not the case, we speak of a general inverse, notation: \( L \rightarrow \text{Inv}(L) \).

A particular inverse models a situation where two link types are each other’s inverse. Not counting source and destination’s sequence, the two link types represent the same set of tuples. The term particular inverse is used because no two link types can share the same particular inverse.

\[
L \leftrightarrow \text{Inv}(L) \iff \exists (n_s, n_d) \in L \in \text{Inv}(L) \\
L = \text{Inv}(\text{Inv}(L))
\]

E.g. `employee.is-member-of ↔ department.members`

A child link type can override its parent’s inverse with its own particular inverse, which is to be a subtype of the parent’s inverse.

E.g. `employee.is-manager-of ↔ department.manager`

However, if no suitable particular inverse exists for a given child link type, it has to inherit its parent’s inverse as a general inverse, without overriding. Hence a general inverse can be shared by multiple link types with a common ancestor.

As to a general inverse, the set of tuples represented by \( L \) is a subset of the one represented by \( \text{Inv}(L) \). This is equal to stating that \( L \subset \text{Inv}(\text{Inv}(L)) \).

\[
L \rightarrow \text{Inv}(L) \iff [\exists (n_s, n_d) \in L \Rightarrow (n_d, n_s) \in \text{Inv}(L)], \exists (n_d, n_s) \in \text{Inv}(L): (n_s, n_d) \notin L \\
L \subset \text{Inv}(\text{Inv}(L))
\]

The general inverse of a child link type must be the particular inverse of one of this child’s ancestors.

E.g. `employee.is-manager-of → department.members`

Summarizing, a child link type either inherits its parent’s inverse as a general inverse, or the inverse property is overridden with a subtype of the parent’s inverse becoming the particular inverse of the child link type:

\[
L_c \subset L_p \Rightarrow [(\exists K_c: L_c \leftrightarrow K_c, K_c \subset \text{Inv}(L_p)) \text{ or } (L_c \rightarrow \text{Inv}(L_p))]
\]
3.3.5 Link type specialization properties

Properties for node type specialization determined whether the latter was total and/or disjoint. On the other hand, no such properties are attributed to a link type classification itself. Without a concession to generality, we can force each specialization to be total by defining an additional child link type $L_{cRest}$ where necessary, to collect instances of the parent that could not be classified into any of the other children. Each $(n_s, n_d)$ tuple that belongs to the parent link type $L_p$, also belongs to at least one of its subtypes $L_{ci}$:

$$L_p := L_{c1} \cup L_{c2} \cup L_{c3} \cup L_{c4} \cup \ldots \cup L_{cRest} \iff (\forall L_{ci}: L_{ci} \subset L_p) \land (\forall (n_s, n_d) \in L_p: \exists L_{ci}: (n_s, n_d) \in L_{ci})$$

Whether overlapping subtypes are allowed is not enforced at the specialization level, but as a property of each subtype separately, leaving space for a finer granularity. This is accomplished by a child link type's singularity property, which denotes whether its instances are allowed to also belong to sibling child types. E.g. we can force $L_{cRest}$ to be disjoint to any other child link type by denoting it as a singular link type.

Just like node types can have multiple aspect descriptors, multiple classifications can be defined over a link type. Since each classification should be total, the union of all sub link types described by one such specialization returns the full parent link type. Conversely, each instance of the parent link type should also belong to at least one subtype for each specialization defined.

E.g. department members can be subclassed according to either a person’s function (worker, clerk or manager), or his mode of employment (full-time or part-time). Two sets of sub link types result, any instance of department members will have to be classified according to both criteria:

$$\text{department.members} := \text{department.workers} \cup \text{department.clerks} \cup \text{department.manager}$$
$$\text{department.full-time-members} \cup \text{department.part-time-members}$$

Whereas the data model could stand on its own to support the analysis and design of hypermedia applications, its full potential only becomes apparent in the light of its role as a foundation to the context-based navigation paradigm, briefly discussed in the next section.

4 Mesh’s context-based navigation paradigm

The navigation paradigm as presented in MESH combines set-based navigation principles with the advantages of typed links and a structured data model. The typed links allow for a generalization of the guided tour construct. The latter is defined as a linear structure that eases the burden placed on the reader, hence reducing disorientation.

As opposed to conventional static guided tour implementations, MESH allows for complex structures of nested tours among related nodes to be generated at run-time, depending upon the context of a user’s navigation. Such context is derived from abstract navigational actions, defined as link type selections. Indeed, apart from selecting a single link instance, similarly to the practice in conventional hypermedia, a navigational action may also consist of selecting an entire link type. Selection of a link type $L$ from a given source node $n_s$, results in a guided tour along a set of nodes being generated. This tour includes all nodes that are linked to the given node by the selected link type:

$$n_s.L := \{n_d \mid (n_s, n_d) \in L\}.$$  

E.g. the action Van Gogh has-painted yields a guided tour along all paintings painted by Van Gogh:

$$\text{Van Gogh.has-painted} := \{\text{Iris}, \text{Potato eaters}, \text{Starry night}, \text{Sunflowers}, \text{Wheatfield}, \ldots\}$$

Navigation is defined in two orthogonal dimensions: on the one hand, navigation within the current tour yields linear access to complex webs of nodes related to the user’s current focus of interest. On the
other hand, navigation orthogonal to a current guided tour, changing the context of the user’s information requirements, offers the navigational freedom that is the trademark of hypertext systems. In addition, the abstract navigational actions and tour definitions sustain the generation of very compact overviews and maps of complete navigation sessions. This information can also be bookmarked, i.e. bookmarks not just refer to a single node but to a complete navigational situation, which can be resumed at a later date.

5 A generic application framework

The information content and navigation structure of the nodes are separated and stored independently. The resulting system consists of three types of components: the nodes, the linkbase/repository and the hyperbase engine. Although a platform-independent implementation framework is provided, all actual prototyping is explicitly targeted at a Web environment.

A node can be defined as a static page or a dynamic object, using e.g. HTML or XML. Its internal content is shielded from the outside world by the indirection of link types playing the role of a node’s interface. Optionally, it can be endowed with the intelligence to tune its reaction to the context in which it is accessed. Indeed, by integrating a node type’s set of attributed link types as a parameter in its layout template’s presentation routines, the multimedia objects that are most relevant to this particular link type can be made current upon node access, hence the so-called context-sensitive visualization principle.

Since a node is not specified as a necessarily searchable object, linkage information cannot be embedded within a node’s body. Links, as well as meta data about node types, link types, aspect descriptors and aspects are captured within a searchable linkbase/repository to provide the necessary information pertaining to the underlying hypermedia model, both at design time and at run-time. This repository is implemented in a relational database environment. Only here, references to physical node addresses are stored, these are never to be embedded in a node’s body. All external references are to be made through location independent node ID’s.

The hyperbase engine is conceived as a server-side application that accepts link (type) selections from the current node, retrieves the correct destination node, keeps track of session information and provides facilities for generating maps and overviews. Since all relevant linkage and meta information is stored in the relational DBMS, the hyperbase engine can access this information by means of simple, pre-defined and parameterized database queries, i.e. without the need for searching through node content.

6 Conclusions

6.1 Advantages of the MESH framework with respect to orientation

MESH’s primary ambition was to reduce the orientation problems perceived by the end users of a hypermedia environment. As explained in section 2.1, this can be accomplished by striving towards increased coherence and reduced cognitive overhead.

Evidently, consistency is improved by MESH’s structured data modeling approach, with typed links explicitly representing relationship semantics, i.e. why the source and destination nodes are related, and the practice of node and aspect typing offering maximal facilities for indicating equivalencies between information units. Moreover, end user comprehension of how the hyperbase is conceived and what kind of relationships exist between the distinct nodes, already facilitates orientation to a great extent.

The use of higher-order information units such as object classes, but also the representation of collections of nodes as (node, link type) combinations, induces a stronger sense of structure and decreases cognitive overhead. The latter is even further reduced thanks to the uniform user interface that is supported by inheriting and refining layout templates in the node typing hierarchy and through the practice of capturing layout properties into aspect types as well. Consistent interface properties not only facilitate interaction with the system, but providing a similar layout to similar nodes also increases the user’s ability to grasp the underlying data structure.
Visualizing the hypertext structure is also beneficial to reducing cognitive overhead. The latter is facilitated by two factors; first, the hyperbase structure being stored within a searchable relational database, such that (partial) maps and overviews can be generated at run-time, without the need to search through the node’s contents. Second, the abundance of meta-information as node, aspect and link types allows for incorporating these abstractions into the overview, so as to be able to present concepts of varying granularity. This is applied e.g. in fish eye views, where more distant items can be shown with less detail, i.e. in aggregated form.

Although MESH’s data model could be efficacious on its own merits, it was contrived with specific navigation semantics in mind. The principle of context-based navigation offers a dynamic linear path throughout the information space, diminishing the risk of disorientation, whereas the task of exhaustively exploring a certain topic becomes much easier. The navigation paradigm is easily understandable and supports a certain notion of direction and position.

Whereas context-based navigation mainly affects cognitive overhead, it also has a positive influence on coherence by indicating the context in which information units are displayed, as a representation of what these units have in common. Obviously, this positive effect is even enlarged through the principle of context sensitive node visualization, which causes a node to present that subset of its embedded information that is most relevant to the current context.

6.2 Advantages of the MESH framework with respect to application development and maintenance

Whereas the development advantages of object-oriented analysis and design are too numerous to mention them all, a few topics can be named that particularly apply to the MESH approach. Obviously, the notion of abstraction is strikingly present in all components of the framework. Both layout templates and link types can be designed on a high level of generality, and refined and enriched on more concrete levels through inheritance and overriding. This not only facilitates authoring, but also benefits consistency, hence the overall quality of the application.

The practice of attributing link types to node types, rather than just attributing links to individual nodes, along with the ability of enforcing constraints such as cardinalities and inverse, allows for checking on consistency and referential integrity. Such constraint preservation should not necessarily be “repressive”, but may well be “preventive”, by suggesting mandatory links and feasible destination nodes to the author. Such would be a great asset, especially in larger hypermedia systems.

Other hypermedia approaches such as EORM, RMM, HDM and OOHDM are also based on conceptual modeling abstractions, either through E.R. or O.O techniques. Among these, OOHDM is the only other methodology to explicitly incorporate a subtyping and inheritance/overriding mechanism. However, subtyping modalities are not explicitly stipulated. Rather, they are borrowed from OMT [Rumbaugh et al., 1991], a general-purpose object-oriented design methodology. MESH deploys a proprietary approach, specifically tailored to hypermedia modeling, where structure and relationships prevail over behavior as important modeling factors. Its full O.O. based data modeling paradigm should allow for hypermedia maintenance capabilities equaling their database counterpart; with unique object identifiers, monitoring of integrity, consistency and completeness checking, efficient querying and a clean separation between authoring content and physical hyperbase maintenance. MESH is the only approach to formulate specific rules for inheriting and overriding layout and link type properties, taking into account the added complexity of plural (possibly overlapping and/or temporal) node classifications. Links are treated as first-class objects, with link types being able to be subject to multiple specializations themselves, not necessarily in parallel with node subtyping. It is also clear that a model-based approach in general facilitates information sharing, reuse, development in parallel, etc.

Separation of node content from link structure and meta information allows for the latter two to be stored in a relational database, whereas node content can be maintained in whatever facility is most suitable. Consequently, link maintenance is uncoupled from content maintenance. The former can be carried out almost entirely through queries upon the linkbase, without having to alter the internals of the nodes involved. Links become independent of physical node location and can be created or adjusted without accessing the nodes themselves, resulting in minimal repercussions of updates.
The very loose definition of the node concept allows for an open system where documents of almost any type can be used as nodes and be seamlessly integrated into the system, while retaining full navigational flexibility. Furthermore, nodes can be designed to be sensitive to different types of links, without knowledge of all nodes they are linked to. Only the various link types have to be taken into consideration, not every separate instance. The latter approach can be considered as more natural in that a node does not react to from which node it is accessed, but to the reason why.

Another benefit of abstraction lies in MESH’s navigation paradigm, where navigational actions are specified on an abstract level as much as possible, resulting in link type selection taking the place of link instance selection. This not only facilitates the design, with such actions being specified on node/aspect type level, but also yields node implementations with anchors that are independent of the actual link instance. As a consequence, links can be reallocated without any modification to the source node. Moreover, the context-based navigation paradigm supports a completely automated generation of guided tours, maps and indices, in contrast to e.g. RMM, HDM and OOHDM, where these are conceived as explicit design components, requiring extensive authoring efforts. In MESH, the author is not even engaged in their realization. Finally, it goes without saying that a well-maintained hyperbase will in its turn mean an invaluable asset to the end user.

References


