MESH: an Object-Oriented Approach to Hypermedia Modeling and Navigation

Dr. Wilfried Lemahieu

Author information:

Dr. 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

MESH: an Object-Oriented Approach to Hypermedia Modeling and Navigation

Dr. Wilfried Lemahieu

Abstract

This paper introduces the *MESH* approach to hypermedia modeling and navigation, which aims at relieving the typical drawbacks of poor maintainability and user disorientation. The framework builds upon two fundamental concepts. The *data model* 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 the *context-based navigation paradigm*, conventional navigation along static links is complemented by run-time generated *guided tours*, which are derived dynamically from the context of a user's information requirements. The result is a two-dimensional navigation paradigm, which reconciles complete navigational freedom and flexibility with a measure of linear guidance. These specifications are captured in a high-level, platform independent *implementation framework*.

1 Introduction

The hypermedia paradigm looks upon data as a network of *nodes*, interconnected by *links*. Whereas each node symbolizes a *concept*, a link not only stands for a *relation* between two items, but also explicitly assumes the semantics of a navigation path, hence the quintessential property of *navigational data access*. Their inherent flexibility and freedom of navigation raises hypermedia systems as utterly suitable to support user-driven exploration and learning. Therefore, hypermedia data retrieval embraces a notion of *location*. Data accessibility depends on a user's position in the network, denoted as the *current node* [Lucarella, 1990]. Manipulation of this position gradually reveals links to related information.

Unfortunately, due to inadequacy of the underlying data models, most hypermedia technologies suffer from severely limited maintainability. Moreover, the explorative, non-linear nature of hypermedia navigation imposes a heavy processing load upon the end user, referred to as *cognitive overhead* [Conklin, 1987]. The stringent problem of cognitive overhead effecting into user disorientation and losing one's chain of thought is known as the 'lost in hyperspace' phenomenon [Nielsen, 1990]; [Hammond, 1993]. Disorientation is further increased by the sense of *fragmentation* that is induced by scattering information over numerous separate nodes [Thüring et al., 1995].

This paper overviews the *MESH* hypermedia framework as deployed in [Lemahieu, 1999], which 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*. The text is partitioned according to *MESH*'s fundamental concepts. To start with, the *object-oriented hypermedia data model* is portrayed. The next section is dedicated to the *context-sensitive navigation paradigm*. A subsequent section translates these blueprints into a high-level *implementation framework*, specified in an abstract and platform independent manner. A last section makes comparisons to related work and formulates conclusions.

2 An object-oriented hypermedia data model

2.1 Introduction

The benefits of data modeling abstractions to both orientation and maintainability were already acknowledged in [Halasz, 1988]. They yield richer domain knowledge specifications and more expressive querying. Typed nodes and links offer increased consistency in both node layout and link structure [Thüring et al., 1991]; [Knopik & Bapat, 1994]. Higher-order information units and perceivable equivalencies (both on a conceptual and a layout level) greatly improve orientation [Thüring et al., 1995]; [Ginige et al., 1995]. Semantic constraints and consistency can be enforced [Garzotto et al., 1995]; [Ashman et al., 1997], tool-based development is facilitated and reuse is encouraged [Nanard & Nanard, 1995].

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*. Along with the *Tower model* [De Bra et al., 1992], *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. Its 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 object-oriented modeling abstractions [Rumbaugh et al., 1991]; [Jacobson et al., 1992]; [Meyer, 1997]; [Snoeck et al., 1999] are coupled to proprietary concepts to provide for a *formal hypermedia data model*.

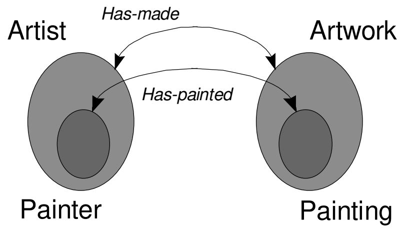
2.2 The basic concepts: node types, layout templates and link types

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 4. 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 such as *domain*, *cardinalities* and *destination/inverse*[1] allow for enforcing constraints upon their instances. These properties can be overridden to provide for stronger restrictions upon inheritance. 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***.

[1] As discussed in detail in [Lemahieu, 1999], a link type's *destination* is a derived property, defined as the *inverse link type's domain*.



2.3 The use of aspect descriptors and aspect types to overcome limitations of a rigid node typing structure

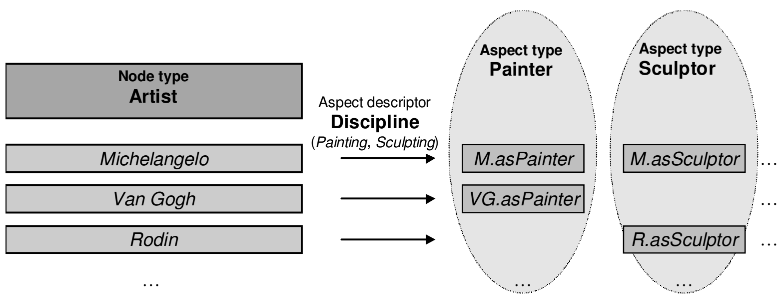
The above model is based on a node typing strategy where node classification is total, disjoint and constant. The aspect construct allows for defining *additional* classification criteria, which are not necessarily subject to these restrictions. Apart from a single "most specific node type", they allow a node to take part in other secondary classifications that are permitted to change over time.

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* nor *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.



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. 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[1].

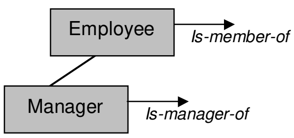
2.4 Link typing and subtyping

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.

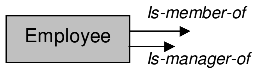
A link instance is defined as a source node - destination node tuple (n_s, n_d). Tuples for which this association represents a similar semantic meaning are grouped into link types. A link type defines instances that comply with the properties of the type and is constrained by its *domain*, its *cardinalities*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

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. Such specialization is called *vertical* if it is the consequence of a parallel classification over the link types' *domain*, denoting that the sub link type is attributed at a 'lower', more specific level in the node typing hierarchy than its parent. If L_c and L_p share the same domain, L_c can still define a subtype of L_p in the case where L_c models a more restricted, more specific kind of relationship than L_p, independently of any node specialization. Both parent and child link type are attributed at the same level in the node type hierarchy, hence the term *horizontal* specialization.

E.g.



Vertical **link specialization**



Horizontal **link specialization**

Apart from the domain, a link type's cardinalities and inverse can be overridden as well upon specialization. *MESH* presents a formal overriding mechanism, wherein particular care is taken so as not to violate the parent's constraints, particularly in case of a non-disjoint classification.

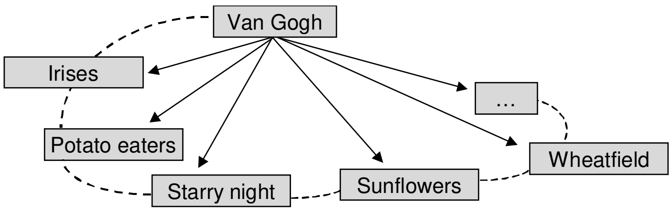
3 The context-based navigation paradigm

3.1 Linearity and guided tours

To highlight the advantages of hypermedia navigation, comparisons are often made to books. Books are said to be *linear* information systems; their pages are organized uni-dimensionally, in a fixed order. Hypertext offers the possibility to break through this linear constraint and organize data in more complex structures, to be accessed following different possible paths, depending on the user's preferences and interests. Cognitive overhead, however, is significantly lower in a linear structure, be it at the cost of navigational freedom. Linearity provides a leading thread that facilitates orientation

[1] We deliberately opted for a single inheritance structure, however, aspects can provide an elegant solution in many situations that would otherwise call for multiple inheritance, much like *interfaces* in the *Java* language. See [Lemahieu, 1999] for further details.

and prevents the reader from getting lost [Jonassen, 1990]. The latter is acknowledged in [Trigg, 1988]; [Nielsen, 1990], where linearity is re-introduced in so-called *guided tours*, chaining together all nodes pertaining to a common subject with *forward/backward* links. E.g. the typical hypermedia links (represented as arrows) between **Van Gogh** and each of his **paintings** can be complemented by a *guided tour* (represented as dotted lines) along these **paintings**.



Unfortunately, such hard-coded guided tours have proven to be inflexible and difficult to maintain. Moreover, they introduce a measure of redundancy into the hyperbase, as a guided tour typically reflects a communal property among its participating nodes. However, the property of '*being painted by the same artist*' is already established within the respective links from each **painting** to its **artist**. Thus, it would be possible to infer this knowledge and generate such guided tour *at run-time*, without burdening hyperbase maintainability.

MESH builds upon its data model to reconcile navigational freedom with the ease of linear navigation. Its 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.

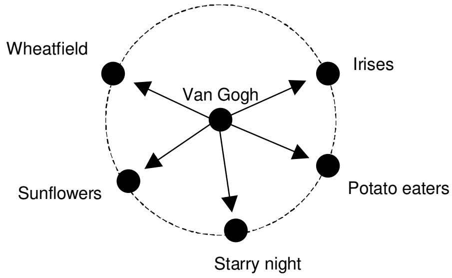
3.2 A guided tour as derived from the current context

In conventional hypermedia applications, the *current node* is the only variable that determines which information is accessible at a given moment; navigation is only possible to nodes that are linked to this current node. Its value changes with each navigation step as it represents the immediate focus of the user's attention. *MESH* introduces the *current context* as a second, longer-term variable that 'glues' the various visited nodes together and provides a background about which common theme is being explored. The current context is defined as the combination of a *context node* and a *context link type*. The context node represents the subject around which the user's broader information requirements 'circle'. The nature of the relationship involved is depicted by the context link type.

A guided tour derives from the current context. Therefore, *MESH* discriminates between *direct* and *indirect* links. A direct link represents a lasting relation between two nodes. Direct links are typed and reflect the underlying conceptual data model. Because they are permanent and context-independent, they are stored explicitly into the hyperbase and are always valid. E.g. the node **Sunflowers** is directly linked to the **Van Gogh** node.

An *indirect* link between two nodes indicates that they share relevancy to a common third node. The latter denotes the *context* within which the indirect link is valid. As indirect links not only reflect the data model, but also depend on a run-time variable, the *current context*, they cannot be stored within the hyperbase. They are to be created *dynamically* at run-time, as inferred from a particular context. E.g. an indirect link between **Sunflowers** and **Wheatfield** is only relevant when exploring information related to **Van Gogh**.

A *guided tour* is defined as a path of *indirect* links along all nodes relevant to the current context. These nodes are directly linked to the context node (through instances of the context link type) and indirectly to their predecessor and successor in the tour. As they are chained into a linear structure, a logical order should be devised in which the subsequent tour nodes can be presented to the user. The most obvious criterion is in alphabetical order of a *node descriptor* field. More powerful alternatives are discussed in [Lemahieu, 1999]. E.g. the context **Van Gogh*.has-painted*** yields a guided tour among the nodes {**Irises**, **Potato eaters**, **Starry night**, **Sunflowers, Wheatfield,** …} with **Van Gogh** as the *context node* and ***has-painted*** as the *context link type*.

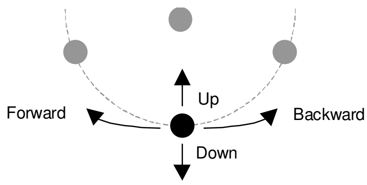


Current context: **Van Gogh.*has-painted***

Note that the discrepancy between *guided tour* and *context* can be compared to the traditional duality in representing a circle either through the points on its *circumference*, or through its *center* and a *radius*. Guided tours are not stored within the hyperbase as *an enumeration of participating nodes*, but are calculated at run-time from the *current context*. Although sequential by nature, such tours do not restrict the user's navigational freedom, as long as sufficient flexibility is offered in choosing which tour to follow. The linearity lies in 'following' the tour. The freedom lies in starting one.

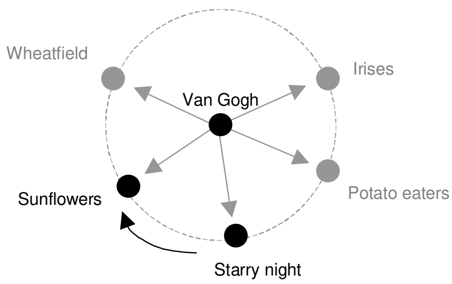
3.3 Navigational actions

Navigational actions can be classified according to two dimensions. First, there is moving *forward* and *backward* within the current tour, along indirect links. Second, and orthogonal to this, there is the option of moving *up* or *down* along direct links, closer to or further away from the session's starting point. Additionally, one can distinguish between actions that change the current context and actions that only influence the current node.



3.3.1 Moving forward/backward within the current tour

Moving forward or backward in a guided tour along indirect links, results in the node following/preceding the current node being accessed to become the new current node. The current context is unaffected.



Current context: **Van Gogh.*has-painted***
Current node: **Starry night ⇒ Sunflowers**

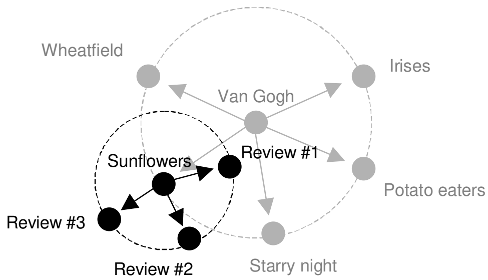
3.3.2 Moving up/down

Moving down implies an action of ‘digging deeper’ into the subject matter, moving away from the starting point. This is accomplished through selection from the current node of either a direct link type or link instance. In the case of a *unique* destination node, the result is the latter node being accessed. In the case of a *set* of destination nodes, the outcome is a new 'nested' tour being started.

In complete analogy to traditional hypermedia navigation, selection of a link instance l from a given source node n_s results in its unique destination node n_d being accessed: $n_s.l := \{n_d \mid l = (n_s, n_d)\}$. E.g. selection of the link (**Sunflowers, National Gallery**) from the current node **Sunflowers**, induces an access to the node **National Gallery**.

However, *MESH* aggregating single *link instances* into *link types*, yields the opportunity of anchoring and consequently selecting a *complete link type* from a given source node. Selection of a link type L from a source node n_s yields a set of all destination nodes n_d of tuples representing link instances of L with n_s as the source node, i.e. all nodes that are linked to the current node by the selected link type: $n_s.L := \{n_d \mid (n_s, n_d) \in L\}$. Depending on maximum cardinality of the link type, the resulting set may contain multiple destination nodes. E.g. with **Sunflowers** as the current node, selection of the link type ***reviews*** generates a *collection* of nodes to-be-accessed: **Sunflowers.*reviews*** := {**review#1**, **review#2**, **review#3**, …}.

The result of such action is a *context change*: a new context emanates, resulting in new indirect links. A new tour is generated, nested within the former, according to this new context. The current node **Sunflowers** is denoted as the new context node. The non-unique link type ***reviews*** defines the context link type, which yields a new *nested* tour: **Sunflowers.*reviews***. The first review is accessed to become the new current node. Such *context change* reflects the user’s decision to concentrate on the current node as a new topic of interest. All indirect links are destroyed and redefined around this new context.



Current context: **Van Gogh.*has-painted* ⇒ Sunflowers.*reviews***
Current node: **Sunflowers ⇒ Review#1**

Hence contexts, and consequently guided tours, can exist in layers. As such, it is possible to ‘delve’ into a subject and have multiple *open* tours, nested within one another, where the context node of one tour is the current node of the tour it is nested in. Navigation along indirect links is invariably carried out within the “deepest”, i.e. most recently started tour. Continuing a tour on a higher level is only possible if all tours on a lower level have been either completed or disbanded. This is accomplished by *moving up*, which reverses the latest *move down* action. If the latter involved a context change, the move up action results in the reestablishment of the previous context and the cancellation of the tour generated through this most recent link type selection. The previous context’s context node and indirect links are restored. The most recent context node (**Sunflowers** in the example) again becomes the current node.

The practice of node and link typing allows for casting navigational actions to a whole *class* of nodes, regardless of the actual instance they are applied to. Hereby, selections of link *types* that exist at a sufficiently high level of abstraction can be imposed upon every single node belonging to a tour. E.g. in the context of **Van Gogh.*has-painted***, a **painting**#x.***reviews*** selection can be issued once on *tour* level, with additional (nested) tours being generated automatically for each node participating in the **Van Gogh.*has-painted*** tour. If these tours in their turn include navigational actions on type level, a complex navigation pattern results, which can be several levels deep. Again, *forward* and *backward* links always apply to the current tour, i.e. to the open tour at the 'deepest' level. In addition, the abstract navigational actions and tour definitions sustain the generation of very compact tree-shaped overviews and maps of complete navigation sessions[1]. In this respect, the *move up* and *move down* actions indeed correspond to moving up or down in the graph. The represented 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 time.

4 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*. In [Lemahieu, 1999], a platform-independent implementation framework was provided, but all subsequent 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 by integrating the node type's set of attributed link types as a parameter in its layout template's presentation routines, hence the so-called *context-sensitive visualization* principle. This allows for different views to be defined over the same information, much like the *city* construct in the *Tower* model.

Since a node is not specified as a necessarily searchable object, linkage information cannot be embedded in 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 script 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*.

5 Evaluation and conclusions

5.1 Data modeling and authoring

Ohter 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 methodology to 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*

[1] See [Lemahieu, 1999] for further details.

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. Authoring is greatly facilitated by O.O. features such as inheritance and overriding, class properties and layout templates that allow for high-level specification and lower-level refinement of node properties. Links can be anchored on *type* level, independently of actual node and link instances. Semantics attributed within the data model permit the automated type checking and integrity constraints we have grown accustomed to in a database environment. Dangling links and inconsistent link attributions can already be detected during the design phase. Finally, it is clear that a model-based approach in general facilitates information sharing, reuse, development in parallel, etc.

5.2 Navigation and orientation

Apart from the obvious benefit of a well-maintained hyperbase, *typed links* should permit a better comprehension of the semantic relations between information objects. The use of *higher-order* information units and the representation of collections of nodes as (source node, link type) combinations, induces a stronger sense of structure. A *node typing hierarchy* with consistent layout and user interface features, reflecting similarities between nodes, is to reduce both cognitive overhead and the impression of fragmentation. The *context* concept, as a representation of what various nodes have in common, will diminish fragmentation, but is to remedy cognitive overhead as well, as the linear guided tours improve a user's sense of position and his ability to ascertain his navigational options. Through the specification of navigational actions *on tour level*, complex navigation patterns can be applied to all nodes in a tour without additional effort. A further decrease in fragmentation and cognitive overhead is obtained by making node *visualization* dependent upon the context in which a node is accessed. The abundance of meta-information as node, aspect and link types allows for enriching *maps* and *overviews* with concepts of varying granularity. A final benefit is the ability to *bookmark a complete navigational situation* in an utterly compact manner, with the possibility of it being resumed later on, from the exact point where it was left.

EORM, *RMM*, *HDM* and *OOHDM* also feature specific topologies such as *guided tours*, *indexes* etc. A fundamental difference is that these are conceived as explicit *design components*, requiring author input for query definitions, node collections and forward/backward links. In *MESH*, neither guided tours nor indexes require any maintenance nor design effort, as the author is not even engaged in their realization. They are generated at run-time upon user request, not to restrict his freedom, but merely to facilitate navigation and support orientation.

References

[Ashman et al., 1997] H. Ashman, A. Garrido and H. Oinas-Kukkonen, Hand-made and Computed Links, Precomputed and Dynamic Links, *Proceedings of Hypertext - Information Retrieval - Multimedia (HIM '97)*, Dortmund (Sep. 1997)

[Conklin, 1987] J. Conklin, Hypertext: An Introduction and Survey, *IEEE Computer Vol. 20*, No. 9 (Sep. 1987)

[De Bra et al., 1992] P. De Bra, G. Houben and Y. Kornatzky, An Extensible Data Model for Hyperdocuments, *Proceedings of the fourth ACM European Conference on Hypermedia Technology (ECHT '92)*, Milan (Dec. 1992)

[Garzotto et al., 1993] F. Garzotto, P. Paolini, and D. Schwabe, HDM - A Model-Based Approach to Hypertext Application Design, *ACM Trans. Inf. Syst. Vol. 11*, No. 1 (Jan. 1993)

[Garzotto et al., 1995] F. Garzotto, L. Mainetti and P. Paolini, Hypermedia Design, Analysis, and Evaluation Issues, *Commun. ACM Vol. 38*, No. 8 (Aug. 1995)

[Ginige et al., 1995] A. Ginige, D. Lowe and J. Robertson, Hypermedia Authoring*, IEEE Multimedia Vol. 2*, No. 4 (Winter 1995)

[Halasz, 1988] F. Halasz, Reflections on NoteCards: Seven Issues for Next Generation Hypermedia Systems, *Commun. ACM Vol. 31*, No. 7 (Jul. 1988)

[Hammond, 1993] N. Hammond, Learning with Hypertext: Problems, principles and Prospects, *HYPERTEXT a psychological perspective*, C. McKnight, A. Dillon and J. Richardson Eds., *Ellis Horwood*, New York (1993)

[Isakowitz et al., 1995] T. Isakowitz, E. Stohr and P. Balasubramanian, RMM, A methodology for structured hypermedia design, *Commun. ACM Vol. 38*, No. 8 (Aug. 1995)

[Isakowitz et al., 1998] T. Isakowitz, A. Kamis and M. Koufaris, The Extended RMM Methodology for Web Publishing, *Working Paper IS-98-18, Center for Research on Information Systems*, 1998 (Currently under review at ACM Trans. Inf. Syst.)

[Jacobson et al., 1992] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, Object-Oriented Software Engineering, *Addison-Wesley*, New York (1992)

[Jonassen, 1990] D. Jonassen, Semantic net elicitation: tools for structuring hypertext, *Hypertext: State of the Art*, R. McAleese and C. Green Eds., *Intellect*, Oxford (1990)

[Knopik & Bapat, 1994] T. Knopik and A. Bapat, The Role of Node and Link Types in Open Hypermedia Systems, *Proceedings of the sixth ACM European Conference on Hypermedia Technology (ECHT '94)*, Edinburgh (Sep. 1994)

[Lange, 1994] D. Lange, An Object-Oriented design method for hypermedia information systems, *Proceedings of the twenty-seventh Hawaii International Conference on System Sciences (HICSS-27)*, Hawaii (Jan. 1994)

[Lemahieu, 1999] W. Lemahieu, Improved Navigation and Maintenance through an Object-Oriented Approach to Hypermedia Modelling, *Doctoral dissertation (unpublished)*, Leuven (Jul. 1999)

[Lucarella, 1990] D. Lucarella, A Model For Hypertext-Based Information Retrieval, *Proceedings of the European Conference on Hypertext*, Versailles (Nov. 1990)

[Meyer, 1997] B. Meyer, Object-Oriented Software Construction, Second Edition, *Prentice Hall Professional Technical Reference*, Santa Barbara (1997)

[Nanard & Nanard, 1995] J. Nanard and M. Nanard, Hypertext Design Environments and the Hypertext Design Process, *Commun. ACM Vol. 38*, No. 8 (Aug. 1995)

[Nielsen, 1990] J. Nielsen, The Art of Navigating Through Hypertext, *Commun. ACM Vol. 33*, No. 3 (Mar. 1990)

[Rumbaugh et al., 1991] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, Object Oriented Modelling and Design, *Prentice Hall*, Englewood Cliffs (1991)

[Schwabe et al., 1996] D. Schwabe, G. Rossi and S. Barbosa, Systematic Hypermedia Application Design with OOHDM, *Proceedings of the seventh ACM conference on hypertext (Hypertext '96)*, Washington DC (Mar. 1996)

[Schwabe & Rossi, 1998a] D. Schwabe and G. Rossi, Developing Hypermedia Applications using OOHDM, *Proceedings of the ninth ACM Conference on Hypertext (Hypertext '98)*, Pittsburgh (Jun. 1998)

[Schwabe & Rossi, 1998b] D. Schwabe and G. Rossi, An O.O. approach to web-based application design, *Draft* (1998)

[Snoeck et al., 1999] M. Snoeck, G. Dedene, M. Verhelst and A. Depuydt, Object-Oriented Enterprise modeling with MERODE, *Universitaire Pers Leuven*, Leuven (1999)

[Thüring et al., 1991] M. Thüring, J. Haake, and J. Hannemann: What's ELIZA doing in the Chinese Room - Incoherent Hyperdocuments and how to Avoid them, *Proceedings of the third ACM Conference on Hypertext (Hypertext '91)*, San Antonio (Nov. 1991)

[Thüring et al., 1995] M. Thüring, J. Hannemann and J. Haake: Hypermedia and Cognition: Designing for comprehension, *Commun. ACM Vol. 38*, No. 8 (Aug. 1995)

[Trigg, 1988] R. Trigg, Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment, *ACM Trans. Office Inf. Syst. Vol. 6*, No. 4 (Oct. 1988)