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 runtime 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, 1999a], 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, an extended version of [Lemahieu, 1999b], 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, e.g. Microcosm [Davis et al., 1992]; [Beitner et al., 1995], Hyperform [Wiil & Leggett, 1997] and Hyperstorm [Bapat et al., 1996]. Along with the Tower model [De Bra et al., 1992], EORM [Lange, 1994] and OO-HDM [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.

MESH's 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. 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 data model provides for a firm hyperbase structure and an abundance of meta-information that facilitates implementation of an enhanced navigation paradigm.

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 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.

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

2.3.1 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.

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

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

\[\text{2 We deliberately opted for a single inheritance structure, however, aspects can provide an elegant solution in many situations that would otherwise call for multiple inheritance, much like interfaces in the Java language. See [Lemahieu, 1999a] for further details.}\]
modeled in these aspect types, such that e.g. the Michelangelo node features the combined properties of its Michelangelo.asPainter and Michelangelo.asSculptor aspects.

2.3.2 Aspect types as node type building blocks

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.

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. Node types can be ‘assembled’ by inheriting the proper aspect types, complemented by their own particular features. Hereby, different aspects associated with the same node instance can have different editing privileges, such that updating multimedia content can be delegated to different parties.

2.4 Link typing and subtyping

2.4.1 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.

2.4.2 Definition and domain of a sub link type

---

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

2.4.3 Overriding link type properties

Apart from the domain, a link type’s cardinalities and inverse can be overridden as well upon specialization. The cardinalities determine the minimum and maximum number of link instances allowed for a given source node. 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. For further details we refer to [Lemahieu, 1999a].

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, e.g. employee.is-member-of \(\leftrightarrow\) department.members. The term particular inverse is used because no two link types can share the same particular inverse.

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: employee.is-manager-of \(\leftrightarrow\) 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, e.g. employee.is-manager-of \(\rightarrow\) department.members and employee.is-clerk-of \(\rightarrow\) department.members.

Link types are deemed extremely important, as they not only enforce semantic constraints but also interface between nodes, such that these can be coded and updated independently of one another. Moreover, they provide the basis for context-sensitive node visualization, as discussed in section 4.
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 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, 1999a]. 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.

![Diagram of a guided tour](image)

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.

![Diagram of navigational actions](image)

**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.
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: 
\[
{n_s} = \{ n_d \mid l = (n_s, n_d) \}
\]
E.g. selection of the link (Sunflowers, National Gallery) from the current node Sunflowers, induces an access to the node National Gallery; Sunflowers.(Sunflowers, National Gallery) := \{ 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, \ldots \}.

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.

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.
The latter 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.

\section{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, 1999a], 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. Upon activation, a node is provided with the link type by which it was accessed. Consequently, the multimedia objects that are most relevant to this particular link type can be made current, 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.

\footnote{See [Lemahieu, 1999a] for further details.}
5 Conclusions

5.1 Data modeling and authoring

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 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 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. Node design is further enhanced by links and layout properties being automatically suggested. For that purpose, a design tool can retrieve the necessary information from the meta knowledge stored within the hyperbase. 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.

Set-based hypermedia paradigms such as CHM [Duval et al., 1995a]; [Duval et al., 1995b], the HM-Data Model [Maurer & Scherbakov, 1992]; [Srinivasan, 1995] and Hyper-G [Andrews et al., 1995a]; [Andrews et al., 1995b] equally provide inherent support for navigation in two orthogonal planes; inside a collection and across collection boundaries. Hereby, their current container and current member concepts are comparable to MESH's current context and current node respectively. A drawback, however, is that they do severely limit navigational freedom. Moreover, they lack a firm underlying data model with typed node interrelations. Likewise, the opportunity of defining abstract navigational actions on tour level is a feature that is exclusive to MESH.

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.

5.3 Ongoing research issues

In the current stage of development, data modeling and navigation have been favored over internal node design. Indeed, each object from the conceptual domain model is to be translated to an active node, which is responsible for its own context-dependent visualization and interaction with the user. In this respect, node types can be seen as tower objects [De Bra et al., 1992], with a node in itself bearing a description on different levels, e.g. structural, semantic, presentational etc. For that purpose, ongoing research is explicitly targeted at a Web environment. As to the latter, the XML standard could be of utter importance.

A related issue is the support for continuous media types. Media type specific manipulations are currently considered to be internal node properties, beyond the scope of the model. However, synchronization of multiple audio and video tracks requires facilities for inter node and intra node timing constraints [Hardman et al., 1994]. These cannot be enforced in MESH at present. Moreover, the constraint of only a single current node at a time may be too restrictive, as suggested in [Hardman et al., 1993]; [De Bra et al., 1994]. Hereby, MESH's context node concept can prove to be a valuable asset to allow for continuous media objects to keep playing “in the background”, while other related nodes are visited in the same context.

References


