

Context-based navigation in the Web by means of dynamically generated guided tours

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

KEYWORDS

Web navigation, guided tours, hypermedia, object-orientation

ABSTRACT

A key advantage of hypermedia systems such as the Web is that the user is able to navigate through the information space in a *non-linear* fashion. He can explore the interlinked documents according to his own interests and insights, instead of being confined to the rigid, linear structure of e.g. pages in a book. A downside, however, is that this navigational freedom entails the risk of *disorientation*, especially in a gigantic hypertext such as the Web. This paper presents a *context-based navigation paradigm* for the Web and reconciles navigational freedom with a measure of linear guidance to prevent the user from becoming disoriented. For that purpose, “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.

1. INTRODUCTION

The *MESH* hypermedia framework as deployed in [31] presents a structured approach to both data modeling and navigation, so as to overcome two crucial problems in the field: *poor maintainability and user disorientation*. *MESH* is an acronym for *Maintainable, End user friendly, Structured Hypermedia* and builds upon an *object-oriented data model* and a *context-based navigation paradigm*. This paper discusses *MESH*’s navigation paradigm and applies it to a Web environment. First, the predicaments surrounding hypermedia navigation are portrayed. Next comes a brief overview of *MESH*’s data model. The body of the paper is dedicated to the context-based navigation paradigm, which relies heavily on the abstractions made in the data model. Thereafter, a platform-independent implementation framework is proposed, which is then applied to a Web environment. A last section makes comparisons to related work and formulates conclusions.

2. HYPERMEDIA NAVIGATION AND COGNITION

2.1 Navigational data access

Hypermedia systems stand out amid other information systems in that data access is accomplished through *navigation*, rather than querying. Their appeal is based upon the ability to store complex, cross-referenced bodies of information as a network of *nodes* and *links*, which can be explored according to the user's personal preferences. 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* [33]. Manipulation of this position gradually reveals links to related information. The environment that really brought hypermedia to the public eye is undoubtedly the *World Wide Web* [4], promoting the hypertext paradigm as the primary access mode to all nodes, implemented as *HTML pages*, on Internet-connected networks around the world.

As initially pointed out in [22], efficient hypermedia data retrieval requires a search mechanism to complement navigational access. Therefore, query-based applications (e.g. search engines) have established themselves in the *Web* as well. However, *navigation* remains an utterly valuable asset to further explore the available information, once an initial *starting point* has been obtained by means of a query.

2.2 Cognitive overhead and user disorientation

As a downside, the explorative, non-linear nature of hypermedia navigation imposes a heavy processing load on the end user, referred to as *cognitive overhead* [14], [44] particularly in such an immense environment as the *Web*. 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 [23], [42].

The above has been thoroughly discussed in literature, e.g. [5], [13], [43], [47]. Interestingly, [36] distinguishes 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. In addition, [51] formulates design principles such as the use of typed links, higher-order information units, a stable screen layout and visual cues, to *increase document coherence* and *reduce cognitive overhead*. These principles are realized in *MESH's* data model, which is briefly reviewed in section 3.2.

With respect to navigation, the same authors suggest 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. Moreover, navigation facilities are to cover aspects of *direction and distance* and should offer the reader maximal support to identify his *current position* within the hypermedia structure. Also, *selection of the next navigation step* should be made as easy as possible. The latter principles provide the fundamentals to *MESH's* navigation paradigm. Another key ingredient is the notion of *guided tours*, as introduced below.

2.3 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 [28]. The latter is acknowledged in [43], [52] where linearity is re-introduced in so-called *guided tours*, chaining together all nodes pertaining to a common subject with *forward/backward* links.

For instance 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** (figure 1).

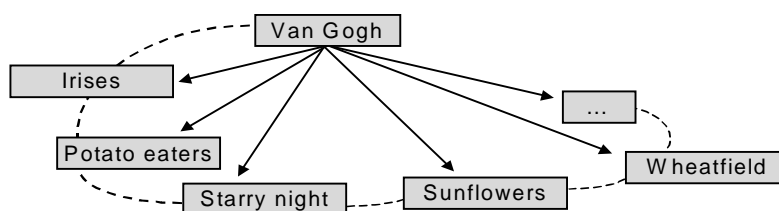


Figure 1: Example of a guided tour

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.

The latter is exploited in *MESH’s* navigation paradigm, referred to as *context-based navigation*, which reconciles navigational freedom with the ease of linear navigation. In this manner, the concept of *at run-time generated guided tours* offers a linear path throughout (part of) the hyperbase to reduce cognitive overhead and user disorientation.

3. THE MESH HYPERMEDIA FRAMEWORK

3.1 Introduction

Although this paper primarily focuses on *MESH’s* *navigation paradigm*, a basic understanding of its underlying data modeling primitives is indispensable. Therefore, this section briefly discusses *MESH’s* data model, wherein elements from both *entity-relationship* [12] and *object-oriented* [39], [48] modeling, along with proprietary hypermedia concepts, are combined into a framework that should improve both *maintainability* and end user *orientation*. A more thorough discussion of *MESH’s* data model can be found in [32].

3.2 The basic concepts: node types and link types

The benefits of data modeling abstractions to both orientation and maintainability were already acknowledged in [22]. They yield richer domain knowledge specifications and more expressive querying. Typed nodes and links offer increased consistency in both node layout and link structure [49]. Higher-order information units and perceivable equivalencies (both on a conceptual and a layout level) greatly improve orientation [51]. Semantic constraints and consistency can be enforced [2], tool-based development is facilitated and reuse is encouraged [41]. The first conceptual hypermedia modeling

approaches such as *HDM* [21] and *RMM* [24], [25] 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* [15], *Hyperform* [53] and *Hyperstorm* [3]. Along with *OOHDM* [45], [46] and *WebML* [11], *MESH* is the first approach where modeling of the *application domain* is fully accomplished through the object-oriented paradigm.

As with any hypermedia model, *MESH*'s basic building blocks are *nodes* and *links*. However, in *MESH*, these concepts are abstracted into types, which explicitly take on the semantics of *objects* and *relationships* in an object-oriented conceptual data model. Because of this, each node corresponds to a real world concept hence navigation in *MESH* closely matches navigating conceptual space as defined in [36]. *MESH* separates the inter-node data modeling aspect from intra-node design (see also section 5.2). Therefore on a conceptual level, a node is considered a black box, i.e. the data model makes abstraction of a node's internal *content*. The latter depends on the actual implementation environment. As to a Web-based implementation, each node corresponds to an HTML or XML page.

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 two concepts: node *layout* and node *interrelations*, abstracted in *layout templates* and *link types* respectively. Whereas link types are well defined at the conceptual level, a node's layout template will again depend upon the actual implementation environment, e.g. as to the Web it may be defined as a DTD or an XML schema. With regard to node layout, we will suffice by stating that with any level in the node typing hierarchy, a template can be associated, where each template is a refinement of its immediate ancestor. Node typing as a basis for layout design allows for uniform behavior and onscreen appearance for nodes representing similar real world objects.

So as to characterize node interaction, each node type is equipped with a set of *link types*. These classify the links according to the semantic interpretation of the relations they embody. The properties of a link type are its *domain*, *minimum cardinality* (optional/mandatory), *maximum cardinality* (unique/non-unique) and *inverse link type* (there is also the *destination*, which is a derived property defined as the *inverse link type's domain*). Semantics attributed within the data model permit automated type checking and integrity constraints. Moreover, node design is greatly facilitated by the abstract-level node and link type definitions, cardinality constraints and layout templates. These can be inherited and refined, but their constraints may never be weaker than the ones imposed at a higher level so as to preserve consistency. For example 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* (figure 2).

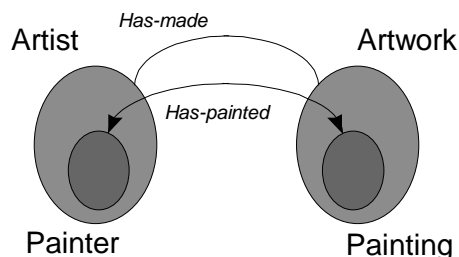


Figure 2: Link type inheritance and overriding

3.3 Aspect descriptors and aspect types

Whereas the data model as depicted thus far is inherently *static*, with each node being an instance of exactly one “most specific type”, which remains unchanged during the whole of the node's lifetime, the *aspect* construct offers a means to overcome modeling restrictions imposed by a rigid subtyping

hierarchy. *Aspect descriptors* provide for additional node classification criteria such that affiliated properties (i.e. both specific layout and link types) can be bundled into *aspects*, attributed to *temporary* and *non-disjoint* sets of nodes. Aspects are defined as instances of *aspect types*, with each aspect type corresponding to a particular value for an aspect descriptor, hence defining a subclass according to the classification criterion defined by the aspect descriptor. An aspect descriptor's properties denote whether such classification is *optional/mandatory*, *overlapping/disjoint* and *temporary/permanent*. 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 to a node at run-time. In this way, apart from a single "most specific node type", aspects allow a node to take part in other secondary classifications that are allowed to change over time. For instance 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 (figure 3). Note that an aspect represents the *same real-world object* as its associated node and can only be visualized as a subordinate of the latter.

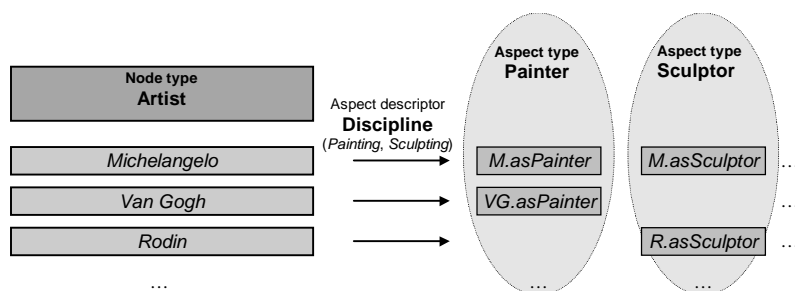


Figure 3: Aspect descriptors and aspect types

3.4 Link subtyping

A final important issue with regard to the data model is the concept of *link 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. Therefore, *MESH* considers link types as full-fledged abstract types that can equally be subject to subtyping. 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. Moreover, its properties (i.e. domain, minimum cardinality, maximum cardinality and destination/inverse) can be overridden to provide for more specific semantic constraints. If a child link type overrides its parent's domain, we speak of *vertical* link specialization: it is the consequence of a parallel specialization over the link type's *domain*, denoting that the sub link type is attributed at a 'lower', more specific level in the node typing hierarchy than its parent. If the domain is *not* overridden, we speak of *horizontal* link specialization: both parent and child link type are attributed at the same level in the node type hierarchy, but the child models a more restricted, more specific kind of relationship than the parent. Examples of both horizontal and vertical link subtyping are presented in figure 4: if the **manager** node type is a subtype of **employee**, the *is-member-of* link type leading to the employee's **department** can be inherited and overridden in **manager**, to become the *is-manager-of* link type, which obviously entails a more specific semantic meaning than

the parent link type. As the domain of *is-manager-of* is a subtype of *is-member-of*, the specialization is *vertical*. However, even if the child node type **manager** does not exist, the link type *is-manager-of* can be defined as a subtype of *is-member-of*. Both parent and child link type have the same domain, hence the link specialization is *horizontal*, but some instances of *is-member-of* may characterize a more specific kind of relationship, by being instances of *is-manager-of* as well.

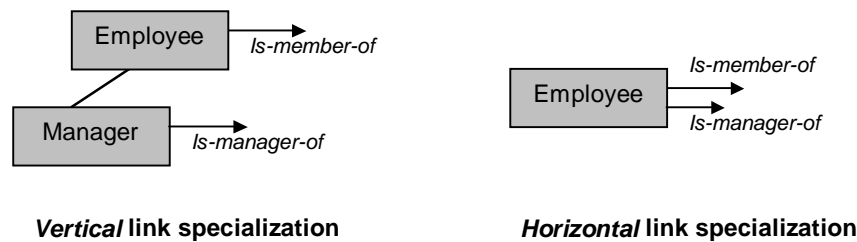


Figure 4: Horizontal and vertical link subtyping

Apart from the domain, a link type’s cardinalities and destination/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. For further details we refer to [32]. 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 define navigational actions on an abstract level, which provides the key to *context-based navigation*, as discussed below.

4. MESH’S CONTEXT-BASED NAVIGATION PARADIGM

4.1 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. For instance 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 relevant when exploring information related to **Van Gogh** but not when retrieving information with regard to “paintings of flowers”.

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, e.g. where the user can provide ordering criteria or where the system can suggest an “optimal” ordering based on previous navigation steps, are discussed in [31]. In figure 5 the context **Van Gogh.has-painted** yields a guided tour (again represented as a dotted line) 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*.

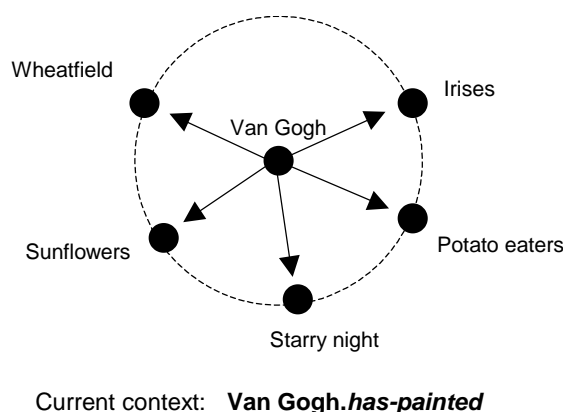


Figure 5: A guided tour derived from the current context

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 in 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.

4.2 Nested tours

Contexts, and consequently guided tours, can exist in ‘layers’. For instance in figure 6, from the current tour’s current node **Sunflowers**, one is able to start a new tour, **Sunflowers.reviews**, nested in the former. A new context emanates, with **Sunflowers** as the new context node, resulting in new indirect links. When this most recent tour has been finished, it is possible to restore the former context and resume the first tour, of which **Sunflowers** again becomes the current node.

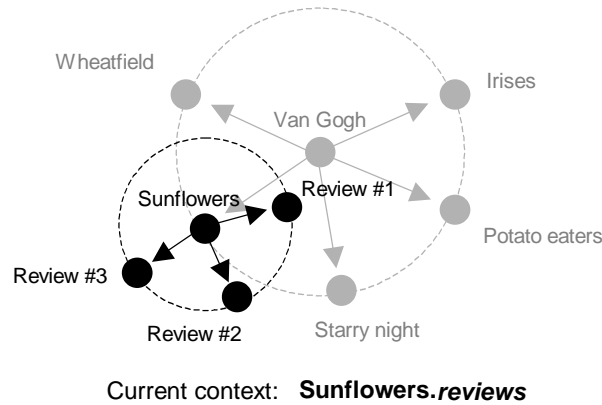


Figure 6: Nested guided tours

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.

4.3 Navigational actions defined in MESH

Navigational actions can then be classified according to two dimensions (figure 7). 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.

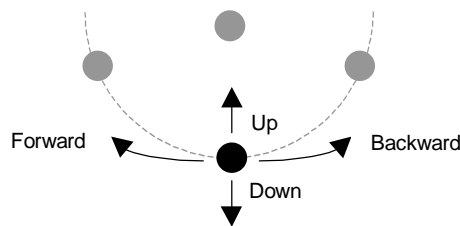


Figure 7: Possible navigational actions

4.3.1 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 (see figure 8).

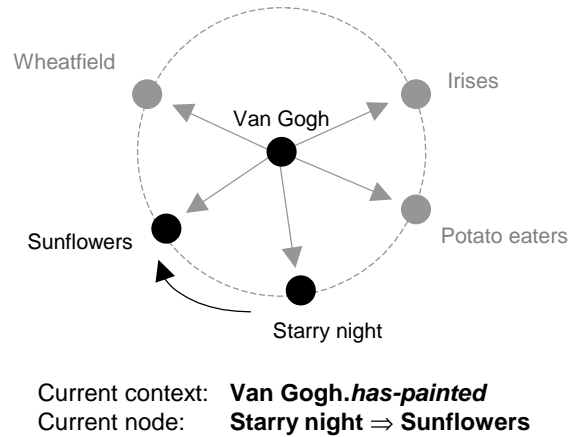


Figure 8: Moving forward within the current tour

Indirect links are not explicitly *anchored*. A node should only return a *move forward* or *move backward* command, independently of the current tour. It's the hyperbase engine's task to map this to the correct destination node (see also section 5.2).

4.3.2 Moving down

Moving down implies an action of 'digging deeper' into the subject matter, looking for additional information regarding the current node. 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.

4.3.2.1 Selection of a link instance

A link *instance* is defined as a (*source node*, *destination node*) tuple. Selection of a link instance l from the source node n_s results in its destination node n_d being accessed:

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

As a link instance corresponds to a single tuple, the outcome is of course a singleton, hence a *single node*, being accessed. E.g. selection of the link (**Sunflowers**, **National Gallery**) from the current node **Sunflowers**, induces an access to the node **National Gallery**:

$$\mathbf{Sunflowers}.\mathbf{(Sunflowers, National Gallery)} := \{\mathbf{National Gallery}\}$$

Anchoring link *instances* is to be avoided where possible since such anchors cannot be managed on an aggregate (i.e. node *type*) level, which is unfavorable from a maintenance point of view. Luckily, thanks to the use of typed links, along with laid down cardinality restrictions, references to link instances can often be replaced by a single reference to a link *type*, as illustrated in the next section.

4.3.2.2 Selection of a link type

Indeed, *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 result may either still be a single node access or a change in context.

4.3.2.2.1 Selection of a unique link type

In the case where a *unique* link type is selected, the resulting “set” of nodes will always be a singleton, since a unique link type can have only one instance per source node. E.g. selecting the link type *exhibited-in* from the node **Sunflowers** has the singleton **{National Gallery}** as a result.

$$\mathbf{Sunflowers.exhibited-in} := \{\text{National Gallery}\}$$

Note that, for a unique link type, anchoring link *type* or link *instance* are equivalent; selection of a single link instance or the complete link type leads to the same single destination node for any source node. Therefore, such links should only be anchored on *type* level, which offers a bonus in maintenance, as such anchor needn't be replaced when the destination node is altered for a given source node. Note that upon *visualization* of a node, the anchor can be tailored to reflect the actual destination node's name if required. The result of selecting a *unique link type* (or a *link instance*) is a node directly linked to the current node becoming the new current node (see figure 9). Other parameters are unaffected.

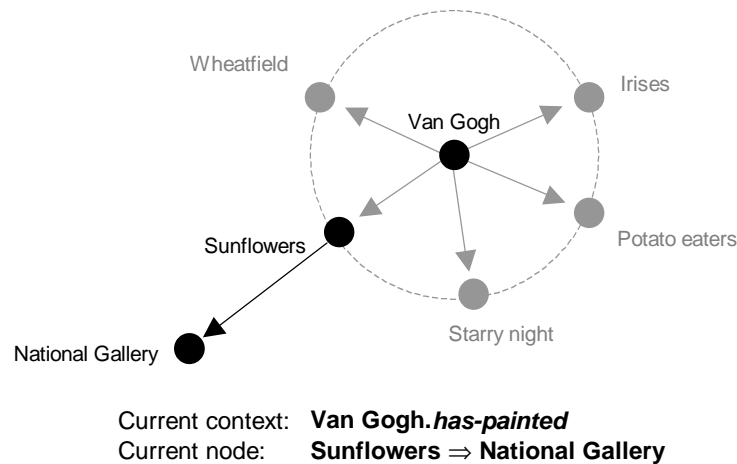


Figure 9: Selection of a unique link type

4.3.2.2.2 Selection of a non-unique link type

If the selected link type is *non-unique*, the resulting set may contain multiple destination nodes. For instance with **Sunflowers** as the current node, selection of the link type *reviews* generates a *collection* of nodes to-be-accessed:

$$\mathbf{Sunflowers.reviews} := \{\text{review\#1, review\#2, review\#3, ...}\}$$

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 (figure 10).

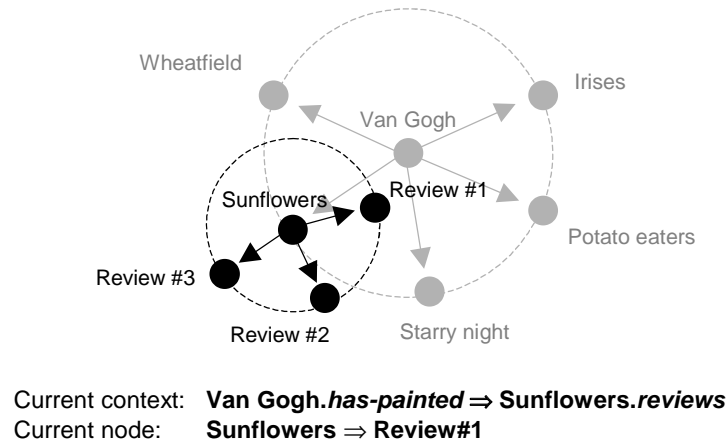


Figure 10: Selection of a non-unique link type

4.3.3 Moving up

Moving up reverses the latest *move down* action. If the latter involved a *unique link type or link instance selection*, the effect of this link selection is cancelled. If the latest link type selected was *non-unique*, 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 becomes the current node. Potentially, a move up action can be triggered automatically upon completion of a guided tour.

4.3.4 Issuing navigational actions on the level of a complete tour

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. In this way, 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, as illustrated in figure 11. Each tour at a lower level is started when its context node becomes the current node in the higher-level tour. Again, *forward* and *backward* links always apply to the "current" tour, i.e. to the open tour at the deepest level.

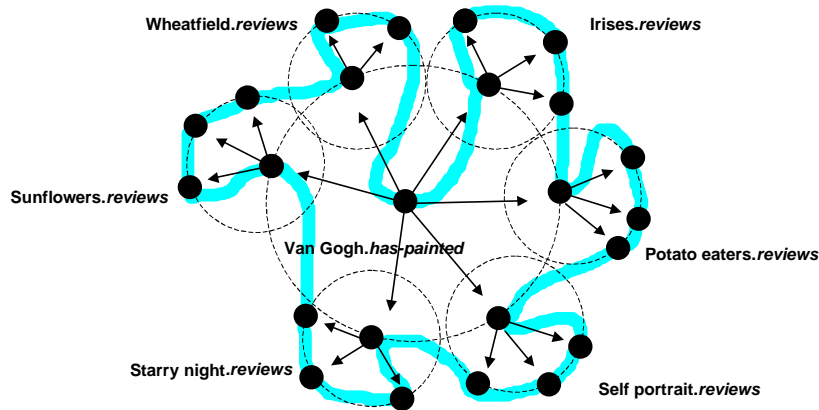


Figure 11: Issuing navigational actions on tour level

4.4 Context-based history lists, maps and bookmarks

MESH exploits a *context-based history list* to account for current tour positions in open tours and to keep track of navigational actions on tour level, so as to aptly impose such actions upon the successive nodes in each newly started tour. The sequence of navigational actions to be cast upon the respective layers of guided tours can be depicted in a *tree*, with each branch representing an abstract navigational action, i.e. a *link type selection*. Each node in the tree symbolizes *the collection of hypermedia nodes accessed through the navigational action represented by the incoming branch*. The *outgoing* branches each stand for a single navigational action, to be applied to all hypermedia nodes in the collection represented by the tree node. The root node denotes the session's *starting point*.

An example is presented in figure 12. With **Paris** as the starting point, the link types *museums* and *squares* are selected. For each node in the **Paris.museums** tour, a nested tour **museum#x.exhibits** is generated and the destination of the unique link type *managed-by* is visited. For all **paintings** belonging to one of the **museum#x.exhibits** tours, the respective unique link types *painting-by*, *style* and *subject* are issued. Currently, there are two nested open tours, with **Louvre** and **Mona Lisa** as respective current nodes.

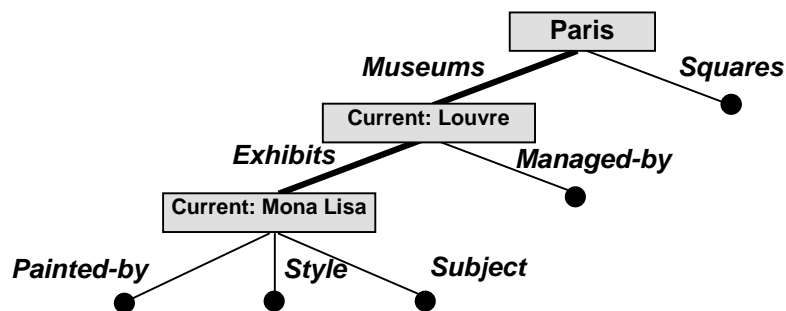


Figure 12: Example of a context-based history list

All information about a complete navigation session can be represented in a compact fashion, based solely on high-level *navigational actions* and currency indications of open tours, without every single *node* participating in the various tours being retained. Consequently, such *context-based* history list will be much shorter and yet more powerful than its traditional counterpart. Indeed, it offers the ability of bookmarking not only single nodes, but of complete navigational situations, allowing to swiftly recapitulate results of earlier exploration. As depicted in figure 13, these can be regenerated by applying

the navigational actions represented in the tree according to a *depth-first* strategy, with the desired sequence of nodes as a result:

```
{Paris,
  {Louvre,
    {Helena Fourment with a Carriage, Rubens, Baroque, Helena Fourment,
      Mona Lisa, Da Vinci, Renaissance, Lisa del Giaconda,
      ...},
    The French Ministry of Culture,
  Musée d'Orsay,
    {Dancing Lesson, Degas, Impressionism, A dance class around 1874,
      Bedroom at Arles, Van Gogh, Expressionism, Residence in Arles,
      ...},
    The city of Paris,
  ...},
  {Place de la Bastille, Place Charles-de-Gaulle, Place de la Concorde, ... }}
```

Figure 13: A navigational situation restored from a context-based history list

Note that the tree representation is not only useful for *storing* a navigational situation, but also allows for *visualizing* a user's current situation in a highly compact manner in maps and overviews. In this respect, the *move up* and *move down* actions indeed correspond to moving up or down in the graph.

5. IMPLEMENTATION ISSUES

5.1 Introduction

Whereas *MESH*'s data model and navigation paradigm can be implemented in several ways, this section proposes an approach that exploits a relational database for link storage, which leaves a large amount of freedom regarding the implementation of the nodes' actual *content*. A first subsection presents a generic framework, independent of the implementation environment. A second subsection applies these principles to a Web environment and provides a description of the current prototype implementation, which is indeed Web-based.

5.2 A generic application framework

The navigational paradigm presented in the previous sections requires a hyperbase that is *searchable* for its link structure: to generate the necessary guided tour links at run-time, the application needs to be able to query the hyperbase for nodes related to the current context node. As a consequence, there are two alternatives for hyperbase implementation. The first one is to encapsulate all links within the body of the nodes, as it is the case in many hypermedia environments, such as traditional HTML pages in the Web. However, unlike many other environments, *MESH* should allow for all nodes to be *queried* for their link information. This would call for an object-oriented database system where each node is an object and where all links are represented as symbolic pointers to other objects, which can be queried by means of an object-oriented query language such as OQL [10]. However, such "universal storage" approach, forcing all nodes with their (possibly very distinct data formats) into one proprietary object-oriented database model, would result in an unacceptable lack of openness and dependence upon one specific object-oriented DBMS. Therefore a second alternative was opted for, which in broad outlines corresponds to the OHP framework proposed by the Open Hypermedia Community [16], [40]. The *information content* and *navigation structure* of the nodes are separated and stored distinctly into storage devices that are tailored to the specific needs of the type of information stored. A simple relational database can be used to capture the link structure and meta-information of the hypermedia system, along

with references to the physical addresses of the corresponding nodes. This “universal access” option leaves much more freedom to implement the *content* of a node. The resulting system consists of three types of components: the *nodes*, the *linkbase/repository* and the *hyperbase engine* (figure 14).

The *nodes* side of the hypermedia system is considered as a potentially *heterogeneous* collection of entities, ranging from flat files (e.g. HTML fragments) to objects in an object-oriented database, each containing one or more embedded multimedia objects. Nodes are very loosely specified. They only have to be associated with a unique identification such as a URL and should be able to return a *navigational action* (i.e. a link type selection) upon closure. However, since link information is stored separately of the nodes, a node does not have to be a searchable object. Its internal content is shielded from the outside world by the indirection of link types playing the role of a node’s *interface*. Optionally, a node can be endowed with the intelligence to tune its reaction to the *context* in which it is accessed, by taking the selected link type into account upon visualization. In this way the multimedia objects that are most relevant to this particular context can be made current upon node access, hence the so-called *context-sensitive visualization* principle. This allows for different *views* to be defined over the same information. For instance, the node **Sunflowers** may present itself differently when accessed within the context **Van Gogh.has-painted** than within the context **Musée d’Orsay.exhibits**. Again, the actual implementation of this principle may depend on the environment, ranging from a simple “anchor” in a HTML document to a real visualization routine being associated with each link type (see [31] for further details regarding context-sensitive node visualization).

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 application that listens for link type selections issued 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 database, the hyperbase engine can access this information by means of simple, pre-defined *database queries*, i.e. without the need for searching through *node content*. In its most rudimentary form, a query to calculate all nodes in a guided tour as resulting from selecting the link type *theSelectedLinkType* from the node *theCurrentNode* looks as follows (an actual query will be more complex as also child link types are to be taken into consideration):

```
select n.nodeID, n.nodeName, n.physicalLocation, lt.linkTypeName
from nodes n, links l, linkTypes lt
where l.linkType = :theSelectedLinkType_ID
and l.sourceNode = :theCurrentNode_ID
and n.nodeID = l.destinationNode
and lt.linkTypeID = l.linkType
order by n.NodeName [or some more complex ordering criterion]
```

The combination of the hyperbase engine and linkbase/repository closely matches the concept of a *linkservice* as defined in OHP. *MESH* has in common with such systems that the coupling between the different components is kept as loose as possible. This results in links being isolated from node content and being stored in a separate link database. However, in contrast to such general-purpose link services, *MESH* explicitly specifies semantics for conceptual data modeling and navigation. Therefore, the hyperbase engine’s task is more elaborate, as a crucial part of its functionality lies in keeping track of the

current context and deducing guided tours from run-time information (the current context) and persistent information (stored links and metadata). It is also to provide the destination node with information about the link type through which it is accessed, to cater for context-sensitive node visualization.

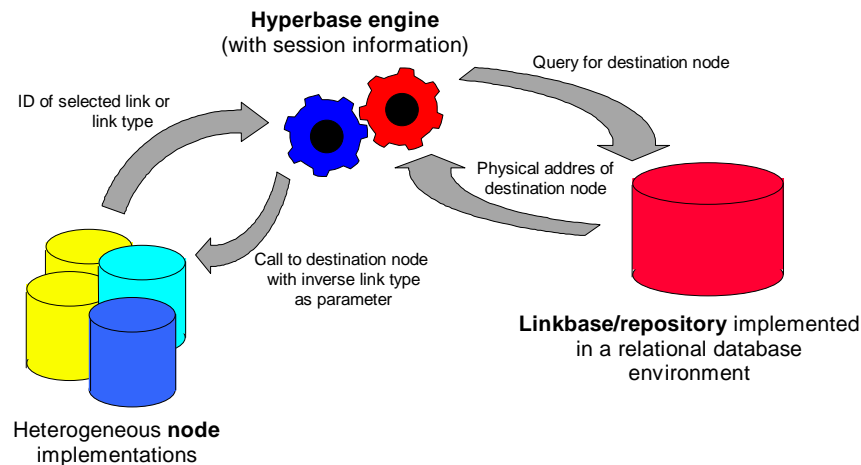


Figure 14: A generic application framework

5.3 The current Web-based prototype

MESH's current prototype implementation provides a servlet-based hypermedia engine, which processes navigational actions and accesses a relational DBMS containing the linkbase/repository. The servlet also keeps track of run-time information such as the current context, which is however duplicated in persistent storage for robustness. Node *content* is presented as HTML code in a "traditional" webbrowser. However, because the framework leaves much freedom as to the actual implementation of nodes and aspects, the HTML code can be stored as a single *static* file, it can be *assembled* from multiple XML or HTML fragments, or it can be generated *dynamically* at runtime. The only requirement is that it can be uniquely identified, e.g. by means of a URL or an object ID. The webbrowser is enhanced with a "navigation applet", which provides enriched user interface functionality and ad-hoc node overviews (although the applet is not strictly required for the core functionality of the system).

Navigational actions can be selected from either *within* an HTML document or from an applet-generated node overview. However, the HTML code does not contain direct references to related nodes. It only defines a node's *multimedia content*: the *links* are removed from the node's content and are stored in the linkbase/repository, along with the meta information. A link as represented in the HTML code consists of a reference to the *servlet's URL* and a *parameter*. The latter identifies the direct link (type) or next/previous action to be selected by the anchor. Upon stimulation of the anchor, the parameter is passed to the servlet, which calculates the correct destination node and detects whether the action induces a context change. In the latter case, the corresponding indirect links are generated by the servlet, based on the selected link type and the information in the linkbase/repository. Hence the servlet's support for navigational actions entails both within-tour navigation and the initiation of new guided tours, as required by the navigation paradigm.

Context-sensitive node visualization again depends on the origin of the HTML code: as to static HTML documents, it consists of defining a mapping between link type and HTML anchor, such that the destination document scrolls to the appropriate section when this link type is selected. As to HTML that is generated entirely dynamically, the link type is mapped to a node-object's *presentation routine* (see [31] for further details), which in fact generates the HTML code for the node and its associated aspects.

Obviously, the former is a rather rudimentary approach to context-sensitivity, whereas the latter allows for much more subtlety in adjusting a node's visualization to the current context. An intermediate solution is the presentation routine dynamically assembling a single HTML document from different static HTML or XML *fragments*, representing the node's and associated aspects' content.

At present, the runtime environment provides a read only system: authoring is executed by means of a separate offline application. In the future, however, the runtime system is intended to enable users with the right privileges to reallocate links and update node content during a navigation session. Note that maintenance efforts are greatly reduced as nodes are very loosely coupled, i.e. their links are not embodied within the HTML code. Hence, updates to the link structure can be executed by means of SQL queries in the relational database, instead of editing node content. Also, the current implementation restricts applicability of the navigation paradigm to a single web site or at the very least to multiple sites controlled by a single linkbase/repository. Therefore, another future research topic targets a *distributed* linkbase/repository. Note, however, that the present restriction in scope is merely a consequence of the actual implementation, not of the navigation paradigm in se.

6. CONCLUSIONS

6.1 Benefits of *MESH*'s proposed navigation paradigm

This section presents conclusions and comparisons to related work with respect to *MESH*'s *navigation paradigm*. For an evaluation of the *entire* framework, as well as more extensive references to related work and an overview of future research issues, see [31].

MESH's context-based navigation paradigm implements the design principles referred to in section 2.2 by means of dynamic linear paths throughout the information space, so as to reduce cognitive overhead. A sense of *direction and position* is induced, both *within* a single tour and orthogonally to the latter. Orthogonality of the navigational actions (*forward/backward* and *down/up*) improves the user's ability to ascertain his navigational options. In this respect, the comprehensive, tree-shaped history list representation imposes a flexible *hierarchical view* upon the hyperbase structure, which can be visualized as an adaptive map or node overview. The abundance of meta-information as node, aspect and link types allows for enriching such maps and overviews with concepts of varying granularity very easily, which is applied e.g. in *fish-eye views*.

Note that also the abstractions of the data model attribute to easier 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. Furthermore, 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. Through the specification of abstract navigational actions *on tour level*, complex navigation patterns can be applied to all nodes in a tour without additional effort.

The *context* concept not only reduces cognitive overhead, but as a representation of what various nodes have in common, also positively influences *document coherence*, especially in combination with *context-sensitive node visualization*. 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.

The current prototype implementation is built around a purely relational linkbase/repository containing links and metadata. The nodes themselves can also be generated from a relational database, but can equally be implemented as static HTML fragments. A future incarnation is likely to be entirely XML-based, using either XLL or RDF for link definition and XML schemas to specify node templates.

6.2 A comparison to related work

6.2.1 Navigation in conceptual space and guided tours

As already said, object-oriented techniques were initially applied in hypermedia to model *functional behavior* of a hypermedia system's components and to support specific media types, i.e. to facilitate *implementation*. Later on, approaches such as *EORM* [30], *RMM* [24], [25], *HDM* [21] and *OOHDM* [45], [46] emerged. They have in common with *MESH* that the modeling of the *conceptual domain* is accomplished through object-oriented (or entity-relationship) techniques. However, whereas the other approaches apply general-purpose object-oriented techniques (e.g. [25]) *MESH* provides modeling primitives specifically tailored to hypermedia, acknowledging the fact that *structure* and *relationships* should prevail over *behavior* as important modeling factors. *MESH* is the only approach to formally define link subtyping semantics and a node classification mechanism supporting plural, dynamic specializations by means of aspect descriptors and aspects. As a consequence, there is no need for an explicit *navigational design*, apart from the *domain model*: one truly navigates in *conceptual space*. A recent, model-driven approach to web site development is *WebML* [11], which also builds on an object-oriented conceptual data model. However, again, *WebML* does not provide its own data model, but borrows from entity-relationship modeling and UML. Among all systems mentioned, *WebML*'s data model is the one that resembles *MESH*'s the most. At least on the level of individual node and link types, as *WebML* does not provide link subtyping nor secondary node classifications.

As to navigation, *MESH*'s context-based navigation paradigm tackles the disorientation problem by providing dynamic guided tours throughout the information space. *EORM*, *RMM*, *HDM*, *OOHDM* and *WebML* 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. For instance in *OOHDM*, topologies are defined through *navigational contexts*. Several types of navigational contexts can be designed. Contexts can also be *nested*, such that complex, hierarchical navigation patterns become possible. However, upon implementation, navigational contexts are to be authored semi-manually by means of explicitly stored queries or enumerations of nodes. In *MESH*, only *direct* links are to be explicitly authored. Their maintenance is accomplished rather effortlessly, as they are stored separately in a relational database. Neither guided tours nor indexes require any maintenance or design effort, as the author is not even engaged in their realization. They are generated at run-time upon user request. For that purpose, the appropriate queries are inferred automatically by the hyperbase engine, with the context node and context link type as input parameters. No additional constructs are to be defined or maintained. Moreover, navigational actions can be anchored on node *type* level, hence they have to be authored only once for a whole class of nodes. For instance a *painting-by* anchor needs to be defined only once for the entire node type **painting**, instead of for each **painting** instance. *MESH* has the guided tour as default construct for navigating through a set of nodes as generated after selection of a non-unique link type, but all kinds of *indexes* can be derived just as easily by querying the linkbase/repository.

Worth noting is the importance of the ready availability of metadata. For example when a user wants to refine a guided tour, this can be accomplished by selecting a *child link type* from the original context link type, hence reducing the tour to nodes with a more specific semantic relationship to the context node. Link typing also enables the end user to issue the same navigational actions to all nodes of a given tour at once, instead of for each separate node, as was discussed in section 4.3.4. Furthermore, these type-level navigational actions provide *MESH* with the unique ability to bookmark a *complete navigational situation* (instead of single nodes) in a very compact manner, with the possibility to resume navigation later on, from the exact point where it was left.

The idea of metadata-driven navigation also prevails in conceptual hypermedia systems such as *Hyperindices* [7] and *COHSE* [9]. These envisage the authoring of relevant links by *reasoning* over the

metadata. The conceptual data model is conceived as a genuine ontology. Although not a prerequisite for implementing *MESH*'s navigation paradigm, a valuable extension could be the incorporation of Artificial Intelligence techniques to infer pertinent links and guided tours. These could take advantage of *MESH*'s semantically rich data model. A future research issue, seen in the light of the Semantic Web vision, could be the specification of (and the extension of) *MESH*'s modelling primitives in RDF(S).

6.2.2 Query-based systems

MESH has in common with *query-based* systems such as *Strudel* [19], [20] that web sites are defined by means of *database techniques*. Management of hypermedia structure and of "internal" node data are perceived as two orthogonal tasks. *Strudel* uses and extends a declarative query-language, *StruQL*, for website implementation, which explicitly allows for expressing integrity constraints. Another similarity is that "links" are modeled by means of queries. However, again, such queries are to be authored explicitly, in contrast to *MESH* where they are generated automatically at runtime, according to the user's actions. A database-like approach similar to *Strudel* can be found in *Araneus* [38]. As in *MESH*, pages are defined as instances of a page scheme. However, *Araneus* lacks *MESH*'s subtyping and inheritance mechanism.

The characteristic of guided tours combining navigation with a query element can also be found in the *StrathTutor* system [37]. However, whereas the latter's queries are based on a predefined set of node *attributes*, *MESH*'s criteria are its *direct links*. Hence (the semantics of) a node's relations to neighboring nodes model its properties and determine its inclusion in guided tours.

6.2.3 Set-based hypermedia approaches

Set-based hypermedia paradigms such as *CHM* [17], [18], the *HM-Data Model* [34], [35], [49] and *Hyper-G/Hyperwave* [1], [29] equally provide inherent support for navigation in two orthogonal planes; *inside a collection* and *across collection boundaries*. Their *current container* and *current member* concepts are comparable to *MESH*'s *current context* and *current node* respectively. Furthermore, *Hyperwave* shares with *MESH* its ability to separate document content from structural links: the link structure is stored in a database, whereas document content is stored separately. As a consequence, it becomes very easy to maintain link integrity and to provide run-time generated overviews of the available information. However, although some set-based systems offer limited support for tagging documents with metadata such as *author*, *topic* etc., they lack the abstractions of a firm underlying conceptual data model with typed node interrelations. Likewise, the opportunity of issuing abstract navigational actions on tour level is a feature that is exclusive to *MESH*. The set paradigm has certainly resulted in some interesting and fruitful insights in hypermedia development. Concepts such as *context-dependent visualization* and *navigational context* originate (at least partially) in set-based systems. However, whereas these concepts were initially claimed to be particular to a set-based approach, *MESH*, among others, has proven that they can be successfully transferred to the node/link paradigm.

In Trellis [50], the concept of *nested sub-systems* induces a browsing behavior in "levels" that is somewhat similar to set-based navigation. Unfortunately, the result is a pre-enforced hierarchic structure, instead of a dynamic one as in *MESH*. Interestingly, the Petri-net paradigm is capable of representing a wealth of browsing semantics such as *guided tours*, *concurrent browsing paths*, *multiple current nodes* and *personalized views*. Again, however, such constructs are to be authored explicitly, as constraints upon the feasible Petri-net *state transitions*.

6.2.4 Context-sensitive node visualization

RMM, *HDM* and *OOHDM* also in one way or another incorporate node visualization mechanisms that are sensitive to the “context” in which a node is accessed. *RMM* divides a given entity type into *slices*, each one grouping a different set of *attributes*. Context-dependent visualization is implemented by denoting a “head” slice accessed by default, but allowing each link to target its own “destination slice”. *HDM* and *WebML* have a very similar approach. In *OOHDM*, context-sensitive visualisation is achieved by means of *context classes*, which are groupings of attributes that are only relevant to (a) specific context(s). *MESH*, however, is the only approach to use a link’s *type* as the parameter to determine the destination node’s visualization. It is also the only approach that stays true to the object-oriented paradigm by looking upon destination anchors as *visualization routines* that provide an interface to a destination node, but hide their actual implementation from the source node.

A feature yet to be implemented in *MESH* is the ability to incorporate “borrowed” multimedia content from adjacent nodes, e.g. a **painter** node including pictures of his most significant **paintings**. One of the merits of *OOHDM* in this respect is that, in the navigational scheme, nodes as instances of navigational classes may combine attributes of different related conceptual classes. This is also true for *pages* as defined in *WebML*’s *Composition Model*. In *MESH*’s current state, a node can only include a single property of related nodes, namely their *description field*. The latter is stored within the database and hence can be easily retrieved and used in a given node, possibly as an outgoing anchor denoting a link to the former node. However, it would be more than desirable to provide a construct for allowing one node to present multimedia data that belongs to related nodes, without having to replicate these data. A similar extension to *RMM* is suggested in [25], with *m-slices* combining attributes belonging to different entities to be presented in a single window.

6.2.5 Adaptive hypermedia

Adaptive hypermedia systems [6], [54] allow for both node visualization and accessible links to be influenced by an inference mechanism, based on knowledge about the *current user*. This knowledge is (partially) obtained by observing the user’s previous actions. For instance direct guidance systems such as *Webwatcher* [27] suggest relevant links based on experience with previous users’ browsing behavior and a set of keywords that have been provided at the beginning of the current user’s session. One is “guided” along potentially relevant pages, with the system actually taking over navigation control, at least partially.

While featuring dynamic node content as well as dynamic links, *MESH* intentionally differs from typical adaptive hypermedia systems in that it does not try to *classify* the current user nor does it account for an explicit *user model*. Also, observation of the user’s navigational actions is not aimed at *influencing* his behavior: *MESH*’s dynamism is merely targeted at facilitating navigation and providing a structured perspective upon the information space. A guided tour of *indirect* links is invariably the result of a *direct* link type selection by the user: the initiative fully resides with the latter.

Therefore, adaptive hypermedia techniques could be seen as a complement to *MESH*, rather than a substitute. Indeed, especially in large hyperbases, with an oversupply of outgoing link anchors for a given source node, it could be required to impose dynamic behavior upon these *direct* links as well. A user profile could then be applied to highlight the most relevant direct link (type) anchors and hide irrelevant ones. This would entail a relaxation of the assertion of complete navigational flexibility and the end user having autonomous control over navigation strategy, with certain links only becoming visible or accessible after a certain precondition is fulfilled, as described e.g. in [8]. Hence, whereas *MESH*’s current dynamism entails the generation of guided tours *in accordance with the user’s actions*, the latter could be assisted by adaptive techniques *suggesting which action to take*. In this respect, *MESH*’s rich modeling abstractions with typed nodes, links and aspects, as well as its *context* notion, could provide

valuable semantics as additional input to existing adaptation techniques. Also, rules could be applied at the desired level of granularity: general rules at parent node type level and specific rules at child node type (or even instance) level. In that case, the inheritance and overriding mechanism would have to be extended to accommodate for such adaptation rules. A related ongoing research issue is the possibility of the hypermedia system taking the initiative of reorganizing and optimizing the nested context structure emanating from the user's actions.

REFERENCES

- [1] K. Andrews, F. Kappe and H. Maurer, The Hyper-G Network Information System, *Journal of Universal Computer Science*, Vol. 1, No. 4, 1995.
- [2] H. Ashman, A. Garrido and H. Oinas-Kukkonen, Hand-made and Computed Links, Precomputed and Dynamic Links, *Proc. of Hypertext - Information Retrieval - Multimedia (HIM ' 97)*, Dortmund, 1997.
- [3] A. Bapat, J. Wäsch, K. Aberer and J. Haake, An Extensible Object-Oriented Hypermedia Engine, *Proceedings of the seventh ACM Conference on Hypertext (Hypertext ' 96)*, Washington D.C., 1996.
- [4] T. Berners-Lee and R. Cailliau, The World-Wide Web, *Commun. ACM* Vol. 37, No. 8, 1994.
- [5] M. Bernstein, The Navigation Problem Reconsidered, *Hypertext/Hypermedia Handbook*, E. Berk and J. Devlin Eds., McGraw-Hill, New York, 1991.
- [6] P. Brusilovsky, Methods and techniques of adaptive hypermedia, *User Modeling and User-Adapted Interaction* Vol. 6, No. 2-3, 1996.
- [7] P. Bruza, Hyperindices: A Novel Aid For Searching in Hypermedia, *Proceedings of the European Conference on Hypertext*, France, 1990.
- [8] L. Calvi and P. De Bra, Improving the Usability of Hypertext Courseware through Adaptive Linking, *Proceedings of the eighth ACM Conference on Hypertext (Hypertext ' 97)*, Southampton, 1997.
- [9] L. Carr, S. Bechhofer, C. Goble and W. Hall, Conceptual linking: Ontology-based Open Hypermedia, *Proceedings of the Tenth International World Wide Web Conference*, Hong Kong, 2001.
- [10] R. Cattell and D. Barry, *The Object Database Standard: ODMG 3.0*, Morgan Kaufmann Publishers Inc., San Francisco, CA, 2000.
- [11] S. Ceri, P. Fraternali and S. Paraboschi, "Web Modeling Language", (WebML): a modeling language for designing Web sites. *Proc. of the 9th. International World Wide Web Conference*, 2000.
- [12] P. Chen, The entity-relationship approach: Toward a unified view of data, *ACM Trans. Database Syst.* Vol. 1, No. 1, 1979.
- [13] A. Cockburn and S. Jones, Which way now? Analyzing and easing inadequacies in WWW navigation, *International Journal of Human-Computer Studies* No. 45, 1996.
- [14] J. Conklin, *Hypertext: An Introduction and Survey*, *IEEE Computer* Vol. 20, No. 9, 1987.
- [15] H. Davis, W. Hall, I. Heath, G. Hill and R. Wilkins, MICROCOSM: An Open Hypermedia Environment for Information Integration, *Computer Science Technical Report CSTR 92-15*, 1992.
- [16] H. Davis, S. Reich and A. Rizk, OHP - Open Hypermedia Protocol, Working Draft 2.0, 1997.
- [17] E. Duval, H. Olivié, P. Hanlon and D. Jameson, HOME: An Environment for Hypermedia Objects, *Journal of Universal Computer Science* Vol. 1, No. 5, 1995.
- [18] E. Duval, H. Olivié and N. Scherbakov, Contained Hypermedia, *Journal of Universal Computer Science*, Vol. 1, No. 10, 1995.
- [19] M. Fernández, D. Florescu, J. Kang, A. Levy and D. Suciú, Catching the boat with Strudel: experiences with a Web-site management system, *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, Seattle, 1998.
- [20] M. Fernández, D. Suciú and I. Tatarinov, Declarative specification of data-intensive Web sites, *Proceedings of the second conference on Domain-specific languages*, Austin, TX, 1999.
- [21] F. Garzotto, P. Paolini, and D. Schwabe, HDM - A Model-Based Approach to Hypertext Application Design, *ACM Trans. Inf. Syst.* Vol. 11, No. 1, 1993.

- [22] F. Halasz, Reflections on NoteCards: Seven Issues for Next Generation Hypermedia Systems, *Commun. ACM* Vol. 31, No. 7, 1988.
- [23] N. Hammond, Learning with Hypertext: Problems, principles and Prospects, *HYPERTEXT a psychological perspective*, C. McKnight, A. Dillon and J. Richardson Eds., Ellis Horwood, NY, 1993.
- [24] T. Isakowitz, E. Stohr and P. Balasubramanian, RMM, A methodology for structured hypermedia design, *Commun. ACM* Vol. 38, No. 8, 1995.
- [25] 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.
- [26] I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard, Object-Oriented Software Engineering, Addison-Wesley, New York, 1992.
- [27] T. Joachims, D. Freitag and T. Mitchell, Webwatcher: A Tour Guide for the World Wide Web, CMU research report, 1996.
- [28] D. Jonassen, Semantic net elicitation: tools for structuring hypertext, *Hypertext: State of the Art*, R. McAleese and C. Green Eds., Intellect, Oxford, 1990.
- [29] F. Kappe, Managing Knowledge with Hyperwave Information Server, Hyperwave White Paper Version 1.2, 1999.
- [30] 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, 1994.
- [31] W. Lemahieu, Improved Navigation and Maintenance through an Object-Oriented Approach to Hypermedia Modelling, Doctoral dissertation (unpublished), K.U.Leuven, Leuven, 1999.
- [32] W. Lemahieu, MESH: A Model-Based Approach to Hypermedia Design, in: Chen, Q. (ed.) *Human Computer Interaction: Issues and Challenges*, Idea Group Publishing, Hershey, PA, 2001.
- [33] D. Lucarella, A Model For Hypertext-Based Information Retrieval, Proceedings of the European Conference on Hypertext, Versailles, 1990.
- [34] H. Maurer and N. Scherbakov, The HM Data Model, IIG Report, Graz, 1992.
- [35] H. Maurer, N. Scherbakov and A. Nedoumov, HM-CARD: A Second Generation Hypermedia Authoring Tool, Proceedings of LAIR-MIPRO '95, Opatija, 1995.
- [36] M. Mayes, A method for evaluating the efficiency of presenting information in a hypermedia environment, *Computer in Education* Vol. 18, No. 1, 1994.
- [37] J. Mayes, M. Kibby and H. Watson, StrathTutor: the development and evaluation of a learning-by-browsing system on the Macintosh, *Computers and Education*, No. 12, 1988.
- [38] G. Mecca, P. Atzeni, A. Masci, P. Merialdo and G. Sindoni, The Araneus Web-Base Management System, Exhibits Program of ACM SIGMOD, 1998.
- [39] B. Meyer, Object-Oriented Software Construction, Second Edition, Prentice Hall Professional Technical Reference, Santa Barbara, 1997.
- [40] D. Millard, S. Reich and H. Davis, Reworking OHP: the road to OHP-Nav. Proceedings of the fourth Workshop on Open Hypermedia Systems at Hypertext '98, Pittsburgh, 1998.
- [41] J. Nanard and M. Nanard, Hypertext Design Environments and the Hypertext Design Process, *Commun. ACM* Vol. 38, No. 8, 1995.
- [42] J. Nielsen, The Art of Navigating Through Hypertext, *Commun. ACM* Vol. 33, No. 3, 1990.
- [43] J. Nielsen, Navigating through Large Information Spaces, *Hypertext and Hypermedia*, Academic Press, Boston, 1990.
- [44] C. Ramaiah, An Overview of Hypertext and Hypermedia, *International Information, Communication & Education* Vol. 11, No. 1, 1992.
- [45] G. Rossi, D. Schwabe and F. Lyardet, Web application models are more than conceptual models, Proc. of the World Wild Web and Conceptual Modeling' 99 Workshop, ER' 99 Conference, Paris, 1999.
- [46] G. Rossi, D. Schwabe and F. Lyardet, Abstraction and Reuse Mechanisms in Web Application Models, Proceedings of the World Wild Web and Conceptual Modeling' 00 Workshop, ER' 00 Conference, Salt Lake City, 2000.
- [47] M. Simpson, Navigation in the hypertext, *Hypermedia* Vol. 16, No. 2, 1994.

- [48] M. Snoeck, G. Dedene, M. Verhelst and A. Depuydt, Object-Oriented Enterprise modeling with MERODE, Universitaire Pers Leuven, Leuven, 1999.
- [49] P. Srinivasan, Incorporating Intelligent Navigational Techniques to Hypermedia, Proceedings of LAIR-MIPRO '95, Opatija, 1995.
- [50] P. Stotts and R. Furuta, Petri Net Based Hypertext: Document Structure with Browsing Semantics, ACM Trans. Inf. Syst. Vol. 7, No. 1, 1989.
- [51] M. Thüring, J. Hannemann and J. Haake, Hypermedia and Cognition: Designing for comprehension, Commun. ACM Vol. 38, No. 8, 1995.
- [52] R. Trigg, Guided Tours and Tabletops: Tools for Communicating in a Hypertext Environment, ACM Trans. Office Inf. Syst. Vol. 6, No. 4, 1988.
- [53] U. Wiil and J. Leggett, Hyperform: a hypermedia system development environment, ACM Trans. Inf. Syst. Vol. 15, No. 1, 1997.
- [54] H. Wu, E. De Kort and P. De Bra, Design Issues for General-Purpose Adaptive Hypermedia Systems. Proceedings of the ACM Conference on Hypertext and Hypermedia, Aarhus, Denmark, 2001.