Structuring Primitives in the Callimachus Component-Based Open Hypermedia System

Manolis Tzagarakis [1], Dimitris Avramidis [1], Maria Kyriakopoulou [1], monica schraefel [2], Michalis Vaitis [1], Dimitris Christodoulakis [1]

[1] Department of Computer Engineering and Informatics, University of Patras, GR-265 00 Patras, Greece
Computer Technology Institute,
Riga Ferraiou 61, GR-262 21 Patras, Greece
`{tzagara, avramidi, kyriakop, vaitis, dxri}@cti.gr`

[2] Department of Computer Science, University of Toronto,
10 Kings College Rd, Toronto, Canada
`ms@cs.toronto.edu`

Abstract. Driven by the philosophy of the “primacy of structure over data”, CB-OHS present an open set of structure servers providing structural abstractions of different hypermedia domains. To address the emerged requirements and to facilitate the development of structure servers, structure should be handled as a first class entity. We propose patterns for structure, called *templates*, that define the structural model upon which structure servers operate. We present how structure servers are developed and operate in the Callimachus CB-OHS. Development of structure servers within Callimachus is based on the explicit specification of structure with the use of an atomic structural primitive called the *structural element*. Explicit structure specification eases the development of structure servers in CB-OHS, make such development less error prone and can provide the basis for tailoring domain specific abstractions.

1 Introduction

The transition from monolithic to Open Hypermedia Systems (OHS) envisioned the provision of hypermedia functionality to third-party applications as well as interoperability among different hypermedia systems, in a distributed, heterogeneous environment. Much research and development work, lasting for many years, resulted in the realization that various hypermedia domains impose different requirements and need different abstractions. This led to common agreement that special protocols should be developed for each hypermedia domain.

The specification and implementation of the Open Hypermedia Protocol (OHP) [2] was the first effective approach for supporting the well-known associative domain. New systems and protocols supporting structural abstractions of other domains, such as taxonomic [12] or spatial [6], had to be built from scratch. Inspired by the general trend in computing towards component-based, distributed, interoperable systems, Component-Based Open Hypermedia Systems (CB-OHS) emerged [10]. Driven by the philosophy of the “primacy of structure over data” [47], CB-OHS provide an open set of components, called *structure servers*, each one providing the foundations to support the structural abstractions of a single hypermedia domain in a convenient and efficient way [8]. Functionality that is common among all domains such as storage, versioning, naming and collaboration is regarded as part of the infrastructure.

Besides providing appropriate abstractions of particular hypermedia domains, the CB-OHS approach may be useful in other situations as well. Sub-domains exist that may place constraints on an individual domain. As an example, the *typed* (or *structured*) *navigational* domain can be considered as a sub-domain or a *specialization* of the navigational domain. Moreover, some domains may be regarded as functional aggregations of other domains, such as the *linguistic* and the *digital libraries* domains where the taxonomic and the navigational domain have to co-exist and even interoperate at some levels [7, 11, 48]. Also, a wide number of applications need specific hypermedia functionality, not only generic link services. For example, a *lexicographic* application requires specific associative mechanisms to express relations among lemmata in a lexicon (hyponyms, hyperonyms, homonyms, synonyms, etc.)[7].

CB-OHSs are one incarnation of structural computing, a philosophical view in which structure is regarded as more important than data. Structural computing forces a paradigm shift from data oriented solutions and systems to structure oriented ones. Such shift imposes new problems and requirements [13, 7, 3] and has implications in the design of CB-OHS many of which are not fully understood. One of these implications is the elevation of structure to a first class entity requiring from CB-OHS explicitly to recognize this in terms of structure oriented models and computations. Hypermedia domains appear thus as properly configured aspects of general structure management facilities. Yet, the way from general structure oriented facilities to hypermedia domain specific concepts is not obvious. In the establishment of such mechanism (i.e. from abstract to concrete), structure servers play an important role since they host the appropriate reification methods. Nevertheless, development of a structure server is a complicated project to be repeated from scratch every time a new domain, sub-domain or application has to be supported [49]. Thus, specific CB-OHS development tools are required to facilitate their development. Such software development tools for structure servers should include structure specification capabilities that will provide the fundamental elements upon which behavioral operations can be defined or programmed, such as traversal semantics, structure constraints and even presentation and transition effects.

In this paper we present how structure servers are developed and operate in the Callimachus CB-OHS. Development of structure servers within Callimachus is based on the explicit specification of structure with the use of an atomic structural primitive called the *structural element*. The need to specify structure explicitly has already been pointed out [46]. Explicit structure specification eases the development of structure servers in CB-OHS, make such development less error prone and can provide the basis for tailoring domain specific abstractions. Patterns for structure are presented, called *templates*, that define the structural model upon which structure servers operate. In Callimachus, the domain specific structural model (e.g., navigational, taxonomic) is perceived as a first class entity that can be formally defined through templates. This is in contrast to contemporary OHS, in which the structure model is closely coupled with the infrastructure or the individual structure servers. The interoperability level is thus raised from the infrastructure level (i.e. the storage) to the template level. The perception of templates is part of a larger effort towards the creation of software engineering tools and methodologies for the development of CB-OHSs within the Callimachus framework.

The paper is organized as follows. Sections 2, 3,4,5 discuss some hypermedia domains that have been reported in the literature focusing on prevailing abstractions and interaction styles. Section 6 compares the domains exhibiting common concepts. Section 7,8 introduce the Callimachus CB-OHS and describe its elementary constructs. Section 9 analyses how structure servers use the elementary constructs to provide navigational services. Section 10 compares Callimachus to other generalized hypermedia approaches. Section 11 concludes the paper.

2 Associative Domain: Navigational Hypermedia Systems

The associative domain was one of the first domains implemented in early hypermedia systems, in which associations were provided between information. These associations – widely known as links – were to be made between semantically related information items, as Vannevar Bush envisioned in the Memex [17]. On the one hand, hypertext links condition the user to expect purposeful and important relationships between linked materials, whereas on the other hand the emphasis upon this linking stimulates and encourages habits of relational thinking in the user [18].

Inspired by Bush's ideas, many researchers have investigated navigational hypertext in computer applications, resulting in many different hypertext systems. Engelbart's NLS/Augment [19] and Nelson's Xanadu [20] were the first approaches that proposed machines to relate pieces of information,. In NoteCards [21] editable information could be linked together by means of typed directional links, in Intermedia [22] the idea of the anchor came up, whereas in Microcosm [23], "generic links," available dynamically, were an important contribution to link formulations.

The evolution of navigational hypermedia systems continued with an arbitrary number of systems, each of them adopting different implementation strategies. However, as stated in the Dexter Open Hypermedia work [24], the basic node/link network structure is the essence of [navigational] hypertext. Moreover, fundamental to Open Hypermedia models like Dexter and Microcosm, the linking information is to be kept separate from documents to allow powerful link structures, such as bi-directional or n-ary links [18, 26, 27, 9]. As a result, the navigational hypertext can be viewed as a network of data containing "components" interconnected by relational "links" [24].

To support the navigational domain, many hypermedia researchers have adopted a fundamental entity that provides the node, link, anchor, and context abstractions [28, 24, 29, 9, 30]. A node provides a wrapper for an arbitrary resource and may have several anchors associated with it, whereas an anchor can be bound to several links

[27, 30]. A link of the navigational domain is defined as an association between anchors with its direction clearly stated [24, 27] and a context is a collection of object references (similar to hypertexts in Dexter [24] and to contexts in [31]), or even more specifically – as stated in HOSS [9] – is a set of link objects. The attribute/value pairs can be used to attach any arbitrary property and its value to a component.

3 Classification Domain: Taxonomic Hypermedia Systems

Taxonomic reasoning is a particular kind of reasoning task that deals with the comparison and classification of highly similar nodes, in which an analyst viewing one node thinks not in terms of linking it to another node, but of including it in or excluding it from a set of related nodes [12]. Taxonomic reasoning in combination with hypersets leads naturally to an a priori hierarchical, or tree-based, organization in the set of sets. The hierarchical structure is not a result of the data but is a way to structure the set of categories into which the artifacts are classified [32]. Therefore, users sort artifacts (the equivalent of nodes in graphs) into categories (sets) based on their characteristics. Different users (or different rules applied to the same nodes) may have different result sets for how the artifacts are partitioned [27].

In general, the task of taxonomic reasoning has the following characteristics [12]:

- The information objects being manipulated are highly similar to one another along some dimensional attribute(s), so much so that the question of how to categorize or organize is not always immediately obvious.
- Taxonomic reasoning develops descriptions of each item along a set of dimensions, but these dimensions are not fully defined when one begins the reasoning process.
- The basic activities that need to be supported to facilitate taxonomic reasoning are essentially set operations, such as sorting objects into sets based on their characteristics, looking together at the members of a single set, examining the different sets of which a single item is a member and generating new sets from old ones.

Eventually, the classification hierarchy that taxonomic reasoning entails is not a strict tree but rather a directed acyclic graph in which one node can have multiple parents. While a directed acyclic graph offers more navigational resources than does an arbitrary graph, a strict tree offers even more, including unique paths between nodes and thus unique distance metrics. A distance measure between artifacts is useful when we move beyond classification of artifacts to storing and retrieving them [32].

There are two basic routes to formalizing a classification hierarchy. The first one involves the lower-level sets that can be elements of higher ones while the second one comprises these sets as subsets of higher ones. The nodes in a classification hierarchy are of two very different sorts. On the one hand, some nodes allow the analyst to select a Perspective from which a given object is desired to be viewed. On the other hand, there are other nodes that require the analyst to sort the object into one of a set of disjoint Categories. Actually, *Categories* can be constructed in such a way that the structure remains a strict tree, and it is *Perspectives* that cause the hierarchy to deviate from a strict tree, since they permit a given artifact to descend from several branches [32]. Taxonomic perspectives, then, allow different views of the organizations of some set of nodes within a branch of hierarchy. Therefore, perspective can be regarded as an abstraction that is, in many ways, similar to that of “context” as used in some navigational hypertext systems [27]. Modeling perspectives, such as associations, not only has the interesting property of allowing users to move between the different views just as if you were following a link, but also provides a convenient way of choosing a view.

A taxonomic reasoning contains three main primary abstractions, the specimen, the taxon and the taxonomy. A specimen has arbitrary content and attributes and represents the elements of the given data. A taxon, on the other hand, has no content but has arbitrary attributes that form a constructed descendant. It can have three sides: supertaxa, subtaxa and specimens. More specifically, a subtaxa can contain an arbitrary number of elements while a supertaxa or a specimen consists of exactly one element. Finally, a taxonomy contains an hierarchy of specimen and taxon. Apart from the primary abstractions, there are some secondary ones such as annotation, posited specimen and delta calculation.

Generally, taxonomy has not been widely used in applications. However, the first application that used the notion of hierarchy and taxonomies was Hyperset [12]. Hypeset supported a full repertoire of set operations that generated new sets, including union, intersection, complement with respect to another set, and symmetric difference. It also provided editors with the ability to enter new artifacts and define new subsets. Another application is the TaxMan in HOSS [9]. It was particularly designed for addressing the problem of botanical taxonomic work. It used hierarchy in order to differentiate each species according to their characteristics and specified the degree of participation in each relationship between different species.

4 Information Analysis Domain: Spatial Hypermedia Systems

In the information analysis problem domain, the focus is on the process of structuring diverse material. Structuring and organization of information is a complicated intellectual process including activities such as collecting, comprehending and interpreting diverse types of information. During such activities, users tend to explore the structures they are creating and organize information by implicit and informal spatial methods formalizing incrementally their structures [33]. Traditional authoring systems were too restrictive to use for such tasks, because they required commitment to formal structure in very early stages. The issues of supporting construction of hypermedia structures through exploration and incremental formalization define the boundaries of the information analysis domain.

To address problems in the information analysis domain new systems emerged emphasizing on the support of the structuring process at the user level. Taking advantage of the human perceptual system as well as spatial and geographic memory [6], these systems allow relationships to be expressed by spatial proximity and visual cues. Such expression of relationships achieves differentiation of implicit from explicit structure hence permitting a smooth shift from informal to formal structure.

Hypermedia systems providing such paradigm to information organization are referred to as spatial hypermedia systems.

Primary entities in spatial hypermedia systems include objects, collections and composites each usually appearing with a different visual symbol having different visual characteristics such as shape, color, border, text font etc. Objects represent wrappers for arbitrary types of data. Object types are supported by means of explicit visual characteristics. Collections contain arbitrary amount of objects or collections. Hierarchical ordering of collections is thus possible. A set of objects or composites in particular visual configuration form composites.

Spatial hypermedia systems provide also visual and spatial means to denote relationships among entities. Consequently, these systems are able to recognize relationships by spatial arrangement, relationships by object type, relationships by collection and relationship by composite. A special structure-finding process – the spatial parser – is responsible for discovering such relationships and suggesting the creation of new collections and composites to the user.

Navigation in spatial hypermedia systems is accomplished through “zoom-in” and “zoom-out” operations know as semantic zooming [50][51] applied on collections. Zoom-in operation effects in filling the whole application window with the content of the affected collection thus revealing a new navigation space. Zoom-out operation on a collection, effects in revealing the navigation space of its parent collection.

Although initial concepts of spatial hypermedia where encountered in Notecards [21] and gIBIS [34], actual examples of spatial hypermedia systems include VIKI [6, 35],CAOS [36], and more recently, VKB [25].

5 Hypermedia Modeling

During hypermedia development the complex relations of subject specific information, for example museum applications or multimedia presentations, are captured in a clear and comprehensible way, making them accessible to the user. Users, often referred to as readers, come across hypermedia applications that enable them going through information by selecting paths to follow on the basis of interests emerging along the way [37]. Hypermedia systems provide environments to facilitate the creation of hypermedia applications. Although readers are unaware of how hypermedia applications are created, two other user groups are very much concerned about that issue; namely, hypermedia designers and authors. Designers, as well as authors, use services provided by hypermedia systems to create hypermedia applications engaging in an activity called hypermedia development.

Hypermedia development is a complicated cognitive process that consists of recursive activities, like emergence of ideas, representation and structuring of ideas, evaluation and update [38], requiring a mental model of the target application domain. As it has been recognized, it is this complexity that roots the problems of cognitive overload and disorientation in hypermedia [38].

Thus, much effort has been given in the creation of sophisticated tools and methodologies that will assist in developing hypermedia applications.

Towards the better support of activities related to hypermedia design, systematic and structured development of hypermedia applications have been proposed [39, 40, 29, 41, 42, 43]. These approaches differentiate between authoring-in-the-large, carried out by designers, aiming at the specification and design of global and structural aspects of a particular hypermedia application, and authoring-in-the-small, carried out by authors, referring to the development of the contents [44]. With regard to authoring-in-the-large, they provide convenient environments within the hypermedia system for hypermedia development, offering a suitable data model helping articulate structural designs. A rich set of abstractions including aggregation, generalization/specialization and the notion of constraints, facilitates the process of tailoring abstractions. Furthermore in these approaches, a hypermedia schema embodies structural designs leading to the notion of hypermedia modeling.

The above kind of modeling is useful to hypermedia development because it can help designers to avoid structural inconsistencies and mistakes, and applications developed according to a model will result in very consistent and predictable representation structures [29]. Flexibility in definition of hypertext semantics, schemas and authoring of specific documents, is an important prerequisite in order to support the authoring mental process and to avoid "premature organization" problems [45]. Additionally, hypermedia modeling provides a framework in which authors can develop hypermedia applications at a high level of abstraction. It helps them to conceptualize these applications via an instantiation of a schema without too much regard to structural details. In general, one of the major goals in hypermedia modeling is to provide authors with the ability to customize knowledge structures for their specific tasks.

6 Levels in Hypermedia: Common abstractions and Templates for Primitives

In the previous sections we have considered several domains in hypermedia that represent a particular set of practices which foreground a particular set of operations. Navigation foregrounds links; spatial hypertext foregrounds associations; taxonomy foregrounds set and (hierarchical) graph operations and so on. In each case, however, we can see that these domains also share operations at a fundamental level. The taxonomic domain needs both to support links (navigation) and associative layout (spatial hypertext) while navigation itself is enhanced by supporting notions from taxonomy of sorting and set making for the creation of user-determined composites. Indeed, we suggest that what characterizes these as specifically hypermedia domains, rather than simply representing a page layout program, a taxonomy program or data management tools is that they must each have access to these shared set of hypermedia-based services.

With so much overlap of core functionality, difference exists primarily at the domain level, and can largely be represented in terms of domain/service focus. We therefore require a way to conceptualize such services at several levels: at the domain level, at the service or middleware level, and at the fundamental architecture level. To this

end, our goal has been to develop both structural primitives for our CB-OHS system Callimachus and the middle layer templates. The primitives allow us to conceptualize the fundamental architecture operational structure processes and the middle layer templates support the representation of those processes as particular domain services.
In conceptualizing these structural primitives and service templates, we have found it useful to borrow the concept of *levels* from programming languages. The hypermedia domain specific abstractions, such as Taxonomic Hypertext or Spatial Hypertext are at the *highest* level. The structural elements, which each domain draws upon, are at the *lowest* level of hypermedia (we describe these in the following sections). High and low levels are determined by how suitable the abstraction is to solve a problem at hand. The domain level is therefore especially useful for highlighting possible contexts of use for combinations of services, such as aggregation, sorting, or association, and for determining any new primitives or combinations of primitives that must be made available to support these services.

Thus, while the structural element is not a suitable abstraction to solve navigational problems (due to its generality), we can, to continue to borrow from programming languages, *transform* or *cast* such elements into a higher level abstraction, based on what we can reason about the properties of a particular domain. To solve navigational problems, for instance, such casted elements would be the node, link, and anchor. To continue the programming languages analogy, we can consider the structural element of structural computing as the assembly level language of hypermedia and the domain specific abstraction as a high level language such as Pascal, Java or C. One can write in assembly to implement a Travelling Salesman Problem algorithm, but it is more convenient to write this in a higher level language. Moreover, we can use a higher level language like C to write new languages to solve in a more convenient fashion certain cases of problems, similar to the way PHP or Python provide effective language tools for crafting Web cgi applications.

By thinking in terms of structural properties in particular, and by working from the domain level to a well-defined set of primitives, we have a model for determining when or if new tools need to be built, and we have a ready-made middleware architecture for reuse of existing primitives or primitive composites for new services.

In the following sections, we do not claim to have defined the complete of core operations for hypermedia. To finish the programming languages analogy with relation to the concept of "computationally complete", the repertoire of core operation here is not "structurally complete;” one may come up with another set. That is our hope: that by presenting the reasoning we have defined for Callimachus primitives, others interested in this domain may use this logic for expanding/extending the structures and service templates. Indeed, a concern of structural computing in general is to define the assembly-level primitives for structural relations. A concern of CB-OHS’s in general and of Callimachus in particular, is to define the rules to create the templates that will function as a middle-layer, higher-level abstraction between the assembly level and the domain level. We touch on both these aspects in the rest of the following sections of this paper.

7 The Callimachus CB-OHS

The Callimachus CB-OHS attempts to provide the framework in which the aforementioned hypermedia domains co-exist and structure servers – providing new abstractions and services - for new domains can be developed. Given the complexity of development of structure servers in new hypermedia domains, special attention has been given in the provision of suitable tools to facilitate such task. Within Callimachus these tools are part of a *methodology* i.e. a systematic, disciplined quantifiable approach in the construction of structure servers.
One such tool are the *structure templates* of Callimachus. The aim of structure templates is to maintain the specifications of the structure model of hypermedia domains. Structure servers operate guided by these structure templates to provide domain specific abstractions and constraints. Figure 2 outlines the architecture of the Callimachus CB-OHS and the role of structure templates.

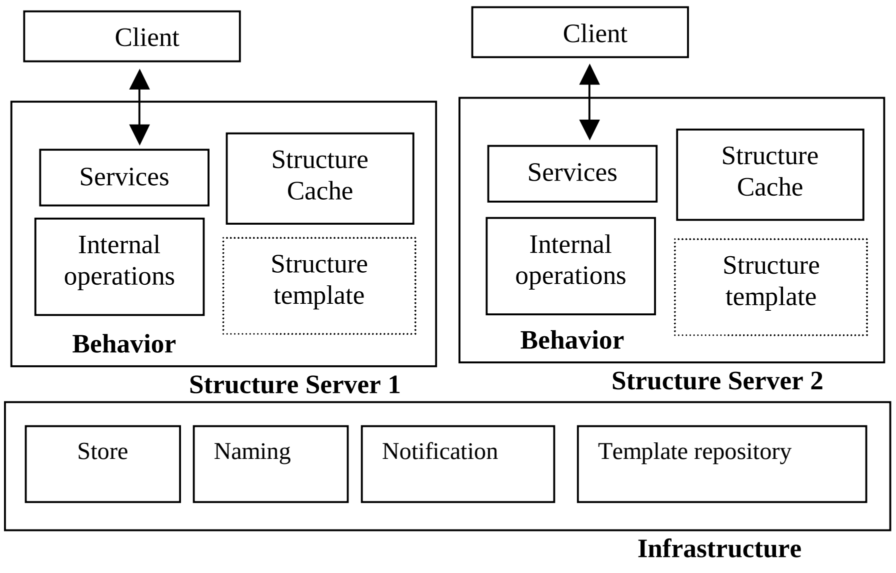


Fig. 1. The role of structure templates in the Callimachus CB-OHS

In the following we briefly illustrate the various parts of the architecture:

- *Structure server*: It consists of a structure model (structure template), structure cache and a set of behaviors that are used to deliver domain, sub-domain and application specific abstractions and services to clients.
- *Structure template*: The formal specification of the abstractions that define a hypermedia domain, sub-domain or application.
- *Structure Cache:* Provides an intermediate store for domain specific abstractions for structure servers. Inspired by [16], behaviors operate upon abstractions that have to be first loaded from the store into the structure cache.
- *Behavior*: Behavior models the computational aspects of a domain, sub-domain or application. Behavior can be divided – depending on its purpose – into two main categories: *services*, which are available to clients through the use of a specific API and protocol (e.g., openNode, followLink, etc.), and *internal operations* that are used by the structure server internally for consistency reasons mainly (e.g., to affirm conditions and constraints or to interpret abstractions in a suitable manner).
- *Infrastructure*: This includes the fundamental hypermedia functionalities that are available to all other entities. Functionalities, such as naming [14], persistent store and notification, constitute an essential part of the infrastructure.
- *Template repository*: A storage place for structure templates. Its main purpose is to hold structure definition as well as to support reusability and extensibility of structure among structure servers.
- *Client*: Any process that requests hypermedia functionality. Clients request hypermedia operations from one ore more structure servers with the use of the appropriate APIs.

The methodology of developing structure servers within Callimachus, can be divided into two phases: the *structural design phase* and the *behavioral design phase*.

During the structural model design phase, the abstractions of the target hypermedia domain as well as their complex interrelationships are captured and formally specified. Structure templates provide the mean to formalize such specifications. Devising structure specification formalism is a difficult task since it should be convenient to use it without compromising expressiveness. The formalism should be open to extensions, model-neutral and provide a common ground for cooperation. Such formalism deployed within the Callimachus framework is described in section 9.

During the behavioral design phase, the computational aspects of structure servers in Callimachus are specified and developed. Such computational aspects indicate how structure servers function in order to realize a particular behavior e.g. following a link or affirm conditions. Although the specification of behavior is not in the scope of this paper we present a set of core operations upon which behaviors of structure servers can be built.

8 Structure Elements

In Callimachus, the methodology for defining the structure model of a domain, sub-domain or application consists of the specification and inter-relation of *structural types*. A structural type is either an instantiation of a basic abstract class, the *Abstract Structural Element (ASE)*, or a specialization of another existing structural type. The notion of the *ASE* is inspired by the *structure object*, proposed by Nürnberg [9] and is similar to the *Object* abstract class in the majority of the object-oriented programming languages. Structure servers operate on structural objects. The set of structural types and the relationships among them defined by the model-designer at the same context, establish the *template* of the domain, sub-domain or application. During the definition of a new structure model, already existing templates may be reused or extended.

In the following subsections we describe the elementary constructs of the methodology, as well as the structure-model definition process.

8.1 Elementary Constructs

The *Abstract Structural Element (ASE)* is a meta-class that consists of two attributes: an identifier and a name. The instances of ASE are *structural types* that are identified by a system-defined unique *structural type identification* (*sid*) and a designer-specified unique *name*. A structure type may have an arbitrary number of *properties* and *endsets*. The instances of a structural type are *structural objects* in the sense described in Nürnberg [9] but they are constrained by their structure type specification, regarding the endsets and properties. Each structural object is identified by a system-defined unique *structural object identification* (*oid*). Figure 3 depicts an instance of an abstract structural element.

The structure type properties are specified by a name, a data type and they may be *single-valued* or *multi-valued*. A special property, named *content*, may be defined in the context of a structural type. At the instantiation level, the value of this property is an address specification of an arbitrary amount of data. Since our purpose is to specify structure, not data, we do not define other semantics for that property (like within-component layer in the Dexter hypertext reference model [5]).

The *endset* of a structural object is a placeholder for oids of other structural objects. It is the most significant construct since it enables the grouping of related objects. An endset of a structure type has the following attributes:

- *Name*: unique among the endsets of the same structural type,
- *Order number*: an integer that designates the sequence of the endsets of a structural type,
- *Total participation indication*: a [min, max] notation that indicates the minimum and maximum number of oids that can participate in the endset.
- *Multiplicity*: a [min, max] notation that indicates the minimum and maximum number of occurrences of that endset that can participate in a structural object of that type.
- *Configuration*: whether or not the members of the endset bear a particular functional or structural relationship to one another (e.g. may form a set, list, queue etc).

Along an endset, a set of the structural types (sids) is defined, called *s-set*, which configures the structure types whose instances may be end-points of the endset. An s-set may be *inclusive* or *exclusive*; an inclusive s-set specifies the allowed structure types, while an exclusive one specifies the forbidden structure types. A *partial participation indication* may be defined for each structure type in an inclusive s-set, indicating the minimum and maximum number of objects of this type that can participate in the endset.

In addition, for each structural type in the s-set, a *cardinality indication* is specified (a [min, max] notation can be used for that) defining the number of structural objects (of the structural type that is currently defined) to which a structural object of that type is allowed to be related (through that endset). This is similar to the cardinality ratio ([4]) in the E-R diagrams that specifies the number of relationship instances that an entity instance can participate in.

As a last notation, an endset with more that one structural types in its s-set, may be characterized as *homogeneous* or *heterogeneous*. At the instantiation level, a homogeneous endset can include structural objects of one type only, while a heterogeneous endset may point to structural objects of different types at the same time.

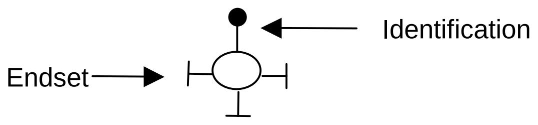


Fig. 2: Structural Element with 3 endsets and identification.

8.2 Operations on Abstract Structural Elements

In Callimachus structural objects are controlled by a core set of operations. This set of operations form the basis for the specification of the computational aspects of structure servers:

- **createStructuralObject() :** Instantiates a new untyped structural object. A unique system identification is given to the created structural object.

- **getType(s) :** Returns a string indicating the structural type of structural object *s*.

- **addEndSet(s, endSetName) :** Adds a new endset named *EndSetName* to the structural object *s*.

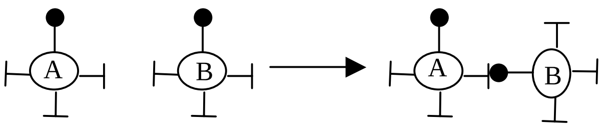
- **addEndSetAttribute(s, endSetName, endSetAttribute) :** Adds a new attribute named *endSetAttribute* to the endset named *EndSetName* of the structural object *s*.

- **setEndSetAttribute(s, endSetName, attributeName, attributeValue):** Sets the value of the attribute named *attributeName* of the endset *EndSetName* to the value *attributeValue* of the structural object *s*.

- **addStructuralObjectAttribute(s, attributeName) :** Adds a new attribute named *attributeName* to the structural object *s*.

- **setStructuralObjectAttribute(s, attributeName, attributeValue) :** Sets the value of the attribute *attributeName* of the structural object *s* to the value *attributeValue*.

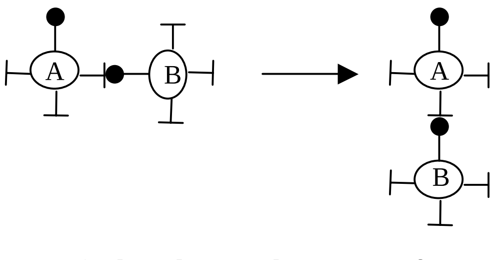
- **addEndSetMember(s, targetS, endSetName):** Adds the structural object *s* to the endset *endSetName* of the structural object *targetS*. This operation is graphically depicted as follows:



- **getEndSetMembers(s, endSetName):** Returns a list of the structural objects oids that are members of the endset *endSetName* of the structural object *s*.

- **filterEndsetMembers(s, endSetName, condition):** Returns a list of all structural objects that are members of the endset *endSetName* of structural object *s* satisfying the expression *condition.*

- **moveEndsetMembers(s, fromEndSetName, toEndsetName) :** moves all structural objects that are members of the endset *fromEndSetName* of structural object *s* to the endset *toEndsetName* of the same structural object *s*.



- **moveSingleEndsetMember(s, t, fromEndSetName, toEndSetName):** moves the structural object *t* that is member of the endset named *fromEndSetName* of structural object *s* to the endset named *toEndSetName* of *s*.

The store of Callimachus is able to *load*, *save* and *query* structural objects. It provides therefore the following operations:

1. **loadStructuralObject(id):** Loads a structural object given its unique identification *id* from the store to structure cache. Successful completion of this operation will result in creating a new structural object, setting the appropriate values of the attributes of the structural object as well as setting the members (oids) of all of its endsets.
2. **saveStructuralObject(s):** Writes the structural object *s* to the store.
3. **queryStructuralObject(s):** Returns a list of structural elements from the store that match the given structural element *s*. In this context, the structural object s2 is the result of the operation queryStructuralObject(s1) when the following conditions hold:
    a. The endset names of s1 are a subset of the endset names of s2.
    b. The attribute names of s1 are a subset of the attribute names of s2.
    c. The non-empty attributes of all endsets of s1 have the same values with the corresponding attributes of all endsets of s2.
    d. The members of each non-empty endset of s1 are a subset of the members of the corresponding endsets of s2.
    e. The s-sets of each endset of s1 are a subset of the s-sets of the corresponding endset of s2.

    The identification of the structural object *s* is not taken into consideration during the *queryStructuralObject* operation.

8.3 Customizing Structure Types

Structure types capture the semantics of domains/sub-domains/applications. Often it is useful to customize structure types of a domain in order to support the structure requirements of a sub-domain or an application. One point of attention is the relation of the customized structure type to the original one and the ability to compare them. We claim that instances of customized types are in fact a subset of the instances of the original type. In this way, compatibility is established between the two types, and instances of customized types can be used in places where instances of the original type are expected.

We define the *extent* of a structure type as the set of the possible instances of this type. The extent of a customized structure type is always a subset of the extent of its parent type, which means that it can take its place in applications without degrading their functionality. However, customization means different behavior. Customized types should support original behavior and extend it in order to accommodate additional functionality.

Within Callimachus, customization of structure types is modeled using the *is-a* abstraction, which may connect two structural types to express that the one is a specialization of the other. The customized type inherits all the properties and endsets of the parent type. Moreover, new properties may be defined and the various constraints of the endsets (participation indication, s-sets, cardinality indication, homogeneity-heterogeneity, multiplicity) may be restricted. Nevertheless, it is not allowed to add or eliminate endsets, since this will change the physiognomy of the structure type.

8.4 Templates

As a *template* we define the container of the structure types specifications that model a domain, sub-domain or application. A template is modeled itself as a structural object. More specifically, a *template-class* is a system-defined structure type that consists of a single endset, with the purpose to group the sid(s) of one or more designer-specified structure types. When the designer specifies the structure types to define a structure model, in fact he creates an instance of the template-class (a template) and assigns a *name* to it. Templates are stored in a special part of the storage infrastructure, called *template repository*.

A template may be created from scratch or by utilizing existing templates, stored in the repository. Each template may have a set of templates that inherits from. This set can be considered as an endset; the *uses* endset. All structure types described in the inherited templates are members of the *uses* endset.

A question that arises is how useful is such *multiple inheritance* functionality? We identify the following situations:

- A new structure model may be constructed that requires abstractions of two or more other structure models. The inherited structure types may be customized to express special needs (as long as the rules of the customization hold).
- A new structure type may be defined that *combines* structure types from different templates, i.e. enabling its endsets to point to the inherited structure types.

Templates could better be represented in a formal specification language, since they include various attributes and complex relationships. A visual representation would be easier to use, but it is not easy to visually express all the details of the specifications. XML is well suited for such a task, as it enables composition and arbitrary details. Furthermore, it is a well-known language and can be manipulated easily.

9 Structure Servers

As mentioned earlier, structure servers provide the basis for delivering hypermedia domain specific abstractions to clients. In particular, they operate on structural objects transforming them to useful abstractions and allowing clients to use them according to the domain specific restrictions. However, in order to operate upon a structural object it has to be first loaded into the structure cache. Templates provide structure servers with the necessary definitions of the modeling primitives with which structural objects are reified.

During their startup, structure servers load the template of the domain they support from the template repository into their structure cache. In Callimachus, structure servers are internally layered with respect to how structural objects are managed. Two layers can be identified: The Abstraction Factory Layer (AFL) which is responsible for reifying untyped structural objects to domain specific abstractions with the use of templates, and the Abstraction Utilization Layer (AUL) where the domain specific abstractions – once created – may be used by clients requesting hypermedia functionality.

Within Callimachus, clients are separate processes from structure servers, understanding how to communicate with them (i.e. request hypermedia functionality using the appropriate protocol). Moreover, clients are "aware" of structural objects and possess a structure cache where structural objects, which are sent back from structure servers (as a result of a request), are maintained.

9.1 Associative Hypermedia Domain: Modeling Navigational Hypermedia Systems using ASE

As outlined earlier, navigational hypermedia systems provide abstractions such as anchor, link, composite and context. The associative domain requires anchors to have one endset where arbitrary typed nodes can be contained and links with two endsets for source and destination nodes. Composites have only one endset keeping the oids of structural objects they aggregate. Context also has only one endset holding the links constituting it. For the navigational domain the following structure types could be defined in the repository:

```
<template name="Navigational" id=1 >

<structure_type name="node" endsets=0 id=2 />
<structure_type name="anchor" endsets=1 id=3>
     <properties>
            <property multivalued=0>
                   <name>LocSpec</name><type>string</type>
            </property>
     </properties>
     <endset name="nodes" configuration="set" >
            <s-set>
                   <member type=2 min=1 max=1/>
            </s-set>
     </endset>
</structure_type>

<structure_type name="link" endsets=2 id=4 >
     <endset name="source" configuration="set" >
            <s-set>
                   <member type=3 min=1 max=1/>
            </s-set>
     </endset>
     <endset name="destination" configuration="set" >
            <s-set>
                   <member type=3 min=1 max=n/>
```

```
               </s-set>
        </endset>
   </structure_type>
   <structure_type name="Composite" endsets=1 id=5>
        <endset name="nodes" configuration="set" >
               <s-set>
                      <member type=2 min=1 max=n/>
               </s-set>
        </endset>
   </structure_type>
   <structure_type name="Context" endsets=1 id=6>
        <endset name="links" configuration="set" >
               <s-set>
                      <member type=4 min=1 max=n/>
               </s-set>
        </endset>
   </structure_type>

   </template>
```

We will describe briefly the operations applied by structure servers upon structural objects in order to provide navigational services to clients. We will analyze the following operations: openNode, followLink. In Callimachus, these operations are expressed using the repertoire of operations applicable to structural objects as shown below.

9.1.1 Opening Nodes

The *openNode* operation of the navigational structure server is called every time users view a document. The task of the *openNode* operation is to locate the anchors associated with chosen document and making them visible to the user. A unique identification of the opened document (which might be its file name for example) is supplied to this operation.

Once the structure server receives an *openNode* request, it issues a *createStructuralObject()* operation in order to create a structural object which is in turn transformed to an object of type *anchor*. That operation occurs at the AFL where the structural object becomes a type according to the loaded template. A set of *addEndSet*, and *addStructuralObjectAttribute* as well as *addEndSetAttribute* operations are applied, shaping the newly created structural object at that level according to the definition of the anchor as it has been recorded in the template modeling the navigational domain. Typing of all structural objects follows such procedure. At the AUL, the structure server issues an *addEndsetMember* operation in order to add the identification of the document to the endsets named "Nodes" of the newly created anchor. The store is queried with the *queryStructuralObject* operation supplying as argument the configured anchor. The result is a set of structural objects of type anchor having in their endset named "*Nodes*" the specified document identification. In order to check which of the returned anchors are *source* anchors, for each returned anchor (i.e. structural object) the following occurs: a structural object of type "Link" is created adding to its endset named *source* the oid of the structural element that were determined by the previous step. The structure server issues then a *queryStructuralObject* query to the store supplying the created structural object of type “Link” as argument. These anchors for which that operation does return an empty result set are removed from the initial result list whereas the remaining anchors (structural objects) are send back to the client. The client is responsible for displaying them to the user.

9.1.2 Following Links

The followLink operation is one of the most important operations in navigational hypermedia systems since it is this operation that gives users the impression of navigating an information space. The followLink operation is issued to the structure server whenever users select anchors that are displayed in the active document and can be divided into two phases: (a) the destination anchor resolution phase where the destination anchors are determined, and (b) the node resolution phase where the nodes to which the anchors of the previous phase belong are computed. Various versions of implementing traversal behavior is possible depending on who (client or structure server) is initiating each phase: client may initiate both phases, i.e. query structure servers to compute the destination anchors which are send back to clients which in turn can initiate the node resolution phase or the structure server initiates both phases (as a result of the followLink message) sending to clients the destination anchors as well as the nodes to which these anchors belong. In any case, a set of structural objects will be sent back to clients and placed in their structure cache.

The oid of the structural object identifying the selected anchor is supplied to the followLink operation. Upon reception of the *followLink* message, the structure server issues a *createStructuralElement()* operation creating a new link object. The *addEndsetMember* operation is issued to insert the specified anchor oid to the endset named *source* of the newly created link. A *queryStructuralElement* operation will load into the structure cache a structural object of type link, having the specified anchor oid in its endset named *source*. Using the *getEndSetMembers* operation the members of the endset named *destination* will be returned. The list of destination structural objects oid is now determined. For each oid in the list a *loadStructuralObject* operation is performed loading the structural objects into the structure cache. Since these structural objects represent anchors, applying *getEndSetMember* to the endset *“nodes”* of each anchor structural object will result in a set of oids denoting structural objects of type *node* to which these anchors belong. Since the oids of the target nodes are determined, a *loadStructuralObject* operation loads the information of nodes (attributes and their values) into the structure cache. The list of structural objects representing anchors and nodes are sent back to clients that are now responsible for displaying the nodes content to the user (fig. 4). The accompanied attributes of the structural object of the structure type *node* supply the information for node content.

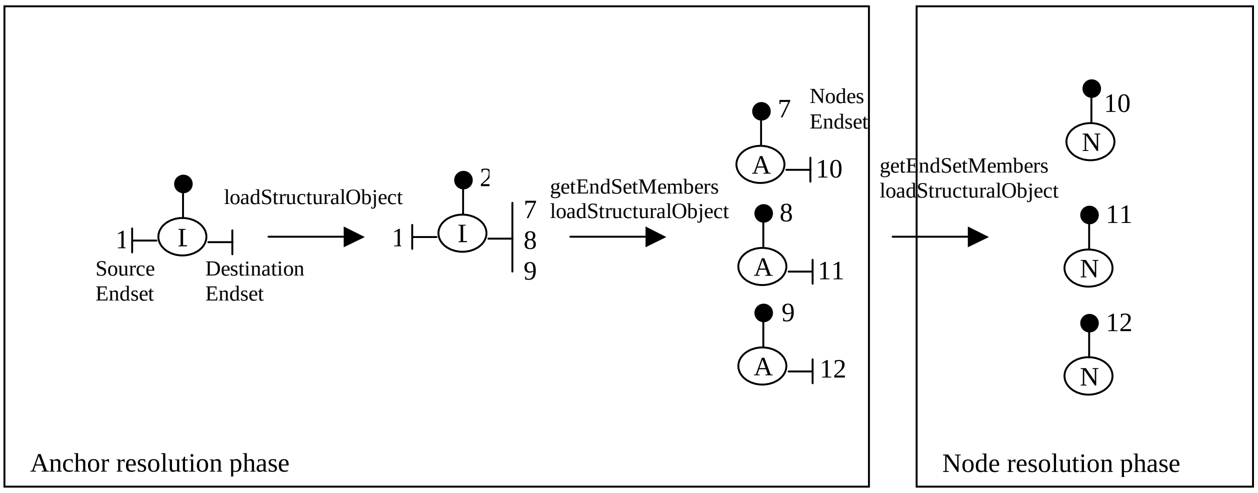


Fig. 3: Link traversal steps

9.1.3 Composites

"Following" composite structures differs significantly from following links within Callimachus. The difference is manifested in the outcome of applying traversal operations on each structure type. While following links will result in a list of structural element denoting anchors and their respective nodes, following a composite will result in one structural object having in its endset the structural objects that are part of that composite (fig. 5). These different outcomes would be reflected in the client's and structure server's structure cache. Thus, it would not be possible to model composite structures using the structure type *link*. As a result, in Callimachus, composites are not followed but rather opened, through a specialized service named *openComposite*.

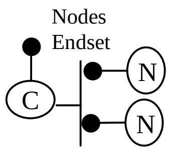


Fig. 4: Result of "following" a composite.

10 Related Work

Hypermedia systems providing general structure oriented facilities are just now emerging. Notably examples include HOSS [9], FOHM [50] and Construct [49, 52] that will be described within this section.

In recent years, HOSS is among the first paradigms that realize the philosophy of structural computing. The conceptual design of HOSS includes the concepts of structure and structural computation models. The structure model supports sidedness, direction, depth, status (n-sided composable structure with direction) as well as the guarantees of resolvability, nonrecursivity and accessibility. The structural computation model provides optional permanence, constancy and temporal invariance guarantees (definitions of these notions can be found in [9]). The basic element of HOSS is the *generic structure object* that may have content, arbitrary attribute/multi-value pairs, and arbitrary named and partitioned endsets. Customizing the structure object can specify various structural elements and masking away-unneeded functionality. In contrast to Callimachus, customization of structural objects to domain specific

FOHM is a fundamental open hypertext model, which identifies data and permitted operations in a client/server style architecture. FOHM formalizes data and operations through the use of an abstract machine that models them by transitions. Its main scope is to represent all the structural abstractions and support the operations needed for the navigational, the taxonomic and the spatial domain. Furthermore, FOHM focuses on interoperability between domains, e.g. following links between navigational and taxonomic structures. In contrast to Callimachus, FOHM does not provide developers with the ability to declare structural models. Thus, the development of new abstractions and operations upon them cannot be accomplished in a methodological manner.

The Construct CB-OHS explicitly supports the development dimension in CB-OHSs and endeavors to be fully compliant with the OHSWG standards. Construct provides development tools targeting the middleware layer of CB-OHSs allowing new hypermedia services to be developed and deployed within its environment. These services can be created from scratch, on top of or using other existing services. Through the use of UML diagrams services can be modeled and in conjunction with the Construct Service Compiler (CSC) service code skeletons can be produced. Nonetheless, generation of services is semi-automated since the developer has to fill in parts of the outcome. Callimachus and Construct share the need of providing mechanisms to facilitate the development of structure servers. However, each of them pays attention to different aspects of CB-OHSs. Namely, while Construct focuses on behavioral aspects of CB-OHSs, Callimachus converges on their structural aspects.

11 Conclusions

In this paper, we have presented the fundamental parts of the Callimachus CB-OHS and in particular we discussed how the structural model of a hypermedia domain is specified with the use of an atomic structural primitive called the structural element.

Moreover, we have introduced a set of core operations upon the structural element forming the basis for the development of structure server behaviors. By the use of the structural model specification and the core operations the development of structure servers can be accomplished, during the structural as well as the behavioral design phase, in a methodological manner.
OHSs in their move from domain-specific frameworks to cross-domain CB-OHSs, should clearly expose the *primacy of structure* and thus need the proper foundations and tools to handle structure as a first class entity. Towards such an evolution path, we outlined the implications of admitting first class structure in Callimachus. OHSs, in general, benefits from such structure handling facilities, mainly in the areas of reusability, interoperability and tool development.

Future work of Callimachus includes evaluating whether the structural primitives as well as the set of operations are sufficient enough for the development of structure servers in less studied domains, such as hyperfiction and workflow. Furthermore, attention will be given to the formal specification of behaviors in terms of a structure behavioral language that can be based on the core set of the operations presented in this paper. Finally, the efficiency of the Callimachus system will be thoroughly assessed. The generality of the provided facilities degrades performance of structure servers. Thus, explicit support is needed to optimize performance for each domain.

References

1. Anderson, K. M.: Software Engineering Requirements for Structural Computing, Proceedings of the 1st Workshop on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg (1999) 22-26.
2. Reich, S., Wiil, U. K., Nürnberg, P. J., Davis, H., C., Groenbaek, K., Anderson, K., M., Millard, D., E., Haake, J., M.: Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol, The New Review of Hypermedia and Multimedia, 5, 207-248, 2000.
3. Demeyer, S.: Structural Computing: The case for reengineering tools, Proceedings of the 1st Workshop on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg (1999) 27-29.
4. Elmasri, R., Navathe, S. B.: Fundamentals of Database Systems, 2nd edition, Benjamin/Cummings (1994).
5. Halatz, F. G., Schwartz, M.: The Dexter Hypertext Reference Model, Hypertext Standardization Workshop, NIST (1990).
6. Marshall, C. C., Shipman, F. M., Coombs, J. H.: VIKI: Spatial Hypertext Supporting Emergent Structure, Proceedings of the European Conference on Hypermedia Technologies (Edinburgh, Scotland, Sep 1994), 13-23.
7. Ntoulas, A., Stamou, S., Tsakou, I., Tsalidis, Ch., Tzagarakis, M., Vagelatos A.: Use of a Morphosyntactic Lexicon as the Basis for the Implementation of the Greek Wordnet, Proceedings of the 2nd International Conference on Natural Language Processing, Patras, Greece (2000).

8. Nürnberg, P. J., Legget, J. J.: A Vision for Open Hypermedia Systems, Journal of Digital Information (JoDI), Special issue on Open Hypermedia Systems 1, 2 (1997).
9. Nürnberg, P. J.: HOSS: An Environment to Support Structural Computing, Ph.D. dissertation, Dept. of Computer Science, Texas A&M University, College Station, TX (1997).
10. Nürnberg, P. J., Legget, J. J., Wiil U. K.: An Agenda for Open Hypermedia Research, Proceedings of the 9th ACM Conference on Hypertext and Hypermedia (1998) 198-206.
11. Nürnberg, P. J., Wiil, U. K., and Leggett, J. L.: Structuring Facilities in Digital Libraries, Proceedings of the Second European Conference on Digital Libraries (ECDL '98), Crete, Greece (1998).
12. Parunak, H.: Don't link me in: Set based hypermedia for taxonomic reasoning, Proceedings of the Third ACM Conference on Hypertext (San Antonio, TX, Dec 1991), 233-242.
13. Rosenberg J.: A Hypertextuality of Arbitrary Structure: A Writer's Point of View, Proceedings of the 1st Workshop on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg (1999) 3-10.
14. Tzagarakis, M., Papadopoulos, A., Vaitis, M., Christodoulakis, D.: A Name Service for Open Hypermedia Systems, Proceedings of the 1st Workshop on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg (1999) 30-33.
15. Wiil, U. K.: Multiple Open Services in a Structural Computing Environment, Proceedings of the 1st Workshop on Structural Computing, Technical Report AUE-CS-99-04, Aalborg University Esbjerg (1999) 34-39.
16. Dirk, B., L., Pedersen, C. A. and Reinert, O., H.: Cooperative Authoring using Open Spatial Hypermedia, Master Thesis, University of Aarhus, 1998.
17. Vanevar Bush. As We May Think. Atlantic Monthly, pages 101–108, July 1945.
18. George P. Landow. Relationally Encoded Links and the Rhetoric of Hypertext. In Hypertext'87 Proceedings, November 13-15, pages 331–343, Chapel Hill, North Carolina, USA, 1987. ACM.
19. Douglas C. Engelbart. A Conceptual Framework for the Augmentation of Man's Intellect. In P. D. Howerton and D. C. Weeks, editors, Vistas in Information Handling, volume 1, pages 1–29. Spartan Books, Washington, D.C., 1963.
20. Theodor H. Nelson. A File Structure for The Complex, The Changing and The Indeterminate. In Association for Computing Machinery Proceedings of the 20th National Conference, pages 84–100, 1965.
21. R. Trigg, L. Suchman, and F. Halasz. Supporting Collaboration in NoteCards. In CSCW 86 Proceedings, pages 153–162, Austin, Texas, USA, 1986. ACM.
22. N. Yankelovich, N. Meyrowitz, and A. van Dam. Reading and Writing the Electronic Book. IEEE Computer, 18(10):16–30, October 1985.
23. Andrew M. Fountain, W. Hall, I. Heath, and H. C. Davis. MICROCOSM: An Open Model for Hypermedia with Dynamic Linking. In Proceedings of the European Conference on Hypertext, pages 298–311, INRIA, France, 1990. Cambridge University Press.
24. F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. Communications of the ACM, 37(2):30–39, 1994.

25. Shipman, F.M., Hsich, H. Maloor, P. Moore, M. The Visual Knowledge Builder: A Second Generation Spatial Hypertext. ACM Proc of Hypertext 2001, Aarhus, Denmark: Aug. 14-18, 2001, 113-122.
26. George P. Landow and Paul Kahn. Where's the Hypertext? The Dickens Web as a System-Independent Hypertext. In ECHT '92 Proceedings, November 30 - December 4, pages 149–160, Milan, Italy, 1992. ACM.
27. Dave E. Millard, Luc Moreau, Hugh C. Davis, and Siegfried Reich. Fohm: A Fundamental Open Hypertext Model for Investigating Interoperability between Hypertext Domains. In Hypertext '00 Proceedings, pages 93–102. ACM, 1992.
28. S. Demeyer. ZYPHER: Tailorability as a Link from Object-Oriented Software Engineering to Open Hypermedia. PhD thesis, Vrije Universiteit Brussel, Departmnent Informatica, July 1996.
29. Franca Garzotto, Paolo Paolini, and Daniel Schwabe. HDM – A Model-Based Approach to Hypertext Application Design. ACM Transactions on Information Systems, 11(1):1–26, 1993.
30. S. Reich, U. K. Wiil, P. J. Nurnberg, H. C. Davis, K. Grnønbæk, K. M. Anderson, D. E. Millard, and J. M. Haake. Addressing Interoperability in Open Hypermedia: The Design of the Open Hypermedia Protocol. New Review of Hypermedia and Multime-dia, 1999.
31. J. J. Leggett and J. L. Schnase. Viewing Dexter with Open Eyes. Communications of the ACM, 37(2):76–86, 1994.
32. H. Van Dyke Parunak. Hypercubes Grow on Trees (and Other Observations from the Land of Hypersets). In Hypertext'93 Proceedings, pages 73–81, Seattle, WA, USA, 1993. ACM.
33. Catherine C. Marshall and Russel A. Rogers. Two Years Before the Mist: Experiences with Aquanet. In ECHT '92 Proceedings, pages 53–62, 1992.
34. Jeff Conklin and Michael L. Begeman. gIBIS: A Hypertext Tool for Team Design Deliberation. In Hypertext '87 Proceedings, pages 247–251. ACM, 1987.
35. Catherine C. Marshall and Frank M. Shipman III. Spatial Hypertext: Designing for Change. Communications of the ACM, 38(8):88–97, 1995.
36. Dirk Bucka-Lassen, Clauss A. Pedersen, and ´ Olavur H. Reinert. Cooperative Authoring using Open Spatial Hypermedia. Master's thesis, Aarhus University, Computer Science Department, December 1998.
37. J. Nielsen. Hypertext and Hypermedia. Academic Press, Inc., 1990.
38. J. Nanard and M. Nanard. Hypertext Design Environments and the Hypertext Design Process. Communications of the ACM, 38(8):49–56, 1995.
39. Catherine C. Marshall, Frank G. Halasz, Russell A. Rogers, and William C. Janssen Jr. Aquanet: A Hypertext Tool to Hold Your Knowledge in Place. In Hypertext '91 Proceedings, pages 261–275. ACM, 1991.
40. J. Nanard and M. Nanard. Using Structured Types to Incorporate Knowledge in Hypertext. In Hypertext '91 Proceedings, pages 329–343. ACM, 1991.
41. Tomas Isakowitz, Edward A. Stohr, and P. Balasubramanian. RMM: A Methodology for Structured Hypermedia Design. Communications of the ACM, 38(8):34–44, 1995.
42. Daniel Schwabe, Gustavo Rossi, and Simone D. J. Barbosa. Systematic Hypermedia Application Design with OOHDM. In Hypertext '91 Proceedings, pages 116–128. ACM, 1996.

43. Weigang Wang and Roy Rada. Structured Hypertext with Domain Semantics. ACM Transactions on Information Systems, 16(4):372–412, 1998.
44. F. Garzotto, P. Paolini, D. Schwabe, and M. Bernstein. Tools for designer. In E. Berk and J. Devlin, editors, Hypertext/Hypermedia Handbook, pages 179–207. McGraw Hill, 1991.
45. Athanasios Papadopoulos, Michalis Vaitis, Manolis Tzagarakis, and Dimitris Christodoulakis. A Methodology for Flexible Authoring of Structured Hypertext Applications. In ED-MEDIA/ED-TELECOM ’98 Proceedings, Freiburg, Germany, 1998.
46. Kenneth M. Anderson, Structural Computing Requirements for the Transformation of Structures and Behaviors, 2nd Workshop on Structural Computing, San Antonio, TX, 2000, pp 140-146.
47. Nürnberg, P. J., Leggett, J. J and Schneider, E. R., As we should have thought, Proceedings of the Hypertext '97 Conference. *(Apr 6-11). Southampton, UK 1997.*
48. Wang, W. and Fernandez, A. : A Graphical User Interface Integrating Features from Different Hypermedia Domains, In proceedings 3rd Workshop on Structural Computing, Aarhus, DK, 2001.
49. Wiil, U., K.: Using the Construct Development Environment to Generate a File-Based Hypermedia Storage Service, In proceedings 2nd Workshop on structural computing, San Antonio, TX, 2000.
50. Herot, CF, Carling, R, Friedell, M. Kramlich, D. and Rosenberg, R.L. Overview of the Spatial Data Management System, Tech. Rept. CCA-81-08, November, Computer Corporation of America, 1981
51. Furnas, G.W. and Bederson, B.B. Space-Scale Diagrams: Understanding Multi-Scale Interfaces, ACM, Proc. CHI ’95, 234-241.
52. Wiil, U., Nürnberg, P., Hicks, D. and Reich, S.: A Development Environment for Building Component-Based Open Hypermedia Systems, Proceedings of the Eleventh ACM Conference on Hypertext and Hypermedia, San Antonio, TX, pp.266-267.