# An Extensible Data Model for Hyperdocuments

Paul De Bra, Geert-Jan Houben
*Dept. of Mathematics and Computing Science, Eindhoven University of Technology*
*PO Box 513, 5600 MB Eindhoven, the Netherlands*
e-mail : debra@win.tue.nl, houben@win.tue.nl

Yoram Kornatzky
*Dept. of Mathematics and Computer Science, Ben-Gurion University of the Negev*
*PO Box 653, Beer-Sheva 84105, Israel*
e-mail : yoramk@bengus.bgu.ac.il

## Abstract

We present an extensible data model for hyperdocuments. It is intended to serve as the basis for integrating hypermedia systems with other information sources, such as object-oriented database management systems, information retrieval systems, and engineering CAD tools. Hyperdocuments are described by means of a small number of powerful constructs that integrate their structural and behavioral aspects. The different instantiations and combinations of these constructs yield an open class of hyperdocuments. Nodes, anchors, and links are all considered first-class objects and modeling constructs are applicable to all of them. These constructs permit a description of the multiple levels of functionality of an object within a hyperdocument, and the packaging of the different views of an object. Composite objects range over an extensible collection of structures including networks, sets, time-lines, and three-dimensional space CAD models.

## 1 Introduction

Current hypermedia systems are, to a large extent, closed application-specific systems catering for small sets of documents and supporting only limited classes of such documents. In particular, the standard graph model imposes a "tyranny of the link" [9] that limits hyperdocument structures to directed graphs made of information chunks represented by nodes, and links representing simple relationships between these nodes. Recent hyperdocument models have considered a richer set of structures including relationships (N-ary links) [15], sets [30], and Petri nets [23]. Other models have included composite nodes, which are themselves graphs of nodes and links, in order to model hierarchical structures of information [4, 7]. These models were motivated by knowledge representation, authoring, and reading concerns which demand a larger class of structures for representing interesting information in a natural way. Industrial applications of hypermedia systems [16] also point to the need for a much larger expansion of the hyperdocument model. In an industrial environment hypermedia systems would serve as integration tools binding together the information from a heterogeneous collection of sources such as information retrieval systems, database management systems, expert systems, and CAD tools. Tufte [27] discusses a rich variety of visualizations of complex scientific and engineering information, including time charts, maps, and distance tables. Existing hyperdocument models ignore this rich variety of composite objects. They hide the semantics of this varied class of information structures from the hypermedia system and limit its ability to serve as integration tool.

One could approach the challenge of enlarging the scope of hypermedia systems, and in particular the need to turn them into open systems, by successively adding new modeling constructs capturing a series of increasingly larger classes of hyperdocuments. However, any fixed class of structures is bound to be found limiting. The apparent contradiction between having a fixed set of modeling constructs and achieving open-ended extensibility, is one of the major problems to be solved in order to turn hypermedia into a true integration technology. We suggest an alternative approach to the development of an extensible data model for hyperdocuments. The model is based on a small set of generic modeling constructs that can be freely combined to define hyperdocuments. The generic nature of these modeling constructs is based on a separation of concerns between the fixed aspects of a hyperdocument that would always be present, and the variable part that is extensible. Extensible database systems for advanced applications have demonstrated the feasibility of isolating a few generic modeling constructs from which different actual structures could be built.

Accordingly, our data model is made of two layers. The lowest layer defines the kinds of first-class objects in a hyperdocument. These objects are nodes, links, and anchors. Each kind is an open-ended collection of types defining the variety of the basic forms of these objects. The second layer is that of modeling constructors that build complex information representations out of simpler ones. These constructors correspond to three ways in which complex objects can be built from their components.

- The **composite object** constructor builds a composite object from its elements. For example, a composite node, as used in recent work, is built using a composite network constructor from sets of nodes, links, and anchors.
- The **tower** constructor packages together the multiple levels at which an object within a hyperdocument is described. These levels would include, among others, a structural description level, and a visual presentation level.
- The **city** constructor binds together multiple views (perspectives) of an object. Examples of different perspectives for nodes representing program modules would be the source code and the object code for every module.

The three constructors can be applied independently and hence we would have for example cities of towers (hence the name city) which define multiple views of the same object where each view includes several layers of description of the same object. Note the difference between towers and cities: a tower gives different information about one object, whereas a city gives different descriptions of basically the same information.

Our data model is based on the object-oriented programming paradigm. Hence, the entities forming a hypermedia application are objects with a state and an associated behavior. While the object-oriented paradigm is a useful approach for application analysis and design, experiences with designing actual applications in object-oriented languages have suggested that one needs more than just a library of classes. With only a library tailored for a specific application area one does not obtain a coherent framework for application design and construction. An essential part of such a design approach would be modeling constructs (often called frameworks or clusters) that describe how individual classes and instances are tied together. Our data model provides such constructs for hyperdocuments in order to serve as a foundation for a high-level design method for hyperdocuments.

## 2 Structural Modeling Constructs

We start by identifying three fundamental kinds of objects : nodes, links, and anchors. Their definition is based on the hypermedia philosophy as outlined in many available hypermedia models. The set of classes of these kinds of objects depends on the application and our model is extensible with respect to the collections of classes employed by the application. The modeling constructs that we will introduce are applicable to all three kinds of objects.

### 2.1 Nodes, Links, and Anchors

The first kind of object is the **node**. A hyperdocument is an interlinked collection of information chunks. These chunks of information are modeled by nodes.

**Definition 2.1** *A node is described by a couple* $(i, v)$*, where* $i$ *is the object's identification and* $v$ *is the node's value (its contents or state).*

We will discuss later what kinds of values we allow, and thus what kinds of information chunks can be represented.

The connections between the nodes are modeled by **links**.

**Definition 2.2** *A link is described by a quadruple* $(i, n, n', v)$*, where* $i$ *is the object's identification,* $n$ *is the link's source anchor,* $n'$ *is the link's destination anchor, and* $v$ *is the link's value (its "meaning").*

An **anchor** describes the attachment of links to nodes, by defining the part of a node related by the link to other nodes. Selecting this specific part in the source node enables the reader to follow the link to one of the related nodes.

**Definition 2.3** *An anchor is described by a couple* $(i, v)$*, where* $i$ *is the object's identification and* $v$ *is the anchor's value (its location in a node).*

Having anchors as first-class citizens allows for more complex links than just links relating entire nodes. Moreover, this allows us to apply our structuring constructs described below to anchors, resulting in a more powerful hyperdocument model than is usually available. Separating the anchor information from the link

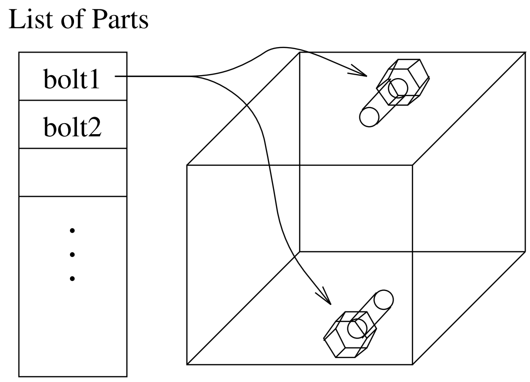


Figure 1: A 3D space composite object.

also has the practical advantage of isolating the linking mechanism from the details of the connected nodes [19]. Such isolation is important in a hyperdocument model intended to support the integration of information from a heterogeneous collection of sources.

**Definition 2.4** *Objects can be nodes, links, or anchors. A value of a node, link, or anchor object can be : a basic value (chosen from application-dependent classes), a composite value, a tower value, or a city value.*

Basic objects are nodes, links, and anchors having a basic value. The latter three kinds of values are discussed in the next subsections, as they are used to define complex higher-level objects.

The value of a node models an information chunk which, depending on the application, can be either simple such as a byte string, or complex such as a form. Thus, we consider an extensible collection of values for the **basic nodes**. The contents of a basic node are opaque to the hypermedia system and are manipulated through node-specific methods (e. g. specialized editors).

A **basic link** is a simple relationship between two nodes in a hyperdocument specifying how one information chunk is connected to another one : this connection could, for example, model an order of reading as intended by the document's author. The value of a link specifies the type of the relationship between the nodes. For basic links the values are chosen from a given domain of relationships.

A **basic anchor** representing the location of a link's endpoint in a node can be either of a familiar type such as a position within a text node, or a location within a more complex node such as an engineering 3D model (see Fig. 1), which is a 3D coordinate.

## 2.2 Composite Objects

A composite object abstracts a collection of objects into one higher-order object. Thus, a composite object models a complex unit of information built from simpler information objects. Existing hyperdocument models include only a limited notion of composite objects [4, 7, 25] : a hyperdocument graph built from a collection of nodes, links, and anchors. However, just as composite information chunks arise naturally, leading to the need for composite nodes, there are many examples of composite links and anchors, and a powerful hyperdocument model should account for them too. First, observe that to capture a general notion of relationship between the objects modeled by nodes one needs N-ary links ([11]). Moreover, a relationship between objects captured by a link may possess an internal structure. For instance, in a system in which the basic nodes are (verbal and factual) statements and the basic links are logical implications between statements, a proof of a theorem can be viewed as a composite link from a set of facts to a conclusion. The theorem is a composite link which consists of both links (logical implications) and nodes (intermediate facts). A link that relates several parts of a node to other nodes requires a composite anchor binding all these different parts. For example, a link from a part list to points in a design where that part is used may require a composite anchor whose elements are the points in the design where that part is used (see Fig. 1).

To represent these different kinds of complex information objects, we introduce the notion of a **composite object**, representing a complex unit of information having an internal structure. A composite object is built using a **composite constructor** that builds a composite object from collections of nodes, links, and anchors.

**Definition 2.5** *A composite object is an object with a composite value. A composite value is a 4-tuple $(c, n, l, a)$, where c is composite constructor, and $n, l, a$ are the sets of nodes, links, and anchors, respectively, that are used by the constructor to define the composite object.*

The composite constructors that can be used depend on the application, and we next discuss their nature.

Composite nodes provided by existing hyperdocument models are essentially graphs. However, tools in an engineering environment for instance use a wider class of complex objects besides graphs. CAD tools use 3D object models composed of elements positioned within a 3D space. Fig. 1 shows a a composite node which is a 3D-drawing whose set of nodes include two nodes representing bolts. Object-oriented database management systems [1, 14] have an extensible class of composite objects representing collections of data including sequenced structures such as arrays and lists. One should also note that a rich variety of visualizations of complex information such as maps, time-lines, 3D coordinate systems, and timetables are employed in engineering and scientific practice [27]. Integrating these sources of information within a hypermedia system implies that the system should be able to regard all these kinds of composite structures on an equal basis with the familiar network structure. The alternative approach taken so far by hypermedia systems is to regard these complex structures as the contents of basic nodes and relegate their manipulation to node-specific tools. Encapsulating these structures within basic nodes has severe conceptual and practical disadvantages. First, observe that the basic hypermedia notions of a hyperdocument as an interlinked collection of information chunks, and the concept of information access through navigation between nodes across links, are very natural concepts with which to describe the structure and access methods for such complex chunks of information. Thus, for the engineer manipulating such 3D space models through a hypermedia system, there is little conceptual distinction between navigation among elements of a composite object across links, and between moving through that 3D space while examining different elements of the design. Both these activities are most naturally modeled through a unified notion of a composite information object in which simpler information objects are placed. Practically, unifying all these different structures as a single notion of a composite object allows the development of extensible tools for representing and manipulating hyperdocuments.

Accordingly, we define a composite object as built from a collection of objects using a **composite object constructor**. Such a constructor defines the structure of the ensemble of information. For instance, a three-dimensional space constructor defines a composite piece of information having the structure of a three-dimensional space (see Fig. 1). Such a general structure demands a description of how component objects are located within that structure, for example a coordinate in the 3D space. Additionally, a composite object constructor includes global information and constraints on the collection's elements. For example, a network composite may include a constraint demanding the topology of the network to be a DAG.

**Definition 2.6** *A composite object constructor is a 6-tuple* $(k, s, co, lc, cs, gi)$*, where:*

1. $k$ *is the kind of object (node, link, or anchor);*
2. $s$ *is a structure constructor;*
3. $co$ *are the components given by* $a_1$ *:set of* $T_1$*,* $a_2$*: set of* $T_2$*, ...;*
4. $lc$ *are the locations of the components;*
5. $cs$ *are the constraints; and*
6. $gi$ *contains global information.*

As can be seen from the definition a composite object as a whole is more than the sum of its parts. Intuitively, a composite object can be viewed as a template containing holes into which the components are plugged. The structure of the template is defined by the structure constructor. Note that a composite link also has an internal structure containing nodes, links, and anchors, organized in a complex structure (which is not necessarily a network). The link global information with respect to such a link would include a global directionality. Note that a composite object may be included anywhere that a basic node can appear. So, components of a composite node may themselves be composites.

The most familiar kind of a composite node is a hyperdocument which is a network over a given collection of nodes, links, and anchors. Such a node is built with a network constructor, whose inputs are the nodes, links, and anchors of the network, and possibly also a constraint on the topology of the network:

$(k : node, s : network,$
$co : (n : set\ of\ nodes, l : set\ of\ links, a : set\ of\ anchors),$
$cs : topology - constraint)$

A familiar example that is encompassed in this definition are webs in Intermedia [28]. These are collections of links that are overlaid over a common set of nodes of the same document. A web is hence a composite node defined by means of a web constructor from a set of links (with anchors), but having no set of nodes as a component. A document is then a set of nodes, and applying a web to a document means creating a new document from the original one together with the web. Overlaying a web over a document may involve an enforcement of a constraint that all links are from and to nodes in the document. So, the web constructor should include this integrity constraint.

Novel kinds of composite objects that were so far regarded as the contents of basic nodes can now be explicitly modeled. For instance, we can formalize time-lines as built with a time-line constructor.

Our generalization of the notion of a composite object within a hyperdocument is powerful enough to encompass previous models which seem to be based on entirely different notions than those of the "standard" hyperdocument model. For instance, the hypermedia model based on sets suggested by Parunak [30] is defined by means of sets overlaid over a universe of nodes. This model can be defined within our framework by having a composite set constructor whose location information with respect to nodes contains a reference (identifier) to the nodes in the intersection. Links between sets are defined with binary links having two composite anchors. The elements of the two anchors are the nodes in the intersection of the two sets.

### 2.3 Towers

The functionality of the objects within a hyperdocument is usually multi-dimensional, and hence a complete description of an object within a hyperdocument usually consists of several levels. For example, a node has a structural dimension consisting of its contents (e. g. text) and operations to manipulate them (e. g. a text editor), and a presentation level describing their graphical rendering on the screen in terms of presentation attributes and methods. A third dimension or level is that of attributes attaching a semantic role to a node, e. g. issue or decision. Time-dependent nodes such as video have a temporal dimension defining the schedule of their presentation [17]. These different dimensions correspond to different levels of description of a hyperdocument and are defined in terms belonging to different conceptual spaces: the different levels address different information. All these levels of description are necessary in order to fully capture the functionality of an object within a hyperdocument.

We suggest a new modeling construct called a **tower** in order to bind together all these levels of description.

**Definition 2.7** *A tower object is an object with a tower value. A tower value is a mapping from a set of labels to a set of objects. Each label is a description level, corresponding to a separate level of the hyperdocument functionality. The domain for the function is fixed for each class of tower objects. If the levels are always given in a specific order, we may represent a tower as a tuple of objects.*

An example of a tower for a text node is shown in Fig. 2.

All kinds of objects are modeled by towers. Thus, a link tower might consist of a structural level maintaining the link's set of anchors and an indication of its direction, a presentation level indicating that it will be drawn as an arrow on the screen with a particular color and appearance, and a semantic level indicating the meaning of the relationship between the source tower and the destination tower.

An anchor tower would include a description of its location in the node to which it is attached and its graphical rendering on the screen. An anchor for an N-ary relationship link [11] would also include a a semantic level giving the role of the attached node in the relationship.

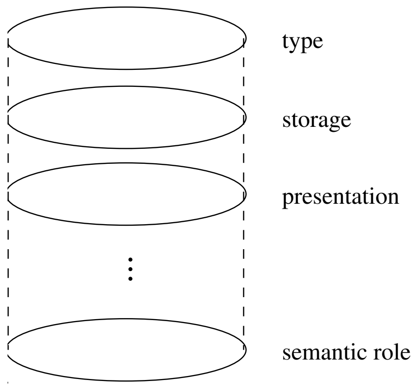


Figure 2: A tower for a text node.

We permit complete flexibility in the number and nature of the levels of different kinds of towers. Thus, one can have towers with only a graphical rendering level. These may be useful to describe graphical overviews of documents where the contents of individual nodes are not part of the overview. Such tailorability of the tower structure permits the integration of information from outside the hypermedia system.

A tower of a composite object, called a **composite tower**, packages together the multiple levels of description of a composite object. As a composite object is a collection of objects which would be themselves towers, a level of the composite tower is built from the corresponding levels of the elements of the composite. We have previously motivated the need for composite objects of all kinds and of a variety of structures but our discussion was purposely not targeted to any particular level. Now, that we have introduced the tower construct, it is natural to apply it to composite objects. One can observe that corresponding to composite structures arising at the structural level of a composite object, there are composite descriptions at the other levels, which are just as varied as its structural level. Accordingly, we would need presentation constructors (e. g. menu constructors) that build a composite presentation from given elements, besides the structural constructors (e. g. tables) that build a composite data structure from its elements.

**Definition 2.8** *A* **composite tower constructor** *is a mapping from a set of labels to a set of composite object constructors. Each label is a description level, corresponding to a separate level of the composite object functionality. A composite constructor builds a composite object of the type of its level from a set of elements (coming from the corresponding level in the towers of the elements of the composite object).*

### 2.4 Cities

An object in a hyperdocument is often viewed from different perspectives depending on the way it is accessed. Thus, an animation node might be viewed as an animation film by a link referring to it from the text that it illustrates, and as a code sequence when accessed from the editor. Similarly, a hyperdocument is often divided into multiple conceptual spaces. For example, in an information retrieval application, information can be divided into two related but separate collections : a base set of cards, and an index representing a collection of terms [3].

The important difference between the city concept and the tower concept is the fact that the different elements of a city deal with a different use of an object: the information is basically the same, but it is styled differently for separate users in order to create separate user views. The different elements of a tower describe rather independent aspects of an object: the information deals with the same object, but in general the described aspects do not have anything in common except the fact that conceptually they describe the same entity.

The **city** constructor packages together the different views of a basic or composite object.

**Definition 2.9** *A city object is an object with a city value. A city value is a mapping from a parameter space into a set of tower objects. The parameter space would be either a set of names or a data type :*
*City: Parameter type* $\rightarrow$ *Tower type.*

The elements (in the range) of a city are called **views**. Each such view is a tower describing the object from a particular (user) perspective. These views modularize the information according to the different ways in which it may be accessed by different reader groups.

In simple cases these views correspond to different roles of the same object [21]. Thus, the different conceptual spaces within an information retrieval system can be modeled as two views in a city: a view including the collection of information chunks to be directly browsed, and a view consisting of the set of terms over which intelligent search can be performed [3].

While cities occur most often with nodes, they are also useful for links and anchors. Thus, a link into a source code node would be a city of two views. The first view links into the text of the source code, while the second defines an active link whose traversal causes an execution of that source code.

## 3 Behavioral Modeling Constructs

In this section we discuss the concept of virtuality. We discuss a language for the definition of virtual structures, and we consider the notion of browsing semantics.

### 3.1 Virtual Structures

Virtual structures [8], such as virtual nodes, links, or composites, are described not by explicitly specifying their components, but by a computation procedure that is activated whenever these objects are accessed. Many hypermedia systems support different kinds of virtual structures that are implicit in the form and the content of the nodes. Such structures include, for instance, navigation by query [26], implicit links from a word to its dictionary entry [31], or virtual hypermedia networks for which the structure is entirely constructed by the system from computed relations among nodes.

Observe that the different kinds of virtual structures correspond to the different modeling constructs that we have defined. For example, the presentation level of a node tower will often be derived from its structural level; e.g. a textual node is presented in a simple text window, whereas an image node is presented in an image window. Similarly, some views in a city will often be computed from others. Accordingly, we introduce the notion of *virtuality* as an orthogonal dimension of all modeling constructs. This means that a **virtual object** can be used anywhere a stored object can.

**Definition 3.1** *A virtual object is specified by supplying a function generating it (from other objects) instead of actually supplying it as a data object.*

The function supplied for a virtual object is activated whenever the object is accessed, creating a process to evaluate it and yielding the object. The language in which these functions are specified is independent of the structural modeling constructs.

Let us consider a few examples of virtual objects that can be defined with our different modeling constructs:

- A basic virtual link is defined by a predicate on the properties of the nodes it connects. For instance, for nodes representing time intervals in knowledge-base application, their relationships represented through links are virtual and specified by evaluating a predicate on the intervals.
- Our generalized notion of a composite object allows for a composite node which is a three-dimensional space in which various objects are located. This is a composite object whose constructor builds a three-dimensional space. This space could be virtual in which case its size and

the density of the elements within that space are computed from the size of the elements.

- Virtual views are often the most natural means for the definition of a city. In the example for computing cross sections of a 3D model, views can be parameterized by the cross section angle provided by the link to the city. It is most natural to regard that city as being built from a variable number of views. Formally, this means that the mapping: $Angle \rightarrow Cross\ section$, as used within the city, could be computed instead of being stored.

While the programming language used for the definition of virtual structures could be a general programming language, and hence independent of the other modeling constructs, it is desirable to use a specialized limited language integrated within the model. Experience with database systems has demonstrated the use of limited query languages (like the relational algebra and SQL) in which a useful class of programs can be expressed.

Accordingly, we suggest a limited language for the definition of virtual structures in our data model. The language will not define all virtual structures needed, but it can be used to define some of them: specifically, at the level of views within a city.

### 3.2 A Virtual Structure Definition Language

The virtual structure definition language is an FP-like language [2] made of two layers corresponding to the layering of structures into constructors and objects. The first layer is small set of functional forms that are higher-order operators (functions), constructing new functions (operators) from given ones. They abstract useful patterns for manipulating the modeling constructs we have defined. These operators are generic, meaning that they are applicable to all instantiations of a modeling construct. For example, the filter operator selects elements of a composite object according to a predicate, irrespective of the actual composite object constructor used to build the object. A second layer consists of operators specific to particular types of structures; e. g. set-theoretic operators for set composite objects [30].

The first three generic operators that we present are applicable to the values of the objects built by means of our three modeling constructs: composite objects, towers, and cities. These operators are defined in terms of manipulating elements of the values of these objects. These elements are the components of a composite object, the levels of a tower, and the views in a city, respectively. Let us denote by $C(o_1, o_2, \ldots)$ the object built from elements $o_1, o_2, \ldots$ by means of a construct $C$, the application of an operator $f$ to an object $o$ by $f : o$, and its application to $n$ objects $o_1, \ldots, o_n$ by $f :< o_1, \ldots, o_n >$.

The operators are:

**Apply-to-All:** Transform all values of elements of an object's value by applying a function to each of them. Such a function is a method defined for the type of the elements. Formally, the operator generated by apply-to-all with a function $f$ is denoted by $\prod_f$. Its application is defined as :
$\prod_f \ : \ C(o_1, o_2, \ldots) \ = \ C(f : o_1, f : o_2, \ldots)$.

**Filter:** Filter a subset of the values of the elements from an object according to a predicate. Such a predicate is a boolean method for the type of the elements. Formally, the operator generated by filter with a predicate $p$ is denoted by $\mathcal{F}_p$. Its application to an object is then defined as :
$\mathcal{F}_p \ : \ C(o_1, o_2, \ldots) \ =$
$C \ :< o_i \ | \ o_i \ \textit{satisfies predicate}\ p \ >$.

**Enumeration:** Construct an object using construct $C$ for an enumerated list of functions generating its elements. Formally, this is denoted by:
$\textit{Enumerate}_{C(f_1, f_2, \ldots)} : o \ = \ C(f_1 : o, f_2 : o, \ldots)$.

**Abstraction:** The abstraction (grouping) operator is applicable only to composite objects and partitions their elements according to a function. Each such partition contains those elements that yield the same value for the function, and is abstracted into a new composite object. Formally, for a function $f$, and a composite object constructor $C$, we define:
$\textit{Abstract}_{f,C} \ : \ C'(o_1, o_2, \ldots) \ =$
$C'(C : S_1, C : S_2, \ldots)'$,
where $S_1, S_2$ is a partition of $o_1, o_2, \ldots$ according to $f$ and $C'$ is a composite object constructor.

**Example 1:** Consider a computer-aided learning system in which courses are represented by hyperdocuments whose nodes contain (among other information) collections of exercises and whose links represent possible transitions. In this application one wants to select from each node those exercises with a particular degree of difficulty while maintaining the link structure. This is done by applying an apply-to-all to the hyperdocument with a function (which is a method for this kind of nodes) that generates from a given node a new node consisting of those exercises at the required level of difficulty.

The different views of the hyperdocument consist of its nodes relating to a particular subject. This is achieved by the application of abstraction: $\textit{Abstract}_{subject,city}$. Selecting a particular view in this city is done via a filter. □

**Example 2:** Consider a composite object representing a time-line. A filter operation is used to select the part of the time-line (and the events it contains) that lies within a specified interval of time. The interval is specified by a predicate $t_1 \leq$ *time* $\leq t_2$ that selects all events with a time (the location along the time-line) within the endpoints of the interval. Note that time is the location component of the time-line composite object. □

### 3.3 Browsing Semantics

Users interact with a hyperdocument by means of browsing through their elements. This behavioral aspect of a hyperdocument is an integral part of its functionality. Hence, we need a behavioral modeling construct that defines the **browsing semantics** of each kind of hyperdocument that we can define within the model[1]. To arrive at the desired general notion of a browsing semantics, observe that the browsing semantics of the familiar network hyperdocument is the set of allowed paths over the network. Hence, the browsing semantics of a particular type of hyperdocument is defined by the set of allowable paths within the hyperdocument. If one regards the familiar network hyperdocument as a finite automaton, the set of allowed paths is the language accepted by the automaton. Thus, a specification of the browsing semantics is a specification of a language of paths. Generally this path language would specify **trajectories** through composite objects. In some cases, the possible paths can be given extensionally by means of enumerating actual paths. However, in general one would specify the possible paths by giving (intensionally) mappings from time to elements of the composite node (or to coordinates in space).

Accordingly, we generally specify trajectories by means of **space-time axioms**. Formally,

**Definition 3.2** *A trajectory (path) is a mapping:* $Time \rightarrow Tower\ type$*, where the tower type is that of the elements in the composite object. The browsing semantics of a composite object is a set of trajectories over its elements. This set is specified through space-time axioms defining properties of the mappings used in trajectories.*

An example of an axiom is the above specification of paths over set composites. Trajectories yield tower objects as we have to specify not only the content of the path but also the visual appearance of the parts of the composite object that the user moves through. Different strategies for navigation in hyperdocument networks, depending on their topology [29], can be regarded as intensional ways of space-time axioms for particular kinds of composite nodes, e. g. multi-dimensional movements along coordinate axis for a multi-dimensional hypercube.

[1] We consider just the reading semantics.

## 4 Related Work

Several recent models for hyperdocuments have considered a notion of composite nodes which are themselves networks of nodes and links [4, 7]. Consequently, the kinds of complex information structures that these models are able to support is very limited compared to our general notion of composite object. The Aquanet data model, motivated by knowledge representation concerns [15] has suggested relations which are N-ary links. However, they have not provided for composite links having an internal structure as we do. This limits the users ability to design a hyperdocument at a high-level of abstraction in which complex relationships are regarded as first-class objects.

The Aquanet model [15] and the nested context model [4] for hyperdocuments have suggested that a description of basic node should include both its contents and a description of its visual presentation. Our tower construct carries this idea to its full generality, including its application to composite objects which does not appear in previous models.

The notion of a city binding the multiple views of an object or even an entire hyperdocument is novel, and captures a common design method for hyperdocuments in which their information is divided into multiple conceptual spaces [24].

Several hypermedia toolkits [20, 22] provide a library of classes to be used for the construction of hyperdocuments. However, these classes provide primitive objects without modeling constructs that would describe the way hyperdocuments are designed from these.

The HyTime [17] standard for hyperdocuments is a general model for describing the graphical presentation of time-based hyperdocuments. However, it deals only with their presentation level and just considers basic nodes of a variety of media. Cruz [6] has recently suggested an object-oriented data model for specifying visual queries to object-oriented databases. The type system used to formalize her model could be used for describing the presentation level of our tower objects, when implemented in a database programming language.

In contrast with models aiming at standardization ([10],[13]) our model aims at being a basis for implementation. Our model is on a much higher level than these other models. Its main goal is to be able to use hypermedia as a tool for the integration of different information sources in a heterogenous environment. An accompanying object oriented design and analysis methodology would help in using our model for that goal. So, instead of as a model suitable for particular application domains the model shoud be used as a kind of application generator.

## 5 Conclusions

We have presented a set of modeling constructs that make it possible to integrate a wide variety of information sources into a hypermedia system. These structures were previously opaque to the hypermedia system, limiting the usage of the common hypermedia functionality for manipulating and browsing them. Such commonality has significant benefits for the user and the designers of these information sources. Accordingly, we believe our model could be regarded as an interface layer that, once supported by different information sources, would permit their integration within the hypermedia system.

We have designed our data model so that its modeling constructs can be implemented with advanced database programming languages [18, 12]. We briefly indicate how such an implementation can be done as a layer on top of such a language. The first layer of our language, that of the basic objects, including basic presentation objects, is implemented by defining a library of appropriate classes [20, 22]. A modeling construct is a type constructor building a new type from the types of its constituents. A composite object constructor such as a set, or timetable, is implemented using the complex structure constructors provided in these languages. Corresponding to the different levels of composite towers, we would have composite type constructors building data structures, composite presentations [6], and composite time schedules from their elements. Thus, a city constructor constructs a function type defining the type of mappings from the parameter type into the view type. To impose the required common functionality on the objects constructed by a particular modeling construct, we define appropriate classes of composite objects, tower objects, and city objects, whose attributes and methods define this common functionality. In a similar way, we define the common functionality of the three kinds of first-class objects: nodes, links, and anchors. Virtual structures are implementable using active values as supported by advanced applications database systems. Such an implementation of our modeling constructs leads to a programming environment supporting the integration of information sources through the hypermedia system. It also allows the construction of such extensible hypermedia systems using available object-oriented database systems.

While many approaches have been suggested for object-oriented analysis and design of software systems. none has considered hyperdocuments and their special features. Existing data models for hyperdocuments have considered the facilities to be provided by a hypermedia system to support hyperdocuments, but have not suggested a systematic analysis methodology for extracting requirements for large hyperdocuments and for systematically developing their design. Our data model is intended to serve as the foundation for such an object-oriented analysis and design methodology for hyperdocuments.

An important direction for future research is the development of our data model into a model for describing virtual worlds [5]. These systems allow their users to browse through computer-generated worlds consisting of objects embedded in a virtual space. Much of their functionality is similar to the facilities that hyperdocuments in our model provide, and developing our model into a virtual reality data model would clarify and serve as a design method for such virtual worlds.

## References


[1] F. Bancilhon et al. The design and implementation of $O_2$, an object-oriented database system. In K.R. Dittrich, editor, *Proc. of the Intl. Workshop on Object-Oriented Database Systems*, pages 1–22, 1988.

[2] J. Backus. Can programming be liberated from the von Neumann style? A functional style and its algebra of programs. *Communications of the ACM*, 21(8):613–641, August 1978.

[3] P.D. Bruza. Hyperindices: a novel aid for searching in hypermedia. In A. Rizk et al., editors, *Hypertext: Concepts, Systems and Applications, Proc. European Conf. on Hypertext*, pages 109–122, 1990.

[4] M.A. Casanova et al. The nested context model for hyperdocuments. In *Proc. ACM Hypertext Conf.*, pages 193–201, 1991.

[5] C. Cruz-Neira et al. The CAVE: audio visual experience automatic virtual environment. *Communications of the ACM*, 64–73, June 1992.

[6] I.F. Cruz. Doodle: a visual language for object-oriented databases. In *ACM SIGMOD Intl. Conf. on Management of Data*, June 1992.

[7] F. Garzotto, P. Paolini, and D. Schwabe. HDM - a model for the design of hypertext applications. In *Proc. ACM Hypertext Conf.*, pages 313–329, 1991.

[8] F.G. Halasz. Reflections on notecards: seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–851, 1988.

[9] F. Halasz. Keynote speech: "seven issues" revisited. In *Proc. ACM Hypertext Conf.*, 1991.

[10] F. Halasz and M. Schwartz. The Dexter Reference Model. In *Proc. NIST Hypertext Standardization Workshop*, 1990.

[11] R. Hull and R. King. Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys*, 19:201–260, September 1987.

[12] M. Jarke et al. DAIDA: an environment for evolving information systems. *ACM Trans. on Information Systems*, 10(1):1–50, January 1992.

[13] D. Lange. A Formal Model of Hypertext. In *Proc. NIST Hypertext Standardization Workshop*, 1990.

[14] D. Maier et al. Development of an object-oriented DBMS. In *Proc. ACM Symp. on Object-Oriented Programming: Systems, Languages, and Applications*, pages 472–482, 1986.

[15] C.C. Marshall et al. Aquanet: a hypertext tool to hold your knowledge in place. In *Proc. ACM Hypertext Conf.*, pages 261–275, 1991.

[16] K.C. Malcolm, S.E. Poltrock, and D. Schuler. Industrial strength hypermedia: requirements for a large engineering enterprise. In *Proc. ACM Hypertext Conf.*, pages 13–24, 1991.

[17] S.R. Newcomb, N.A. Kipp, and V.T. Newcomb. The "HyTime" hypermedia/time-based document structuring language. *Communications of the ACM*, 67–83, November 1991.

[18] A. Ohori, P. Buneman, and V. Breazu-Tannen. Database programming in Machiavelli — a polymorphic language with static type inference. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 46–57, 1989.

[19] A. Pearl. Sun's link service: a protocol for open linking. In *Proc. ACM Hypertext Conf.*, pages 137–146, 1989.

[20] J. Puttress and N.M. Guimaraes. The toolkit approach to hypermedia. In *Hypertext: Concepts, Systems and Applications, Proc. European Conf. on Hypertext*, pages 25–37, 1990.

[21] J. Richardson and P. Schwarz. Aspects: extending objects to support multiple, independent roles. In *ACM SIGMOD Intl. Conf. on Management of Data*, pages 298–307, 1991.

[22] M. Sherman et al. Building hypertext on a hypermedia toolkit: an overview Andrew Toolkit hypermedia facilities. In *Hypertext: Concepts, Systems and Applications, Proc. European Conf. on Hypertext*, pages 13–24, 1990.

[23] P.D. Stotts and R. Furuta. Petri-net-based hypertext: document structure with browsing semantics. *ACM Trans. on Information Systems*, 7(1):3–29, 1989.

[24] N.A. Streitz, J. Hannemann, and M. Thuring. From ideas and arguments to hyperdocuments: travelling through activity spaces. In *Proc. ACM Hypertext Conf.*, pages 343–364, 1989.

[25] H. Schutt and N. Streitz. HyperBase: a hypermedia engine based on a relation database management system. In A. Rizk et al., editors, *Hypertext: Concepts, Systems and Applications, Proc. European Conf. on Hypertext*, pages 95–104, November 1990.

[26] K. Tanaka, N. Nishikawa, S. Hirayama, and K. Nanba. Query pairs as hypertext links. In *Proc. IEEE Data Engineering Conf.*, pages 456–463, 1991.

[27] E.R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990.

[28] K. Utting and N. Yankelovich. Context and orientation in hypermedia networks. *ACM Trans. on Information Systems*, 7(1):58–84, January 1989.

[29] H. Van Dyke Parunak. Hypermedia topologies and user navigation. In *Proc. ACM Hypertext Conf.*, pages 43–50, November 1989.

[30] H. Van Dyke Parunak. Don't link me in: set based hypermedia for taxonomic reasoning. In *Proc. ACM Hypertext Conf.*, pages 233–242, 1991.

[31] N. Yankelovich, B. Haan, N. Meyrowitz, and S. Drucker. Intermedia: the concept and the construction of a seamless information environment. *IEEE Computer*, 21(1):81–96, 1988.