UNIVERSITÄT DORTMUND
Fachbereich Informatik
Lehrstuhl für Software-Technologie

M E M O Nr. 126

Towards the algebraic analysis of hyperlink structures

Alexander Fronk

August 2002

Internes Memorandum des
Lehrstuhls für Software-Technologie
Prof. Dr. Ernst-Erich Doberkat
Fachbereich Informatik
Universität Dortmund
Baroper Straße 301

D-44227 Dortmund

ISSN 0933-7725

Towards the algebraic analysis of hyperlink structures

Alexander Fronk
Software-Technology, University of Dortmund
44221 Dortmund, Germany
`fronk@LS10.de`

August 16, 2002

Abstract

Structuring media objects such as text, graphics etc. by means of XML is a broadly discussed issue in hypermedia modeling. Thereby, an entire hypermedia document is not only arranged in such a way different developers may interchange data and have easy access to the inner structure of media objects by means of powerful tools available in the XML scene. Moreover, utilizing a given document structure to find new possibilities of linking documents is a major concern. Formal approaches, however, rarely appear in this context. In this paper, we contribute to formally structuring media objects and their linkage, thereby aiming at analyzing hyperlink structures. That is, properties of hyperlinks between media objects underlie a mathematical verification in advance of encoding the concrete hyperdocument. Algebraic specifications serve as a formal model allowing to obtain algebras reflecting hyperlink structures open to analysis.

1 Introduction

In [13], an object-oriented document description language for implementing hyperdocuments, *DoDL* for short [7, 8], was given an algebraic semantics. A hypermedia document, or hyperdocument for short, is understood as a collection of media objects such as texts, graphics, videos, etc. which are connected to each other in a non-linear fashion, that is, they are hyperlinked.

Media objects were given a tree-like structure which allows us to determine positions serving as link anchors. This approach is vital for the description of hyperdocuments by using the object-oriented language *DoDL*. The approach contrasts very much to the common HTML/XML implementation of hyperdocuments. Most remarkably, both *DoDL* and hyperdocuments were subject to formal investigations concerning the language's denotational semantics given by algebras, as well as the hyperdocuments structure. Hence, properties of structured media objects and hyperlinks can be checked and verified on a formal, algebraic level.

Due to their object-oriented description, hyperdocuments specified in *DoDL* can easily be comprehended as programs, or more generally, as software. Hyperdocuments hence inherently possess both document and program qualities. This document/program-dualism (c.f. [12]) can clearly be observed by the fact that hyperdocuments not only offer non-linearly related media and hence hyperlinked information. Moreover, navigational aspects, i.e. the hyperlinks of the document, are entirely encoded within the hyperdocument itself. If we understand any browser as a runtime environment allowing to navigate, a hyperdocument encompasses control structure. Hyperdocuments are often treated merely as documents leaving their characteristics as programs nearly completely aside.

Depending on the specific task a hyperdocument has to satisfy, its media objects may underlie frequent change. For example, hypermedia information systems showing environmental conditions such as air temperature, water-levels, or radio activity, etc. need to be updated or supplemented periodically with the latest measuring results. Thereby, the link structure has to be adapted to the new data. In the view of programs, the control structure has to be reimplemented. If, however, the link structure depends on positions within these media objects that can be described without referring to specific content (for example, if a certain occurrence of a city name, regardless of where exactly this name occurs within the media object, is always linked to a map showing some city details), hyperlinks can be captured abstractly without referring to concrete media objects. In the view of programs again, the control structure can be specified. Hence, an adequate abstract description of the documents' link structure can be reused, if it is given without reference to concrete data. Analogously, programs are specified without referring to their concrete input data. Exactly here, the specification of link structures finds its place.

In HTML/XML, hyperlinks and media objects are assembled within one single document. Using *tags* directly embedded within media objects makes it impossible to separate the description of hyperlinks from them. Our approach allows in contrast to specify the link structure abstractly and separately from media objects. Thereby, we strictly follow the principle of *separation of concerns*, and contribute to maintaining hyperdocuments [10] immediately on the level of maintaining software. A suitable compiler system allows to generate a specified hyperdocument by binding concrete media objects to the abstract description of a link structure. Thus, our approach is open to changing media objects frequently without neither touching the hyperdocument itself nor its specification. Some advantages become obvious here: By exchanging the values given in a binding and hence obeying the principle of *locality*, we are able to generated arbitrary many hyperdocuments underlying a certain link structure; further, the code is given in an object-oriented way and is thus open to inheritance, polymorphism, overloading, etc., and thus to high-level implementation of hyperdocuments. More details can be found in [23] and [14].

The notion of position is vital to obtain a proper specification of a hyperdocument. Therefore, we structure media objects in a tree-like manner. By introducing positions only based on this structure, we do not need to use coordinate systems laid upon media objects, and establish a uniform mechanism fitting to any kind of media object. Positions, and thereupon hyperlinks as pairs of positions, can then be elaborated formally. Algebraic specifications serve as formalism. Their loose semantics, i.e. a class of algebras some carrier-sets of which are not fixed, allows to analyze a specified link structure: dangling links, circles, undesired connections, etc. can be detected computationally. This approach develops its full power when the number of hyperlinks exceeds a level that makes it cumbersome to analyze hyperlinks by walking through the entire hyperdocument manually. Hence, formally testing link structures saves time, is less error-prone, helps to ensure certain properties, and, last not least, allows to mathematically prove structural and hyperlink properties. This is important for example in such environments where operating a hyperdocument relies on reasoning about its security.

The construction of hyperdocuments is generally not limited to structuring the media objects involved and their link structure (c.f. [20], chap. 9). Moreover, an overall view on *hypermedia systems* has to be taken into account including usability and presentation of hyperdocuments (c.f. [20], p. 7). For the time being, we are not interested in modeling hypermedia systems. Our approach is focussed on the specification and, more over, on the analysis of hyperlink structures. Hence, the visual appearance of media objects, i.e. their layout, is not a concern here, although we consider it as a very important aspect of a hyperdocument: layout is undoubtedly responsible for usability and acceptance. In addition to layout, the presentation of a hyperdocument encompasses the (animated) arrangement of its media objects in time and space (c.f. [16], p. 264). For our approach, capturing elements and their relative positions to each other is more important than their visual representation. This allows to talk about the composition of documents, about the notion of subdocuments, and about positions of subdocuments in the document under consideration.

This paper, which is an extension of the author's Ph.D. thesis, reports on our formal approach to structuring media objects and analyzing their linkage. The pragmatic goal of this work is to test the link structure of a hyperdocument in advance of its physical incorporation, i.e. on a conceptual level. The paper is organized as follows: Section 2 discusses related work; the structuring mechanism under consideration is introduced in Section 3; Section 4 discusses the formal analysis of link structures; Section 5 gives a conclusion and suggests further work.

2 Related work

A recursive and thus hierarchical document structure is proposed by Gamma et al. (c.f. [15], Sec. 2.2). They understand documents as a collection of textual and graphical elements, called *glyphs*. These elements are arranged in columns and rows, and are themselves refined recursively. The COMPOSITE design pattern (c.f. [15], p. 163) is based on this concept. Hence, the "formalization" of document structure is done by fixing a structuring mechanism.

A virtual document hierarchy is proposed by Gloor (c.f. [16], Sec. 1.5). An algorithm is responsible for computing so called *clusters* which assemble documents recursively by regarding their content. The clustered documents need not to form a physical unity, hence they are called *virtual*. This mechanism is dedicated to the construction of hierarchical overviews on document collections.

The language HyTime [22] offers tags to structure media objects in a modular way. As any language based on SGML/XML, those tags are textually placed within the media objects themselves. Separating a link structure from this description is not possible.

Furuta and Stotts formally model document structure by Petri nets [29]. Thereby, they concentrate on browsing information. In [30], they use model checking to analyze browsing properties.

Design and implementation of hyperdocuments is proposed by HDM [24] and OOHDM [25, 26]. This model distinguishes between the design of content, link structure, and views, followed by their implementation. This four-phase process-model captures content and links abstractly by so called *entity* and *link types*. Entities are structured stepwise before they are connected by concrete links. A hypermedia application is modeled object-orientedly in each phase. Refinement steps between these phases are allowed and lead to a top-down incremental prototype-based process. This approach, however, does not use formal methods.

The Dexter Hypertext Reference Model [17] characterizes hypermedia systems by runtime behavior, link structure and composition of media objects. It proposes the description of link structure by anchors which themselves have a structure specified by the model. In Section 3, we will refer to the Dexter Model again to relate our approach more precisely to this model.

The Dexter Model has been formalized by Tochtermann [31] using VDM [19]. This work formally describes structuring media objects and hyperlinks, but analyzing a specified link structure is not a concern. A similar work was done by Ossenbruggen and Eliëns [32]. They use Object-Z [28] instead of VDM.

Mattick and Wirth [21] specify a hypertext reference model algebraically. They focus on the automated support of hyperdocument development processes by presenting a formal description of the steps taken by such a process. Operations are specified the application of which allows a stepwise development of both a hyperdocuments' structure and linkage. They do not aim at analyzing link structures.

The XML path language, XPath (c.f. [6], chap. 11), is proposed to be used for locating parts of an XML document. This concept is not provided by XML itself. To use XPath, an XML document is seen as a tree in which each part of the document is represented as a node. XPath is a string-based language providing the user with operations to locate nodes and to operate on them. XPath-expressions aim at being used by other XML concepts such as document transformation or conversion. Our approach can be seen as a formalization of the tree structure used in XPath.

The XML linking language, XLink (c.f. [6], chap. 14), offers ways to link documents which are not provided by HTML. The language again uses tags, and hence the link structure is not separated from a media object it refers to. Analyzing the link structure can be covered by appropriate XML tools, but it is not expected be done formally. These XML-related technologies use tree-like structuring mechanisms, and due to their wide-spread use they are, to this extend, superior to our approach. Nonetheless, they lack the advantage of separating the description of link structures from concrete media objects. Further, XML-code does not underlie object-oriented features. This is a drawback when structuring and reusing code in a high-level manner becomes mandatory for large hyperdocuments.

3 Structuring media objects

This section introduces link structure formally. We discuss structured media objects offering a formal notion of position and, hence, a formal notion of link structure. Hyperdocuments are then characterized by structured media objects, positions and links. This approach to structuring and linkage is related to the Dexter model.

3.1 Basic notions

In this section, we introduce the notions vital for our approach.

A **media object** is characterized as any arbitrary, digitally storable information unit. In the hypertext context, media objects are referred to as *nodes* (c.f. [27], chap. 3). In XPath, nodes are parts of an XML documents. In our approach, a node represents a composite media object (see Sec. 3.2).

A **position** is a designated and unambiguously addressable location within a media object. In the Dexter model, positions are modeled by *anchors*, in hypertext they are also called *buttons* (c.f. [27], chap. 4). To characterize positions physically, we need to respect the media type, such as `doc`, `rtf`, `ASCII`, etc. for text, or `jpg`, `gif`, `png`, etc. for graphics, or `mpg`, `avi`, etc. for video. In our approach, media types are not necessary to specify positions.

A **hyperlink**, or link for short, is a pair of positions in not necessarily different media objects. The first component of such a pair is called **source** of the link, the second component is called **sink**.

A **hyperdocument** is a collection of media objects which are linked to each other by means of positions. Here, we only consider the link structure of a hyperdocument as one of its characterizing properties. Layout is not respected, since analyzing a link structure does not depend on layout.

A **link structure** is a collection of pairs of positions. Link structures can be viewed as directed, attributes graphs the nodes of which relate to positions, and the edges of which relate to links (c.f. [5]).

A **document** is a media object or a hyperdocument. Our notion of document thus subsumes information units as well as a linked collection of them.

3.2 Compositional media objects

Media objects are given a tree-like structure. We restrict ourselves to this structure for technical convenience only. Arbitrary graph structures can equally well be put to use though they get technically more complicated.

In our approach, *atomic media objects* form the smallest units by which we compose larger ones, called *compositional media objects*.

DEFINITION 1 **Compositional media objects** are recursively defined as follows:

1. Each atomic media object is a compositional media object.
2. Let $m_i, i = 1, \ldots, n$, with $n \geq 2$, be compositional media objects, which need not to be pairwise disjoint. Then, $m = \mathit{compose}(m_1, \ldots, m_n)$ is a compositional media object.

In the sequel, we refer to atomic media objects as *atoms*, and to compositional media objects as *compositionals*. The tree representation of a media object, m, is given as follows:

- An atom, a, is a tree consisting of a root node only; the root is labeled with a.
- A compositional, $m = \mathit{compose}(m_1, \ldots, m_n), n \geq 2$, is a tree the root node of which is labeled with m. The root has exactly n children labeled with m_i, such that m_i is the i-th son of the root.

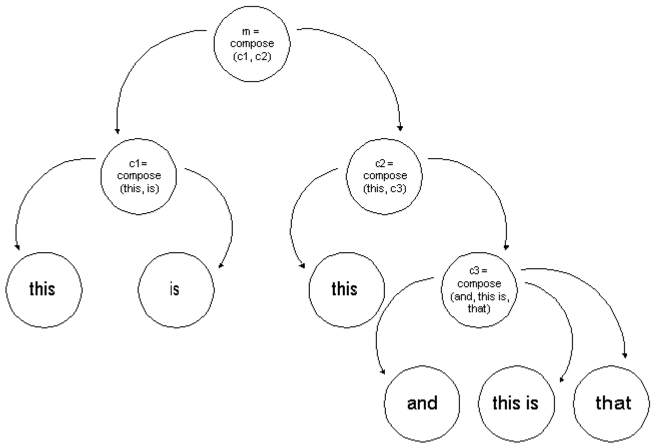
The set of constituents and the set of components of a media object, m, is defined by the immediate submedia objects of m, and by all the inner nodes of the tree representation of m, resp.:

DEFINITION 2 Let $m = compose(m_1, \ldots, m_n)$ be a media objects. Each $m_i, i = 1, \ldots, n$ is called a **part** of m. The set of **constituents** of m is the set of all its parts.
The set of **components** of m is recursively defined as follows:

$$components(m) := \begin{cases} \{m\}, & \text{if } m \text{ is atomic} \\ constituents(m) \cup \bigcup_{i=1}^{n} components(m_i), \\ \qquad \text{if } m = compose(m_1, \ldots, m_n), n \geq 2, \end{cases}$$

EXAMPLE 1 Figure 1 shows the tree representation of a compositional media object, $compose(c_1, c_2)$.

FIGURE 1 A compositional media object



The root represents the entire compositional, m, which is composed of two compositionals, c_1 and c_2. Those form the constituents of m. c_1 is composed of two atoms, this and is. We write c_1 as $compose(\text{this}, \text{is})$. Since this and is are smallest units, here words, they cannot be further decomposed and, hence, we cannot refer to their letters. The level a designer wants to fix atoms on is a matter of choice.

The compositional c_2, written as $compose(\text{this}, c_3)$, is composed of the atom this and another compositional c_3 the constituents of which are the atoms and, this is and that. Hence, we write c_3 as $compose(\text{and}, \text{this is}, \text{that})$. Since arbitrary information units can serve as media objects, we can also use any other media type here. For simplicity of presentation, we prefer text. Finally, the components of m are given by the set

$$\begin{aligned} components(m) := \{&\text{this}, \text{is}, \text{this is}, \text{that}, \\ &compose(\text{this}, \text{is}), compose(\text{and}, \text{this is}, \text{that}), \\ &compose(\text{this}, compose(\text{and}, \text{this is}, \text{that})), \\ &compose(compose(\text{this}, \text{is}), compose(\text{and}, \text{this is}, \text{that}))\}. \end{aligned}$$

■

The ordering of the atoms a compositional is composed of is important. Different orderings yield different compositionals. The same ordering, however, can be represented by trees with different structures. Referring to example 1, we can compose m by using its atoms directly, that is, m can be written as $compose(\text{this}, \text{is}, \text{this}, \text{and}, \text{this is}, \text{that})$. Hence, the tree structure of m is simply a sequence of leaves. This consideration leads to three different kinds of equality predicates definable on compositionals. They are based on the notion of *fringe* (we denote tuples by square brackets, and $\circ$ is their concatenation):

DEFINITION 3 Let m be a media object. The **fringe** of m is a tuple recursively defined as follows:

$$fringe(m) := \begin{cases} \langle m \rangle, & \text{if } m \text{ is atomic} \\ fringe(m_1) \circ \ldots \circ fringe(m_n), & \text{if } m = compose(m_1, \ldots, m_n), n \geq 2 \end{cases}$$

Two media objects, m and m', are called **fringe-equivalent**, $m \approx m'$ for short, if and only if they have equal fringes.
Two media objects, m and m' are called **structurally equivalent**, $m \cong m'$ for short, if and only if their respective set of components are equal.

Structural equivalence expresses the fact that two media objects are composed of structurally equivalent components. It is easy to see that fringe-equivalence and structural equivalence are independent from each other.

DEFINITION 4 Two media objects, m and m', are called **identical**, $m \equiv m'$ for short, if and only if $m \approx m'$ and $m \cong m'$ both hold.

The possibility to arrange media objects with equal fringes within different tree structures leads to the notion of *representative sets*, $\mathcal{C}_m$ for short. Each representative set collects all those media objects that are fringe-equivalent to a given media object, m, and the elements of $\mathcal{C}_m$ have pairwise disjoint tree structures. These sets allow for locating submedia objects without regarding their respective tree structure. They will be used in the definition of weak paths in section 3.3. Representative sets are hence the basic concept to search components in media objects, i.e. to find all occurrences of any given subcompositional.

It turns out that $\mathcal{C}_m$ is a singleton if m is atomic. Furthermore, $\mathcal{C}_m$ corresponds to the equivalence class $[m]_\approx$ of m under $\approx$, since $\approx$ is an equivalence relation.

The existence of this set for each media object can be shown by construction. In the sequel, let $\mathcal{MO}$ be the **universe of all media objects**.

DEFINITION 5 For each media object, m, the **representative set** of m is the smallest set $\mathcal{C}_m \subsetneq \mathcal{MO}$ with:

1. m is in $\mathcal{C}_m$
2. each media object m' with both
   (a) $m' \approx m$ and
   (b) $m' \ncong m$

   is in $\mathcal{C}_m$.

Construction of $\mathcal{C}_m$: Let $\{m_1, \ldots, m_n\}$, $n \geq 2$, be a finite set of arbitrary media objects, and let m be a compositional media object composed of m_i, $i = 1, \ldots, n$. Let the fringe of m be $fringe(m) = \langle m_1, \ldots, m_n \rangle$. Further, let $\mathcal{S}$ be a fringe, $\#\mathcal{S}$ its length, and $\pi_{\#\mathcal{S},i}$ its i-th component with $1 \leq i \leq \#\mathcal{S}$. To construct all media objects $comp \in \mathcal{C}_m$ with fringes equal to m but with different tree structures, do the following:

```
1. let $m := compose(m_1, \ldots, m_n)$, and let $\mathcal{C}_m := \emptyset$
2. **repeat** to compose a new media object $comp$:
   (a) let $\mathcal{S} := fringe(m)$
   (b) **repeat**
```

```
            i. with $k$ and $l$ where $1 \leq k \leq l \leq \#\mathcal{S}$, choose a part of $\mathcal{S}$ the elements of which
               are to be composed,
           ii. let $\mathcal{S}' := \langle \pi_{\#\mathcal{S},1}(\mathcal{S}), \ldots, \pi_{\#\mathcal{S},k-1}(\mathcal{S}) \rangle \circ \langle comp \rangle \circ \langle \pi_{\#\mathcal{S},l+1}(\mathcal{S}), \ldots, \pi_{\#\mathcal{S},n}(\mathcal{S}) \rangle$
               where $comp := compose^{(l-k)+1}(\pi_{\#\mathcal{S},k}(\mathcal{S}), \ldots, \pi_{\#\mathcal{S},l}(\mathcal{S}))$,
          iii. let $\mathcal{S} := \mathcal{S}'$
           until $fringe(comp) = fringe(m)$
       (c) if there does not exist a $m'$ in $\mathcal{C}_m$ with $comp \cong m'$, let $\mathcal{C}_m := \mathcal{C}_m \cup \{comp\}$
    until no media object can be formed over $\{m_1, \ldots, m_n\}$ the fringe of which is equal to
    $fringe(m)$, and which is not structurally equal to any media object already contained
    in $\mathcal{C}_m$.
```

This construction needs some remarks:

1. The process constructs fresh media objects $comp$ bottom up. Arbitrary elements $\pi_{\#\mathcal{S},i}(\mathcal{S}), i = k, \ldots, l$, of the fringe of m, $\mathcal{S}$, are composed. The fringe is cut down thereby, and serves as a new basis for the further construction of $comp$. $\mathcal{S}$ contains exactly the atoms m is composed of. By reducing the fringe it is guaranteed that each atom is used exactly ones within the construction process. The ordering of the atoms within the fringe is not changed. Hence, the fringe of $comp$ equals the fringe of m as soon as $\mathcal{S}$ has length 1. Then, the inner loop terminates.

2. Each media object in $\{m_1, \ldots, m_n\}$ is either used as a submedia object in $comp$, or is a constituent of $comp$. Hence, finitely many different combinations can be formed, and the set of trees is finite. This terminates the outer loop.

3. It is guaranteed that a media object m' is constructed during this process the tree representation of which is identical to that of m. Then, both $m' \approx m$ and $m' \equiv m$ hold, and m is inserted into $\mathcal{C}_m$.

3.3 Positions and links

Determining positions in media objects generally requires a suitable measuring system, i.e. a not necessarily numerical system imposed on a media object. For example, counting words in a text is suitable to unambiguously determine a specific location within a text; a coordinate system may help to determine areas within a picture by means of intervals; for videos, or those media objects that are arranged both in time and space simultaneously, fuzzy notations like "shortly before" or "while" may even help to fix a location (or at least a certain area) within those media. Hence, different measuring systems may be used for different media types. Our approach, however, is conceptual and consequently more abstract. Due to the recursive tree structure given to any kind of media object, we do not need different measuring systems for different media types, and use exactly one method for any media type instead. The notion of position we use here is adopted from the standard enumeration of vertices in trees by means of numerical strings (c.f. [1], p. 36).

The structure of a media object, m, allows to unambiguously address each submedia object, m', of m by means of a numerical string. We call such a string a *path*. The empty path, ϵ for short, refers to the root of a tree representation, and thus addresses the entire media object under discourse. We denote the composition of strings by $\frown$.

Definition 6 Let $m = compose(m_1, \ldots, m_n), n \geq 2$, and m' be two media objects. A **path** in m to m', $path(m, m')$ for short, is recursively defined as follows:

$$path(m, m') := \begin{cases} \epsilon, & \text{if } m \equiv m' \\ i \frown path(part(m, i), m'), & \text{if } m' \in components(part(m, i)), i = 1, \ldots, n, \\ \bot, & \text{else} \end{cases}$$

To determine a path to m', the media object m' must be a component of m. Hence, a path to m' leads to such a component of m that is identical and thus both fringe-equivalent and structurally equivalent to m'. In case, only fringe-equivalence is needed, i.e. if the specific structure of both m and m' is not a concern, we can define *weak paths* by means of representative sets:

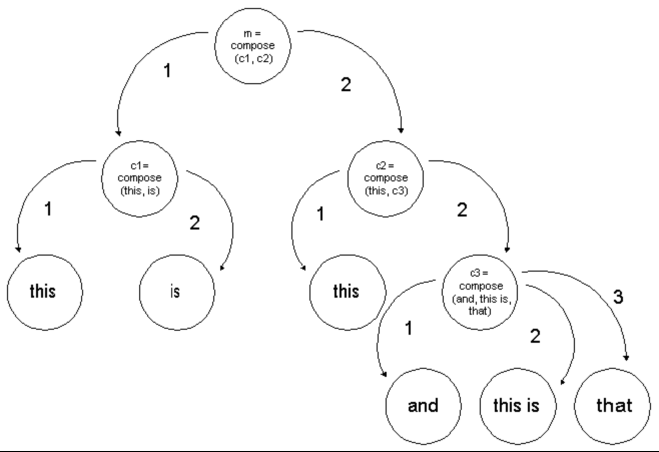
DEFINITION 7 Let $m = compose(m_1, \ldots, m_n), n \geq 2$, and m' be two media objects, and let $\mathcal{C}_{m'}$ be the representative set of m'. A **weak path** in m to m', $path_{\approx}(m, m')$ for short, is recursively defined as follows:

$$path_{\approx}(m,m') := \begin{cases} \epsilon, & \text{if } m \approx m' \\ i \frown path_{\approx}(part(m,i),m'), & \text{if there exists an } m'' \text{ in } \mathcal{MO} \text{ such that both} \\ & \quad 1.\ m'' \in components(part(m,i)) \text{ and} \\ & \quad 2.\ m'' \in \mathcal{C}_{m'} \text{ hold for } i = 1,\ldots,n \\ \bot, & \text{else} \end{cases}$$

Here, only the existence of a component in m is required which has a fringe equal to that of m'. Further, it is easy to see that each path is a weak path.

EXAMPLE 2 Figure 2 shows the compositional media object presented in example 1 on page 5. To determine paths, for each inner node we enumerate its outgoing edges from 1 to its outdegree. A path to the atom is is given by the string `12`, since the atom occurs as the second part of the

FIGURE 2 Paths in a media object



first part of m. Although the word is is found within the atom this is on path `222`, this submedia object is not a suitable choice because the atom this is cannot be further decomposed, and thus it has no parts that could be addressed. A path to this is, however, is given by `222`. Further, the compositional $c_1 = compose(\text{this}, \text{is})$ addressed by the path `1` is also a candidate for a weak path to this is, since the atom this is and c_1 have equal fringes.
■

At first glance, the example shows a disadvantage for our construction. In each media object, m, one can only address atoms or compositionals. It is not possible to address neighboring components such as this is this, if they are not explicitly composed to a media object. If the structure of m is neglected, however, the sequence this is this occurs in m. A path to this media object, $compose(\text{this}, \text{is}, \text{this})$, can be computed, however, if not only m is considered but also its representative set, $\mathcal{C}_m$. Then, each possible tree structure of m is available as search space. Now we can find several paths to the compositional $compose(\text{this}, \text{is}, \text{this})$ in different representations of m. Annotating paths with information about the representative of m makes addressing unique again. The construction we propose here is moreover advantageous: Remember that links are to be defined as pairs of positions. A designer can obviously fix a certain structure of m as an underlying search space aiming at explicitly hiding information within atoms not accessible for linkage.

We can consider paths as a concept to uniquely address any component of a (compositional) media object:

Definition 8 Let m be a media object. A **position** p_m in m is a weak path in m. The set of all positions in m is denoted by $\mathcal{P}_m$.

Notice that we use weak paths here, and concentrate on fringe-equivalence only. This allows for locating components independently of their tree structure. But we still consider the structure of the media object we are searching in.

We now define an interface to media objects which only respects positions. This allows to address specific locations within a media object without knowing its specific structure. For the time being, the interface only contains operations to address the beginning, the end, and any occurrence of any media object. Additional operations can be added if found useful in certain applications. Henceforth, let m and m' be two media objects.

Definition 9 The **beginning** of m, $\alpha(m)$ for short, is the path defined as follows:

$$\alpha(m) := \begin{cases} \epsilon, & \text{if } m \text{ is atomic} \\ 1 \frown \alpha(part(m,1)), & \text{else} \end{cases}$$

The **end** of m, $\omega(m)$ for short, is the path defined as follows:

$$\omega(m) := \begin{cases} \epsilon, & \text{if } m \text{ is atomic} \\ n \frown \omega(part(m,n)), & \text{if } m = compose(m_1, \ldots, m_n),\ n \geq 2 \end{cases}$$

These paths lead to the left-most and the right-most atom of m, respectively.

Since we allow a component to occur more than once in m, we define the set of all its occurrences:

Definition 10 The **set of occurrences** of m' in m, $\theta(m, m')$ for short, is defined as follows:

$$\theta(m, m') := \{p_m \in \mathcal{P}_m \mid p_m \text{ is a weak path in } m \text{ to } m'\}$$

Each position $p_m \in \theta(m, m')$ is called **occurrence** of m' in m.

Notice that $\theta(m, m')$ is empty if m' does not occur in m.

It is easy to see that the set of all positions, $\mathcal{P}_m$, given in def. 8 can be defined in analogy to [1], p. 36, as follows:

Definition 11 The **set of all positions**, $\mathcal{P}_m$, is defined as follows:

$$\mathcal{P}_m := \begin{cases} \{\epsilon\}, & \text{if } m \text{ is atomic} \\ \{\epsilon\} \cup \bigcup_{i=1}^{n} \{i \frown p \mid p_m \in \mathcal{P}_{m_i}\}, & \text{if } m = compose(m_1, \ldots, m_n),\ n \geq 2 \end{cases}$$

Further, we can prove that $\mathcal{P}_m$ equals the set $\bigcup_{m' \in (components(m) \cup \{m\})} \theta(m, m')$ which denotes the set of all weak paths in m.

Observation 1 The set $\mathcal{P}_m$ of all positions in m equals the set of all weak paths in m to each component of m and to m itself.

We prove the observation by induction on the tree structure of m.

Start of the induction: m is atomic. Then, m has the form $m = compose(m)$. By def. 2, the set of components of m is $components(m) = \{m\}$. By def. 7, ϵ is the only weak path in m to m itself. By def. 10, we have $\mathcal{P}_m = \theta(m, m) = \{\epsilon\}$.

Induction step: m has the form $compose(m_1, \ldots, m_n)$, $n \geq 2$. Let $\mathcal{K}_{m_i}$ be the sets of components of m_i, $i = 1, \ldots, n$. The set of components of m is

$$components(m) = \bigcup_{i=1}^{n} \mathcal{K}_{m_i} \cup \{m_1, \ldots, m_n\}$$

By the inductive assumption it holds for $i = 1, \ldots, n$ that

$$\mathcal{P}_{m_i} = \bigcup_{m' \in \ (\mathcal{K}_{m_i} \cup \{m_i\})} \theta(m_i, m').$$

It remains to show that

$$\mathcal{P}_m = \bigcup_{i=1}^{n} \mathcal{P}_{m_i} \cup \theta(m, m)$$

holds, since m is the only media object not regarded yet.

By def. 1 and def. 7, we have

$$\theta(m, m) = \{path_{\approx}(m, m)\} = \{\epsilon\}$$

as the only path in m to m. Then, with

$$\begin{aligned}
\mathcal{P}_m &= \bigcup_{i=1}^{n} \mathcal{P}_{m_i} \cup \theta(m, m) \\
&= \bigcup_{i=1}^{n} \left(\bigcup_{m' \in \ (\mathcal{K}_{m_i} \cup \{m_i\})} \theta(m_i, m') \right) \cup \theta(m, m) \\
&= \bigcup_{i=1}^{n} \left(\bigcup_{m' \in (\mathcal{K}_{m_i} \cup \{m_1, \ldots, m_n\})} \theta(m_i, m') \right) \cup \theta(m, m') \\
&= \bigcup_{m' \in components(m)} \theta(m, m') \cup \theta(m, m) \\
&= \bigcup_{m' \in (components(m) \cup \{m\})} \theta(m, m')
\end{aligned}$$

the observation is proved.

Finally, we can define links as pairs of positions:

Definition 12 Let p_m and $p'_{m'}$ be two positions in two not necessarily different media objects, m and m'. A **link** is defined as a pair of positions, $l = (p_m, p'_{m'})$, where p_m is in $\mathcal{P}_m$, and $p'_{m'}$ is in $\mathcal{P}_{m'}$. The **source** of l, $source(l)$ for short, is p_m, the **sink** of l, $sink(l)$ for short, is $p'_{m'}$.

3.4 Hyperdocuments

Now we are in a position to characterize our hyperdocuments on a conceptual level.

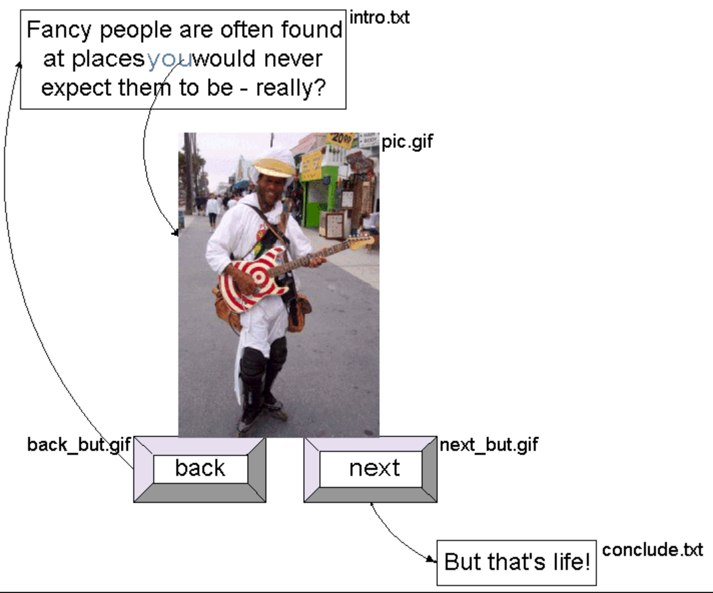
DEFINITION 13 A **hyperdocument** is a triple $\mathcal{H} = (\mathcal{M}, \mathcal{P}, \mathcal{L})$, where

- $\mathcal{M}$ is a non-empty set of media objects,
- $\mathcal{P}$ is any arbitrary set of positions in media objects in $\mathcal{M}$,
- $\mathcal{L}$ is any arbitrary set of links, $(p_m, p'_{m'})$, such that $p_m, p'_{m'} \in \mathcal{P}$.

We look at an example to explain how we model hyperdocuments by means of the formalization given so far. The algebraic definitions will be given in the next section.

EXAMPLE 3 Figure 3 shows a simple hyperdocument. It consists of five media objects: `intro.txt`, `pic.gif`, `back_but.gif`, `next_but.gif`, and `conclude.txt`. They are connected by three links. One link leads from the word `you` found in `intro.txt` to the picture, the other two links lead from the buttons back and next to `intro.txt` and `conclude.txt`, resp. For simplicity, we always assume texts to be composed of words. The picture, together with the two buttons, represents a compositional media object, *cntr*, and is written as $compose(\texttt{back\_but}, \texttt{pic}, \texttt{next\_but})$ (we omit the suffixes for convenience).

FIGURE 3 An explanatory hyperdocument



Fancy people are often found at places you would never expect them to be - really?
intro.txt
pic.gif
back_but.gif
back
next
next_but.gif
But that's life!
conclude.txt

The set of media objects, $\mathcal{M}$, can thus be given by

$$\begin{aligned}\mathcal{M} := \{&\texttt{intro}, \texttt{"you"},\\ &\texttt{pic}, \texttt{back\_but}, \texttt{next\_but},\\ &\texttt{conclude},\\ &compose(\texttt{back\_but}, \texttt{pic}, \texttt{next\_but})\}.\end{aligned}$$

Note that the word `"you"` is an atomic media object, and the compositional media object *cntr* is a member of $\mathcal{M}$ as well.

Hence, the beginning of *cntr* is `back_but`, the end is `next_but`. Since we assume texts to be composed of words, the representative tree of a text with n words consists of a root representing the entire text and having n sons representing each word, the atoms, as leaves. The i-th word of a text is hence the i-th son of the root. The position of the word "you", $\texttt{8}_{\texttt{you}}$, is thus given by its number within the text.

For simplicity again, we assume pictures to be atoms. The link starting in "you" leads to the picture. The sink of this link is given by the beginning of `pic` (the end of `pic` works equally well), or the second component of $cntr$. The two buttons are sources of the links leading to the beginning of the respective texts. They form the beginning (the end, resp.) of the compositional $cntr$. Hence, the set of positions is given by

$$\begin{aligned}\mathcal{P} := \{&\theta(\texttt{intro}, \texttt{you}), \alpha(\texttt{pic})\\ &\alpha(cntr), \omega(cntr),\\ &\alpha(\texttt{intro}), \alpha(\texttt{conclude})\}\end{aligned}$$

Written as paths, we have

$$\begin{aligned}\theta(\texttt{intro}, \texttt{you}) &= \{\texttt{8}_{\texttt{intro}}\},\\ \alpha(\texttt{pic}) &= \texttt{2}_{cntr} = \epsilon_{\texttt{pic}},\\ \alpha(cntr) &= \texttt{1}_{cntr} = \epsilon_{\texttt{back\_but}},\\ \omega(cntr) &= \texttt{3}_{cntr} = \epsilon_{\texttt{next\_but}},\\ \alpha(\texttt{intro}) &= \texttt{1}_{\texttt{intro}},\\ \alpha(\texttt{conclude}) &= \texttt{1}_{\texttt{conclude}}.\end{aligned}$$

The set of links is thus given by the following pairs of positions:

$$\mathcal{L} := \{(\texttt{8}_{\texttt{intro}}, \epsilon_{\texttt{pic}}), (\texttt{1}_{cntr}, \texttt{1}_{\texttt{intro}}), (\texttt{3}_{cntr}, \texttt{1}_{\texttt{conclude}})\}$$

Depending on the concrete text `intro`, the set of occurrences of the word `"you"` may differ: we may obtain more than one link, if each occurrence is specified to be connected to the picture. The set may even be empty, if `"you"` does not occur in `intro` at all.
■

3.5 Relation to the Dexter model

The Dexter model, as described in [17], aims at establishing a standard on hypermedia systems in general, that is, it proposes a model each hypermedia system should be an instance of.

The model consists of three layers, the run-time layer, the storage layer, and the within-component layer. The last layer is purposely left unspecified to allow for different structuring mechanisms as well as different media types. Our tree-like structures can be seen as a formalization of this layer. The run-time layer is not a concern in our approach, since presentation and layout are not crucial for analyzing link structures.

The storage layer, however, is of greater interest. The entity used here is a *component*. It is divided into an information component and a base component. The former possesses a unique component identifier, *UID* for short, and a set of *anchors*. The latter consists of atomic and compositional media objects, as well as links. Anchors serve as sources and sinks for links, and are pairs of identifiers and values. The value describes a certain position within a component.

A *specifier* consists of a component specification and an anchor identifier (for the time being, we do not consider direction and presentation). A link is a sequence of specifiers, such that arbitrary linkage is feasible.

In our approach, links are modeled as pairs on positions. The link set thus forms a relation. Hence, we allow for arbitrary link structures, too.

Positions are evaluated within media objects. The variable used to address a media object can be seen as a *UID* in the Dexter model, since positions are unambiguously assigned to media objects. Operations to determine positions, such as `getBegin` etc., serve as an anchoring mechanism, the interface between the storage and within-component layer. Our positions directly correspond to anchors, since an anchor value is given by a path within a media object, and the anchor identifier relates to the vertice addressed by the path, or, equally, its unique number within a tree representation.

Similar to the Dexter model, concrete positions depend on concrete media objects and their structure, and the measuring system is the media object itself. In [18], the notions of *time* and *context* are added to the Dexter model making it applicable to more complex media. Our approach is open to extensions by adding further operations and structuring mechanisms yielding to determining positions in more complex structures than trees.

4 An algebraic model for link structures

In this section, we consider the analysis of hyperlink structures. We use a loose semantics approach. First, we introduce some preliminaries and fix the notation used here. The reader familiar with algebraic specification may proceed to section 4.3, where the hyperdocument discussed in example 3 on page 11 is specified.

4.1 Algebraic preliminaries

A **signature** is a tuple $\langle S, \Gamma \rangle$ where S is a set of **sorts** and Γ is a set of **operation symbols**.

An **algebraic specification**, SP, is a tuple $\langle \Sigma, E \rangle$ consisting of a signature, $\Sigma = sig(SP)$, and a set of formulas, $E = axms(SP)$, called **axioms**, over Σ.

A Σ-**algebra**, $\mathcal{A}$, is a pair $(\{A_s\}_{s \in S}, \{f^{\mathcal{A}}\}_{f \in \Gamma})$ consisting of a family $\{A_s\}_{s \in S}$ of non-empty **carrier-sets**, A_s, for each $s \in S$, and a set $\{f^{\mathcal{A}}\}_{f \in \Gamma}$ of **operations** $f^{\mathcal{A}} : A_{s1} \times \ldots \times A_{sn} \to A_s$ for each $f : s_1 \times \ldots \times s_n \to s \in \Gamma$.

A Σ-algebra, $\mathcal{A}$, is a **model** of a specification, $\langle \Sigma, E \rangle$, if $\mathcal{A}$ satisfies each formula $e \in E$. The set of all models of $\langle \Sigma, E \rangle$ is denoted by $Alg(\Sigma, E)$.

The **loose semantics** of a specification $SP = \langle \Sigma, E \rangle$, $Mod(SP)$ for short, is defined as the set $Alg(\Sigma, E)$.

Let $\Sigma = \langle S, \Gamma \rangle$ and $\Sigma' = \langle S', \Gamma' \rangle$ be two signatures with $\Sigma \subseteq \Sigma'$, and let $\mathcal{A}'$ be a Σ'-algebra. The Σ-algebra $\mathcal{A}'|_{\Sigma}$ is called Σ**-reduct** of $\mathcal{A}'$, if for each $s \in S$ the carrier-set $(\mathcal{A}'|_{\Sigma})_s$ is defined as A'_s, and for each $f \in \Gamma$ the operation $f^{\mathcal{A}'|_{\Sigma}}$ is defined as $f^{\mathcal{A}'}$.

Let SP and SP' be two specifications, such that all pairwise equal operation symbols have the same characteristics. The specification-building operation **import into** : $SPEC \times SPEC \to SPEC$ is defined as follows:

$$\begin{aligned} sig(import\ SP\ into\ SP') &:= sig(SP) \cup sig(SP') \\ Mod(import\ SP\ into\ SP') &:= \{\mathcal{A} \in Alg(\Sigma_{import\ SP\ into\ SP'}) \mid \mathcal{A}|_{\Sigma_{SP}} \in Mod(SP)\} \end{aligned}$$

We write **import** $SP_1, \ldots, SP_n$ **into** SP as an abbreviation for

$$\textbf{import } SP_1 \textbf{ into } (\ldots(\textbf{import } SP_n \textbf{ into } SP)\ldots)$$

4.2 Compositional media objects and hyperdocuments

The composition of media objects as defined in section 3.2 can be specified algebraically as shown in specification 1 on page 14.

The sort $MedObj$ represents the universe of media objects, $\mathcal{MO}$. The operation symbols follow the names given in section 3.2, the axioms follow the respective definitions. We write $m' \,\partial\, m$ to abbreviate the fact that a media object, m', is a component of a compositional m. The specification uses a specification for sets, which is given in specification 3 on page 16, and a specification for tuples, which is given in specification 2 on page 15.

We further assume a specification NAT for the data type `integer` with a sort nat, and a specification $BOOL$ for data type `bool` with a sort $bool$ together with the usual constants $true$ and $false$ to be given in each specification. Similarly, we assume a ternary operation $if\ \_\ then\ \_\ else\ \_$:

SPECIFICATION 1 Compositional media objects

$COMPOSITIONAL =$
sorts $\quad MedObj$
opns

$$
\begin{aligned}
compose^n &: MedObj^n \to MedObj, && n \geq 2\\
part_n &: MedObj \times nat \to MedObj, && n \geq 2\\
constituents &: MedObj \to set(MedObj),\\
components &: MedObj \to set(MedObj),\\
fringe &: MedObj \to \langle MedObj \rangle^n, && n \in \mathbb{N}^+\\
\_\approx_{MedObj}\_ &: MedObj \times MedObj \to bool,\\
\_\partial_{MedObj}\_ &: MedObj \times MedObj \to bool
\end{aligned}
$$

vars $\quad d, d_1, \ldots, d_n : MedObj; i, n : nat$
axms

$$
\begin{aligned}
part_n(compose^n(d_1, \ldots, d_n), i) &= d_i, \qquad i = 1, \ldots, n && (1)\\
compose^n(part_n(d, 1), \ldots, part_n(d, n)) &= d, && (2)\\
constituents(d) &= \{d\} && (3)\\
constituents(compose^n(d_1, \ldots, d_n)) &= \{d_1\} \cup \ldots \cup \{d_n\}, && (4)\\
components(d) &= \{d\}, && (5)\\
components(compose^n(d_1, \ldots, d_n)) &= constituents(compose^n(d_1, \ldots, d_n))\\
&\quad \cup components(d_1) && (6)\\
&\quad \cup \ldots\\
&\quad \cup components(d_n)\\
fringe(d) &= \langle d \rangle, && (7)\\
fringe(compose^n(d_1, \ldots, d_n)) &= fringe(d_1) \circ \ldots \circ fringe(d_n), && (8)\\
(d_1 \approx_{MedObj} d_2) &= (fringe(d_1) = fringe(d_2)), && (9)\\
(d\, \partial_{MedObj}\, compose^n(d_1, \ldots, d_n)) &= (d \in components(compose^n(d_1, \ldots, d_n))) && (10)
\end{aligned}
$$

$bool \times s \times s \to s$ to be given for each sort $s \in S$, which has the usual semantics if $true$ then x else $y =_s x$ and if $false$ then x else $y =_s y$. A standard definition for $BOOL$ and NAT is for example given in [33], on page 699 and 700, resp. The μ-operator used is defined as usual for a boolean function $f : nat \to bool$, such that $\mu i.f(i)$ denotes the smallest i in $\mathbb{N}$ for which $f(i)$ holds.

SPECIFICATION 2 A specification for tuples

$TUPLE =$
sorts $s_1, \ldots, s_n, \langle s_1, \ldots, s_n \rangle$
opns

$$\begin{aligned} tup^n &: s_1 \times \ldots \times s_n \to \langle s_1, \ldots, s_n \rangle, & n \in \mathbb{N}^+ \\ \pi_{n,i} &: \langle s_1, \ldots, s_n \rangle \to s_i, & i = 1, \ldots, n \\ \_ =_{\langle s_1, \ldots, s_n \rangle} \_ &: \langle s_1, \ldots, s_n \rangle \times \langle s_1, \ldots, s_n \rangle \to bool & \\ \_ \circ_{n,m} \_ &: \langle s_1, \ldots, s_n \rangle \times \langle s_1, \ldots, s_m \rangle \to \langle s_1, \ldots, s_{n+m} \rangle, & n, m \in \mathbb{N}^+ \end{aligned}$$

vars $x_1 : s_1, \ldots, x_n : s_n, y_1 : s_1, \ldots, y_n : s_n, t_1, t_2 : \langle s_1, \ldots, s_n \rangle$
axms

$$\pi_{n,i}(tup^n(x_1, \ldots, x_n)) =_{s_i} x_i, \qquad i = 1, \ldots, n \tag{11}$$

$$(t_1 =_{\langle s_1, \ldots, s_n \rangle} t_2) =_{bool} \bigwedge_{i=1}^{n} (\pi_{n,i}(t_1) =_s \pi_{n,i}(t_2)), \tag{12}$$

$$t =_{\langle s_1, \ldots, s_n \rangle} tup^n(\pi_{n,1}(t), \ldots, \pi_{n,n}(t)), \tag{13}$$

$$tup^n(x_1, \ldots, x_n) \circ tup^m(y_1, \ldots, y_m) =_{\langle s_1, \ldots, s_{n+m} \rangle} tup^{n+m}(x_1, \ldots, x_n, y_1, \ldots, y_m) \tag{14}$$

With these specifications, we can specify our hyperdocuments under consideration. Specification 4 on page 17 imports $COMPOSITIONAL$ and introduces new sorts for positions and links. The operation symbols again follow those used in sections 3.3 and 3.4. The constant ϵ represents the empty path, and $error_{position}$ reflects the case a path is undefined. The operation $ispath$ is used to check whether there exists a path to a component m' of a compositional media object m.

Specification 3 A specification for sets

$SET =$
sorts $s, set(s)$
opns

$$\begin{aligned} empty &:\to set(s), \\ insert &: set(s) \times s \to set(s), \\ delete &: set(s) \times s \to set(s), \\ \_\in_s\_ &: s \times set(s) \to bool, \\ \_\subset_s\_ &: set(s) \times set(s) \to bool, \\ \_=_{set(s)}\_ &: set(s) \times set(s) \to bool \end{aligned}$$

vars $a, b, c : set(s); x, y : s$
axms

$$insert(insert(a, y), x) =_{set(s)} insert(insert(a, x), y), \tag{15}$$
$$insert(delete(a, x), x)) =_{set(s)} insert(a, x), \tag{16}$$
$$delete(empty, x) =_{set(s)} empty, \tag{17}$$
$$delete(insert(empty, x), x) =_{set(s)} empty, \tag{18}$$
$$(delete(insert(a, x), x) =_{set(s)} a) =_{bool} \neg(x \in_s a), \tag{19}$$
$$(a =_{set(s)} b) =_{bool} (a \subset_s b) \wedge (b \subset_s a), \tag{20}$$
$$(empty \subset_s a) =_{bool} true, \tag{21}$$
$$(insert(a, x) \subset_s b) =_{bool} (a \subset_s b) \wedge (x \in_s b), \tag{22}$$
$$(a \subset_s insert(b, x)) =_{bool} (a \subset_s b), \tag{23}$$
$$((a \subset_s delete(b, x)) \to a \subset_s b) =_{bool} true, \tag{24}$$
$$(a \subset_s b \wedge b \subset_s c \to a \subset_s c) =_{bool} true, \tag{25}$$
$$(x \in_s insert(a, x)) =_{bool} true, \tag{26}$$
$$(x \in_s delete(a, x)) =_{bool} false \tag{27}$$

Specification 4 An algebraic specification for hyperdocuments

$HDOC =$ **import** $COMPOSITIONAL$ **into**
sorts $position, link$
opns

$$\begin{aligned}
path_{\approx} &: MedObj \times MedObj \rightarrow position,\\
ispath &: MedObj \times MedObj \times nat \rightarrow bool,\\
\epsilon &:\rightarrow position,\\
error_{position} &:\rightarrow position,\\
\alpha &: MedObj \rightarrow position,\\
\omega &: MedObj \rightarrow position,\\
\theta &: MedObj \times MedObj \rightarrow set(position),\\
\psi &: position \times position \rightarrow link,\\
\psi_{set(link)} &: set(position) \times position \rightarrow set(link),\\
source &: link \rightarrow position,\\
end &: link \rightarrow position,\\
=_{link} &: link \times link \rightarrow bool
\end{aligned}$$

vars $m, m_1, \ldots, m_n, m' : MedObj; p, q : position; pset : set(position); l, l' : link; i : nat$
axms

$$\begin{aligned}
ispath(compose^n(m_1,\ldots,m_n), m', i) = {} & \exists m'' : MedObj\ .\\
& \quad m'' \,\partial\, part_n(compose^n(m_1,\ldots,m_n), i) && (28)\\
& \quad \wedge m'' \approx m', \qquad i = 1,\ldots,n\\
path_{\approx}(compose^n(m_1,\ldots,m_n), m') = {} & \text{if } compose^n(m_1,\ldots,m_n) \approx m' \text{ then } \epsilon\\
& \text{else} && (29)\\
& \quad \text{if } \exists i : nat.ispath(compose^n(m_1,\ldots,m_n), m', i)\\
& \quad \text{then } (\mu i.ispath(compose^n(m_1,\ldots,m_n), m', i))\\
& \qquad \frown path_{\approx}(part_n(compose^n(m_1,\ldots,m_n), i), m')\\
& \quad \text{else } error_{position}, \qquad i = 1,\ldots,n\\
\alpha(m) = {} & \epsilon, && (30)\\
\alpha(compose^n(m_1,\ldots,m_n)) = {} & 1 \frown \alpha(part_n(compose^n(m_1,\ldots,m_n), 1)), && (31)\\
\omega(m) = {} & \epsilon, && (32)\\
\omega(compose^n(m_1,\ldots,m_n)) = {} & n \frown \omega(part_n(compose^n(m_1,\ldots,m_n), n)), && (33)\\
\theta(compose^n(m_1,\ldots,m_n), m') = {} & \text{if } compose^n(m_1,\ldots,m_n) \approx m' \text{ then } \{\epsilon\}\\
& \text{else} && (34)\\
& \quad \text{if } \exists i : nat.ispath(compose^n(m_1,\ldots,m_n), m', i)\\
& \quad \text{then } \{i \frown p \mid ispath(compose^n(m_1,\ldots,m_n), m', i),\\
& \qquad\qquad p \in \theta(part_n(compose^n(m_1,\ldots,m_n), i), m')\}\\
& \quad \text{else } \{\}, \qquad i = 1,\ldots,n\\
(l \in \psi_{set(link)}(pset, p)) = {} & (source(l) \in pset \wedge sink(l) = p) && (35)\\
source(\psi(p,q)) = {} & p, && (36)\\
sink(\psi(p,q)) = {} & q, && (37)\\
(l =_{link} l') = {} & ((source(l) = source(l')) \wedge (sink(l) = sink(l'))) && (38)
\end{aligned}$$

We consider example 3 again and show how specification $HDOC$ is used to formally describe the hyperdocument (see spec. 1) introduced in the example.

Example 4 First, we need a new sort, $example$, for technical reasons.

Specification 1 An explanatory specification

$EXAMPLE =$ **import** $HDOC$ **into**
sorts $example$
opns $intro : example \rightarrow MedObj$,
$conclude : example \rightarrow MedObj$,
$pic : example \rightarrow MedObj$,
$back : example \rightarrow MedObj$,
$next : example \rightarrow MedObj$,
$word : example \rightarrow MedObj$,
$cntr : example \rightarrow MedObj$,
$linkIntro : example \times MedObj \rightarrow set(link)$,
$linkBack : example \times MedObj \rightarrow link$,
$linkNext : example \times MedObj \rightarrow link$
vars $e : example$
axms $cntr(e) = compose(back(e), pic(e), next(e))$,
$linkIntro(e, intro(e)) = \psi_{set(link)}(\theta(intro(e), word(e)), \alpha(pic(e)))$,
$linkBack(e, intro(e)) = \psi(\alpha(cntr(e)), \alpha(intro(e)))$,
$linkNext(e, conclude(e)) = \psi(\omega(cntr(e)), \alpha(conclude(e)))$

The sorts $MedObj$, $position$ and $link$ are already introduced through $HDOC$. Each media object is represented by an unary operation. The operations $linkIntro$, $linkBack$, $linkNext$ are used to specify the desired hyperlinks. The axioms fix their semantics and describe the compositional media object $cntr$. $linkIntro$ specifies a set of links starting at each occurrence of a specific word in the text (represented by $intro$) and ending in the beginning of the picture (represented by pic). The operation ψ, defined in $HDOC$, pairs two positions to a link, whereas the operation $\psi_{set(link)}$ yields a set of links by pairing each element of a set of positions with exactly one other position. The former are used as different link sources all leading to the same sink. The operation $linkBack$ and $linkNext$ specify links from the beginning and the end of $cntr$ to the beginning of $intro$ and $conclude$, resp.

This specification can be implemented in *DoDL* as shown in listing 1 on page 19. We use a class, `Example`, to collect the media objects involved in the class attributes, called document variables. They are introduced by the keyword **documents** and are of type **MedObj**, representing media objects. Document variables are assigned concrete values within the binding-section. Here, we can simply exchange the given values to generate a different hyperdocument while still using the same specification of its link structure. This link structure is abstractly captured in class methods, introduced by the keyword **construct**. The method `composeCntr` is used to compose the graphical items. It returns the value `cntr`. The method `compose` is predefined in *DoDL*. The method `createLinks` is responsible for constructing the links. The methods `getBegin`, `getEnd`, `getOcc` and `setLink` are also predefined in *DoDL*. They correspond to α, ω, etc. as defined in section 3. This implementation can be translated by a compiler and executed on a machine. Doing so, the hyperdocument is generated using HTML.

The advantages mentioned in the introduction become obvious here: by exchanging the values given in the binding, we are able to generated arbitrary many hyperdocuments underlying a certain link structure; the rearrangement of the compositional is simply done by changing `composeCntr`.

■

In the example, the choice of the texts, `conclude` and `intro`, is worth a note. We can generate different hyperdocuments with different positions and different links simply by exchanging

LISTING 1 An explanatory *DoDL*-Listing

```
class Example is
  documents intro, conclude, pic, but1, but2: MedObj;
  construct
    MedObj composeCntr(void){
      MedObj cntr;
      cntr = compose(but1, pic, but2);
      return cntr;
    };

    void createLinks(void){
      getOcc(intro, "you").setLink(pic.getBegin());        // a link from "you" to the picture
      getBegin(composeCntr()).setLink(intro.getBegin()); // linking the back−button
      getEnd(composeCntr()).setLink(conclude.getBegin()); // linking the next−button
    };
end Example;

binding Example is
  intro = intro.txt;
  conclude = conclude.txt;
  pic = picture.gif;
  but1 = back_but.gif;
  but2 = next_but.gif;
end;
```

the texts. Hence, it is not clear from a specification as shown in the specification above which specific hyperdocument is obtained. Its link structure depends on the choice of media objects given in a certain algebra, i.e. an interpretation of the respective operations given in the specification, and is thus worth an analysis to certify desired properties in advance of the hyperdocument's incorporation.

4.3 Analyzing hyperlink structures

To analyze a given link structure, we have to look at algebras serving as interpretations of an algebraic specification and thus, in our case, as interpretations of a specified hyperdocument. Within such an algebra, carrier-sets have to be fixed and operations need to be interpreted. By fixing the carrier-set for $MedObj$, we regard a certain set of media objects on which a concrete link structure depends. Hence, we can formally "compute" positions and links by interpretation.

We use a loose semantics approach here, and are not interested in initial or terminal models. The carrier-sets for $MedObj$, $position$ and $link$ are of most interest, since they completely characterize a hyperdocument. All other sorts can be interpreted arbitrarily. We reconsider example 3 on page 11, but now more formally.

EXAMPLE 5 Let $\mathcal{E}$ be a $\Sigma_{EXAMPLE}$-algebra, and let the carrier-set for $MedObj$ be as follows:

$$\begin{aligned} E_{MedObj} = \{ & \texttt{intro}, \texttt{you}, \\ & \texttt{pic}, \texttt{back\_but}, \texttt{next\_but}, \\ & \texttt{conclude}, \\ & compose(\texttt{back\_but}, \texttt{pic}, \texttt{next\_but})\} \end{aligned}$$

Hence, certain documents are given by interpretation of the sort $MedObj$.

With this carrier-set at hand, we can interpret the operation representing media objects:

$$\begin{array}{lll} intro^{\mathcal{E}}(e) = \texttt{intro}, & word^{\mathcal{E}}(e) = \texttt{you}, & conclude^{\mathcal{E}}(e) = \texttt{conclude} \\ pic^{\mathcal{E}}(e) = \texttt{pic}, & back^{\mathcal{E}}(e) = \texttt{back\_but}, & next^{\mathcal{E}}(e) = \texttt{next\_but}, \\ cntr^{\mathcal{E}}(e) = compose^{\mathcal{E}}(back^{\mathcal{E}}(e), pic^{\mathcal{E}}(e), next^{\mathcal{E}}(e)) & & \end{array}$$

Similarly, we interpret the link constructing operations following the axioms given:

$$\begin{aligned} linkIntro^{\mathcal{E}}(e, intro^{\mathcal{E}}(e)) &= \psi^{\mathcal{E}}_{set(link)}(\theta^{\mathcal{E}}(intro^{\mathcal{E}}(e), word^{\mathcal{E}}(e)), \alpha^{\mathcal{E}}(pic^{\mathcal{E}}(e))), \\ linkBack^{\mathcal{E}}(e, intro^{\mathcal{E}}(e)) &= \psi^{\mathcal{E}}(\alpha^{\mathcal{E}}(cntr^{\mathcal{E}}(e)), \alpha^{\mathcal{E}}(intro^{\mathcal{E}}(e))), \\ linkNext^{\mathcal{E}}(e, conclude^{\mathcal{E}}(e)) &= \psi^{\mathcal{E}}(\omega^{\mathcal{E}}(cntr^{\mathcal{E}}(e)), \alpha^{\mathcal{E}}(conclude^{\mathcal{E}}(e))) \end{aligned}$$

The other operations introduced by $HDOC$ are interpreted in $\mathcal{E}$ following the axioms given in $HDOC$, or their interpretation is term-generated. That is, for example, $compose$ and α are interpreted as follows (a specification for tuples is given in spec. 2 on page 15):

$$compose^{\mathcal{E}}(m_1, \dots, m_n) = compose(m_1, \dots, m_n)$$
$$\text{for each } m_i \in E_{MedObj}, i = 2, \dots, n$$
$$\alpha^{\mathcal{E}}(m) := \begin{cases} \epsilon^{\mathcal{E}}, & \text{if } m \text{ is atomic} \\ \langle 1 \rangle \circ \alpha^{\mathcal{E}}(part^{\mathcal{E}}_n(m, 1)), & \text{else} \end{cases}$$

Hence, it is easy to prove that $\mathcal{E}$ is a model for specification $EXAMPLE$. We can construct specification $HDOC$ in such a way that its model class corresponds to the class of free term-algebras (c.f. [9], chap. 10.5) over the set of atomic media objects. We avoid this construction, however, for technical simplicity.

Following the interpretations, we can restrict the carrier-sets for $position$ and $link$ to exactly those values needed within the interpretations. That is, we evaluate the interpretations and yield the following carrier-sets:

$$\begin{aligned} E_{position} &= \{\texttt{8}_{\texttt{intro}}, \texttt{1}_{\texttt{intro}}, \texttt{1}_{\texttt{conclude}}, \epsilon_{\texttt{pic}}, \texttt{1}_{cntr}, \texttt{3}_{cntr}\} \\ E_{link} &= \{(\texttt{8}_{\texttt{intro}}, \epsilon_{\texttt{pic}}), (\texttt{1}_{cntr}, \texttt{1}_{\texttt{intro}}), (\texttt{3}_{cntr}, \texttt{1}_{\texttt{conclude}})\} \end{aligned}$$

These values correspond to those given in example 3.
■

Since we know that our definition of hyperdocuments given in def. 13 on page 11 and, by that, our concepts of structured media objects, positions and links can be properly formalized, we could omit the algebraic part discussed so far. But doing this returns us into the position of not specifying a hyperdocuments link structure. This specification, however, is crucial to compute a concrete hyperlink structure and can be reused by simply changing the interpretation of media objects. For large documents, the possibility to *compute* positions and links is advantageous. For small documents, of course, one can use the definitions given in section 3.

In the sequel of this section, we discuss some properties of hyperlink structures and show how they can be checked on carrier-sets. Note, that a concrete link structure depends on the media objects chosen. Hence, properties may hold or may not hold due to this choice.

The Dexter model allows for links with multiple targets, i.e. sinks. In HTML, however, this is not realizable. For this reason, it might be convenient to forbid such a property. If we have to check that each link has a unique sink, we have to search in E_{link} for element-pairs of the form (p, p') and (p, p''), where p' and p'' are different positions. This can easily be done by pairwise comparing elements in E_{link}, or by using hashing if the structure of E_{link} is not supposed to be a flat list of pairs.

Inline-links, i.e. links within one single media object, may not be allowed in some applications. Hence we have to check, if their exist pairs of the form (p_m, p'_m).

Reachability is often a concern. We can check this property by following sequences of the form (p_m, p_{1m_1}), $(p_{2m_1}, \ldots)$, ..., $(\ldots, p'_{m'})$ to see that a media object m' is reachable from m.

Those and further properties can be formalized and tested mechanically depending on the structure of E_{link}. We are currently checking if relational methods (c.f. [4]) can do a good service here, since the link set is a relation on positions. Formalized in the relation algebraic sense, we can formulate properties of this relation by relational statements. With a system like RELVIEW (c.f. [3, 2]), such properties can efficiently and simply be checked.

5 Conclusion and future work

We have discussed structured media objects allowing to consider notions of *position* and *link* that relate to the Dexter anchoring mechanism. The measuring system obtained by paths in structured media objects is uniformly used for any kind of media object and allows to specify source and sink of links abstractly, that is separated from concrete media. This allows to exchange media without touching the underlying hyperlink structure. We formalized this approach using algebraic specifications, and thus yield a possibility to analyze a hyperlinks structure in advance of the physical incorporation of a hyperdocument. This is an advantage for testing hyperlink structures especially when large hyperdocuments are considered, or when media change frequently.

The algebraic specification is embedded into more complex structured specifications (c.f. [13], part 2). This allows for high-level specifications which can be implemented easily in the object-oriented programming paradigm. By specifying a hyperdocument in the way we do, we enhance testing and maintenance of hyperdocuments. Since the structured algebraic specifications used in [13] prepare an object-oriented implementation, we think about an entire development process for hyperdocuments that is based on a "standard" software process. This process is currently under investigation [11].

In the future, we can generalize the tree-like structures to acyclic graphs. The notion of position is still valid then, but has to be reformulated regarding the new structure. Graph-searching algorithms like *depth first search* allow to unambiguously address media objects.

The interface of position- and link-creating operations can be expanded depending on applications. We currently specify and implement a hypermedial car cockpit as a case study (c.f. [13], chap. 13). Links are given types, and positions need to be equipped with attributes concerning layout. The implementation concepts need to be adapted to technological aspects such as implementation with Java and Swing using threads and events to communicate, i.e. sending data from one media object to another. These and other technical aspects are currently being reconsidered, and our concept is put to use for complex applications.

Tool support for analyzing hyperlink structures is also mandatory and feasible. The composition of media objects can be done using drag and drop techniques, and the specification of links follows a well-defined interface to positions and links. The structure of the resulting algebras follows certain properties and can thus be established by tool support. The specification can then be used to compute positions and links from the media objects given. Properties of the link structure being checked can be formalized as axioms. Hence, the resulting algebras can be checked to be models by conventional proof methods.

Acknowledgement We thank Prof. Dr. E.-E. Doberkat for his constructive remarks on this work.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] R. Berghammer and Th. Hoffmann. Modeling sequences within the relview system. *Journal of Universal Computer Science*, 7(2):107–123, 2001.

[3] R. Berghammer and Th. Hoffmann. Relational depth-first-search with applications. *Information Sciences*, 139(3-4):167–186, 2001.

[4] C. Brink, W. Kahl, and G. Schmidt, editors. *Relational Methods in Computer Science*. Advances in Computing Science. Springer, 1997.

[5] G. H. Collier. Thoth-II: Hypertext with Explicit Semantics. In *Hypertext '87*, pages 269 – 288, November 1987.

[6] H. M. Deitel, P. J. Deitel, T. R. Nieto, T. M. Lin, and P. Sadhu. *XML How to Program*. Prentice Hall, 2001.

[7] E.-E. Doberkat. A language for specifying hyperdocuments. *Software - Concepts and Tools*, 17:163–172, April 1996.

[8] E.-E. Doberkat. Using logic for the specification of hypermedia documents. In J. Balderjahn, R. Mathar, and M. Schader, editors, *Classification, Data Analysis and Data Highways*, pages 205–212. Springer, 1998.

[9] H. Ehrig, B. Mahr, F. Cornelius, M. Grosse-Rhode, and P. Zeitz. *Mathematisch-strukturelle Grundlagen der Informatik*. Springer, 1999.

[10] A. Fronk. Support for hypertext maintenance. IEEE Computer, June 1999. Letter to the Editor.

[11] A. Fronk. A software-technological approach to the construction of hyerpdocuments. In *I-Know '02 Preconference Workshop, Graz*, 2002. Accepted for publication.

[12] A. Fronk. A tool system for an object-oriented approach to construction and maintenance of hypermedia documents. In W. Gaul and G. Ritter, editors, *Classification, Automation and New Media*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 265 – 272. Springer, 2002.

[13] A. Fronk. *Algebraische Semantik einer objektorientierten Sprache zur Spezifikation von Hyperdokumenten*. PhD thesis, Universität Dortmund, Shaker Verlag, 2002.

[14] A. Fronk and J. Pleumann. Der *DoDL*-Compiler. Memorandum 100, Universität Dortmund, Fachbereich Informatik, Lehrstuhl für Software-Technologie, June 1999. ISSN 0933-7725.

[15] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison-Wesley, 1995.

[16] P. Gloor. *Elements of hypermedia design: techniques for navigation & visualization in cyberspace*. Birkhäuser, 1997.

[17] F. Halasz and M. Schwartz. The dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, February 1994.

[18] L. Hardman, D. C.A. Bulterman, and G. van Rossum. The amsterdam hypermedia model: Adding time and context to the dexter model. *Communications of the ACM*, 37(2):50 – 62, February 1994.

[19] C. B. Jones. *Systematic Software Development using VDM*. Prentice Hall, 1990.

[20] D. Lowe and W. Hall. *Hypermedia & the Web - an engineering approach*. Wiley & Sons, 1999.

[21] V. Mattick and C.-P. Wirth. An algebraic dexter-based hypertext reference model. Technical Report 719, Unversität Dortmund, Fachbereich Informatik, Lehrstuhl 5, November 1999.

[22] S. R. Newcomb, N. A. Kipp, and V. T. Newcomb. The hytime hypermedia/time-based document structuring language. *Communications of the ACM*, 34(11):67 – 83, November 1991.

[23] J. Pleumann. dodl2html - Ein Generator zum Erzeugen von graphspezifizierten Hyperdokumenten. Master's thesis, Universität Dortmund, Fachbereich Informatik, Lehrstuhl für Software-Technologie, 2000.

[24] D. Schwabe, F. Garzotto, and P. Paolini. HDM - A Model for the Design of Hypertext Applications. In *Hypertext '91*, pages 313 – 328, December 1991.

[25] D. Schwabe and G. Rossi. OOHDM. An object-oriented Hypermedia Design Model. Technical report, Pontifícia Universidade Católica do Rio (PUC-Rio), Departamento de Informatica, 1994.

[26] D. Schwabe and G. Rossi. The Object-Oriented Hypermedia Design Model. *Communications of the ACM*, 38(8):45–46, August 1995.

[27] P. Seyer. *Understanding hypertext: concepts and applications*. Windcrest / MacGraw-Hill, 1991.

[28] Graeme Smith. *The Object-Z Specification Language*. Kluwer Academic Publisher, 2000.

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

[30] P. D. Stotts, R. Furuta, and C. R. Cabarrus. Hyperdocuments as automata: Verification of trace-based browsing properties by model checking. *ACM Transactions on Information Systems*, 16(1):1–30, January 1998.

[31] K. Tochtermann. *Ein Modell für Hypermedia*. PhD thesis, Universität Dortmund, Fachbereich Informatik, Lehrstuhl 1, 1994.

[32] J. van Ossenbruggen and A. Eliëns. The Dexter Hypertext Reference Model in Object-Z. Technical report, Vrije Universiteit Amsterdam, Dept. of Mathematics and Computer Science, 1995.

[33] M. Wirsing. Algebraic specifications. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, pages 675 – 788. Elsevier, 1990.