International Journal of Software Engineering
and Knowledge Engineering
Vol. 13, No. 6 (2003) 655–684

World Scientific
www.worldscientific.com

TOWARDS THE ALGEBRAIC ANALYSIS OF HYPERLINK STRUCTURES

ALEXANDER FRONK

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

Submitted 16 August 2002
Accepted 26 January 2003

Structuring media objects such as text or graphics by means of XML is a broadly discussed issue in hypermedia modelling. Thereby, an entire hypermedia document is not only arranged in such a way that different developers may interchange data and have easy access to the inner structure of media objects. 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 hyperlink structures such as reachability, existence of certain paths through a hyperdocument, or dangling links may be verified mathematically in advance of implementing the hyperdocument. Algebraic specifications serve as a formal model which allows to obtain algebras reflecting hyperlink structures and which is open to analyze their static properties.

1. Introduction

A hypermedia document, or hyperdocument for short, is understood as a collection of media objects such as texts, graphics, videos, or interactive ones such as text fields, radio buttons, or forms, which are connected to each other in a non-linear fashion, that is, they are hyperlinked. 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. This fact aggravates not only maintenance of hyperdocuments, but rather does not allow for flexibly exchanging media objects without reconsidering their linkage. Moreover, structuring HTML/XML code is not comparable to structuring code in object-oriented programming: aggregation, inheritance, locality, or genericity are not appropriately available.

Our approach allows in contrast to formally specify the link structure separately from media objects. Since we use algebraic specification, we obtain algebras that allow us to model media objects and their linkage as carrier-sets. Before implement-

ing the hyperdocument, these algebras are open to analyze link structure properties such as reachability, existence of links to ensure certain paths or walk-throughs, dangling links, etc. We adapted algebraic specification to the object-oriented paradigm [11] and can thus refine the hyperdocument specification to an object-oriented program [12]. Thereby, we strictly follow the principle of *separation of concerns*, and contribute to maintaining hyperdocuments immediately on the level of maintaining software. Our object-oriented view on hyperdocuments is realized within a simple yet effective document description language, *DoDL* for short [8, 9]. This language was given a formal semantics [11, 13] to allow the generation of formally specified hyperdocuments. A suitable compiler system [23, 14] allows to realize a hyperdocument by binding specific 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 verify structural properties and to generate arbitrarily many hyperdocuments underlying a specified 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. Figure 1 graphically subsumes our approach.

1.1. *Hyperdocuments are software*

Hyperdocuments are often treated merely as documents leaving their characteristics as programs nearly completely aside, although it is well-known that they inherently possess both document and program qualities. The document/program-dualism can clearly be observed by the fact that hyperdocuments not only offer nonlinearly related media and hence hyperlinked information on the server side. 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 at client-side, a hyperdocument encompasses control structure. Due to our object-oriented description with *DoDL*, we contribute to comprehend more easily hyperdocuments as programs, or more generally, as software.

Moreover, we clearly separate server-side issues from client-side ones. The computational client-server model as a basic support of the functionality of the Web [20] as well as of Hyperwave as an alternative to the Web [21] finds its place in our approach as follows. We chose a complete server-side approach to formally specify and implement a hyperlink structure. Our compiler-system eventually delivers HTML-code to the client. However and aside from object-oriented specification and implementation, the compiler-system could be modified to deliver code for dynamic page creation on the client-side. The concept of specifying and implementing hyperdocuments presented in this paper hence does not depend on how the resulting web pages are delivered to the client. The integrity of a hyperdocument's link structure dynamically requested by a client is verified in analogy to exchanging media objects

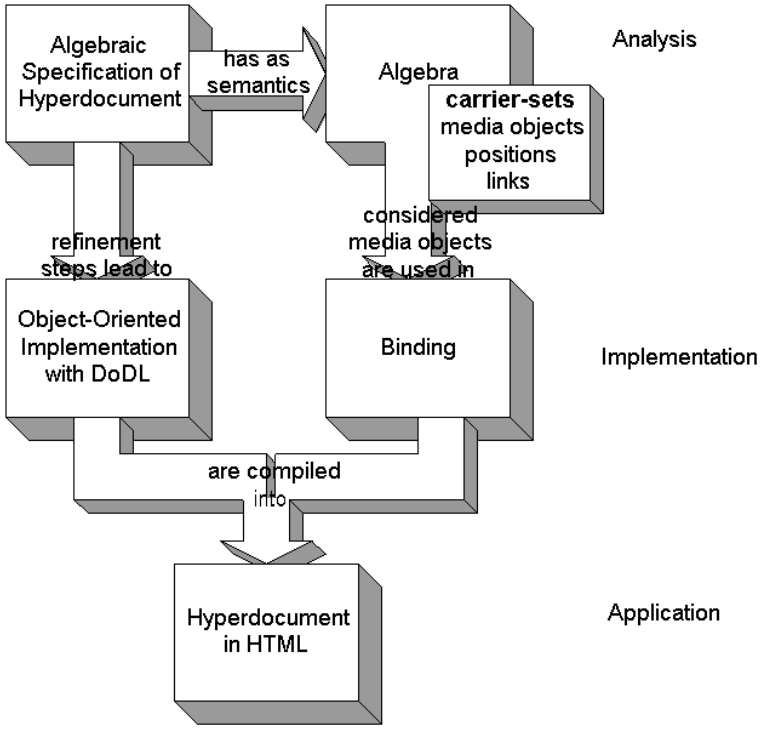

Fig. 1. Our approach to analyzing and implementing hyperlink structures.

on the server-side in advance of creating and delivering HTML-code.

1.2. *Why specify and analyze link structures?*

The possibility to exchange media objects without reconsidering their linkage raises the need to check the properties of the resulting link structure before generating and delivering a web page. Depending on the specific information a hyperdocument has to present, its media objects may underlie even 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 re-implemented. 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 specific 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. Thereupon, static properties of both structured media objects (e.g. the existence of a certain required sub-object) and of the hyperlinks between media objects (e.g. reachability of media objects, validity of links, or even the existence of a certain link) can be checked and verified on a formal level.

The notion of *positions within media objects* 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. In contrast to link checkers working on already encoded hyperdocuments, formally specifying link structures may save development time, is less error-prone, helps to ensure certain properties, and, last but not least, allows to mathematically prove the structural properties of both media objects and their linkage. This is important, for example, in such environments where operating a hyperdocument relies on reasoning about its security.

Links leading to "external" web pages are "out of focus" since the existence of such external pages cannot be guaranteed. Checking (external) link integrity is hence also a concern of our formal approach and will be discussed in Sec. 4.3.

1.3. *Layout issues*

The construction of hyperdocuments is generally not limited to structuring the media objects involved and their link structure (cf. [20], Chap. 9). Moreover, an overall view on *hypermedia systems* has to be taken into account including usability and presentation of hyperdocuments (cf. [20], p. 7). Our approach focuses on the specification and, moreover, on the analysis of hyperlink structures. 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 (cf. [15], p. 264). For our approach, capturing elements and their relative positions to each other is sufficient since the approach is independent from the visual representation of media objects. Our approach only needs to respect the composition of documents, the notion of subdocuments, and positions of subdocuments in the document under consideration in order to argue about and analyze properties of hyperlink structures.

The interaction found in hypermedia systems can both be understood as a

specific functionality of media objects or of links being traversed. We exploited the latter in our HyCop project [1] which will be used throughout this paper as a running example. In this project, a digital and hypermedial car cockpit was realized using and refining the approach discussed in this paper.

This paper, which is an extension of parts 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 analyze the link structure of a hyperdocument in advance of its physical incorporation, i.e. we primarily work on a conceptual level. The paper is organized as follows: Section 2 discusses related work; the structuring mechanism under consideration is introduced in Sec. 3; Sec. 4 discusses the formal analysis of link structures; the paper concludes in Sec. 5 and suggests further work.

2. Related Work

Many environments support the development and construction of hypermedia documents and systems by sets of suitable tools, for example Matilda [19], or Microcosm [17]. Our approach, however, concentrates on the specification of hypermedia documents, i.e. we focus on a phase prior to hyperdocument implementation. We aim at analyzing static aspects of different hyperlink structures evolving from one hyperdocument specification by exchanging its underlying media objects in advance of encoding the hyperdocument. Moreover, our approach directly leads to an object-oriented development process [12].

Furuta and Stotts formally model document structure by Petri nets [28]. Thereby, they concentrate on browsing information. In [29], they use model checking to analyze browsing properties, i.e. dynamic aspects occurring during traversal. We concentrate on static properties of the structure being traversed.

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 modelled object-oriented 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 as we do.

The language HyTime [22] offers tags to structure media objects in a modular way. Like 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 as in our approach.

The XML path language, XPath (cf. [7], 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 encompasses a formalization of a tree structure such as that used in XPath.

The XML linking language, XLink (cf. [7], 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 to be done formally. These XML-related technologies use tree-like structuring mechanisms, and due to their widespread use they are, to this extent, superior to our approach. Nonetheless, they lack the advantage of separating the description of link structures from specific 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.

The Dexter Hypertext Reference Model, as described in [16], 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. The Dexter Reference Model aims at establishing a standard on hypermedia systems in general, that is, it proposes a model each hypermedia system should be an instance of. Since it is understood as a very basis for hypermedia modelling, we relate to this model in more detail in the remainder of this section.

The Dexter Reference 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. A link is a sequence of specifiers, such that arbitrary linkage is feasible.

In our approach, links are modelled 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. 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 vertex addressed by the path, or, equally, its unique number within a tree representation.

Similar to the Dexter model, specific positions depend on specific 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.

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.

3.1. *Basic notions*

In this section, we introduce the notions vital to 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* (cf. [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 modelled by *anchors*, in hypertext they are also called *buttons* (cf. [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, attributed graphs the nodes of which relate to positions, and the edges of which relate to links (cf. [6]).

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 following, 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 labelled with a.
- A compositional, $m = \mathit{compose}(m_1, \ldots, m_n), n \geq 2$, is a tree the root node of which is labelled with m. The root has exactly n children labelled with m_i, such that m_i is the ith son of the root.

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

Definition 2. Let $m = \mathit{compose}(m_1, \ldots, m_n)$ be a media object. 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:

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

Example 1. Figure 2 shows an interactive compositional media object, `Address.dsp`. The media object is part of a hypermedial car cockpit as presented in Example 3 on page 671. It is used to enter a destination which the navigation device may calculate a route to.

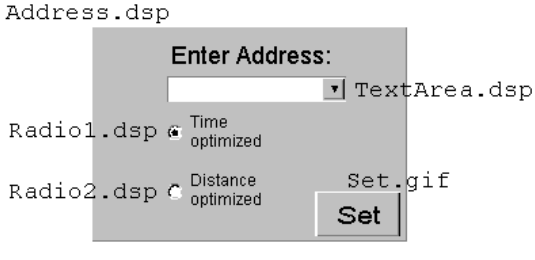

Fig. 2. An interactive compositional media object.

The media object is composed of a text with a text field attached to it underneath. Two radio buttons allow to choose between a time or distance optimized

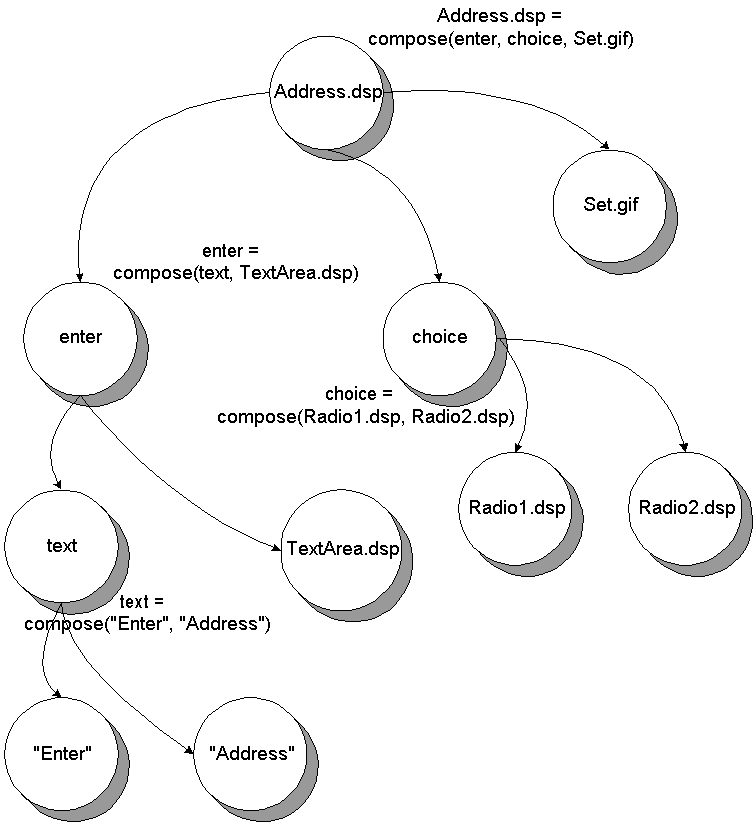

Fig. 3. A tree representation.

route. A set-button allows to leave the interaction and to send data to the navigation device. The media type `dsp` abbreviates "**d**evice **sp**ecification" and says that a media object represents an interactive device providing a certain functionality.

The inner structure of such a media object is purely a matter of choice and may be given by a media designer. For presentational purposes, we assume `Address.dsp` to be composed of three parts, viz an area to enter addresses, a radio button area, and a data delivery area. The address entering area is composed of a text consisting of two words, `Enter` and `Address`, and a text field, `TextArea.dsp`; the radio button area consists of two choices, `Radio1.dsp` and `Radio2.dsp`; the data delivery area only contains a set-button, `Set.gif`. Figure 3 shows the tree representation of `Address.dsp`.

The root represents the entire compositional `Address.dsp` and relates it to the term *compose*(enter, choice, `Set.gif`). Its children, i.e. its direct descendants, form the set of constituents. The sub-media object enter, for example, is composed of text and `TextArea.dsp`. We write enter as *compose*(text, `TextArea.dsp`). Since `"Enter"` and `"Address"` are smallest units, here words, they cannot be further decomposed

and, hence, we cannot refer to their letters. The level which a designer wants to fix atoms on is a matter of choice. Since arbitrary information units can serve as media objects, we can also use any other media types as atoms. Finally, the components of `Address.dsp` are given by the following set:

$$
\begin{aligned}
&\mathit{components}(\texttt{Address.dsp}) :=\\
&\{\texttt{"Enter"}, \texttt{"Address"}, \texttt{TextArea.dsp}, \texttt{Radio1.dsp}, \texttt{Radio2.dsp}, \texttt{Set.gif},\\
&\quad \mathit{compose}(\texttt{"Enter"}, \texttt{"Address"}),\\
&\quad \mathit{compose}(\texttt{Radio1.dsp}, \texttt{Radio2.dsp}),\\
&\quad \mathit{compose}(\mathit{compose}(\texttt{"Enter"}, \texttt{"Address"}), \texttt{TextArea.dsp}),\\
&\quad \mathit{compose}(\mathit{compose}(\texttt{"Enter"}, \texttt{"Address"}),\\
&\qquad\qquad \mathit{compose}(\texttt{Radio1.dsp}, \texttt{Radio2.dsp}, \texttt{Set.gif}))\}.
\end{aligned}
$$

The ordering of the atoms a compositional contains is important. Different orderings yield different compositionals. The same ordering, however, can be represented by trees with different structures. Referring to Example 1 and omitting suffixes, we can compose `Address` by using its atoms directly, that is, `Address` can be written as $\mathit{compose}(\texttt{"Enter"}, \texttt{"Address"}, \texttt{TextArea}, \texttt{Radio1}, \texttt{Radio2}, \texttt{Set})$. In this case, the tree structure of `Address` is simply a root with a sequence of leaves as its children. This consideration leads to three different kinds of equality predicates definable on compositionals. They are based on the notion of *fringe*. A fringe refers only to the leaves of a tree and neglects the tree's inner structure. 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:

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

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 sets 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 sub-media objects without regarding their respective tree structure. They will be used in the definition of weak paths in Sec. 3.3. Representative sets are hence the basic concept to search for components in media objects, i.e. to find all occurrences of any given sub-compositional.

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 following, 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, ..., 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 ith 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

$$\begin{aligned}\mathcal{S}' := \; & \langle \pi_{\#\mathcal{S},1}(\mathcal{S}), \ldots, \pi_{\#\mathcal{S},k-1}(\mathcal{S}) \rangle \\ & \circ \langle comp \rangle \\ & \circ \langle \pi_{\#\mathcal{S},l+1}(\mathcal{S}), \ldots, \pi_{\#\mathcal{S},n}(\mathcal{S}) \rangle\end{aligned}$$

      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 $\mathit{fringe}(m)$, 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 which m is composed of. By reducing the fringe it is guaranteed that each atom is used exactly once 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 sub-media 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 (cf. [2], p. 36). The structure of a media object, m, allows to unambiguously address each sub-media object, m', of m by means of such a numerical string. We call this 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 $m = \mathit{compose}(m_1, \ldots, m_n), n \geq 2$, and m' be two media objects.

A **path** in m to m', $path(m, m')$ for short, is recursively defined as follows:

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

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 this 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:

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

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 also a weak path.

Example 2. Figure 4 shows the compositional media object presented in Example 1 on page 662. To determine paths, for each inner node we enumerate its outgoing edges from 1 to its outdegree.

A path to the atom `"Address"` is given by the string `112`, since the atom occurs as the second part of the first part of `Address.dsp`.

A path to the word `"Time"` found in the description of the first radio button cannot be computed, since the radio button is treated as an atom. A path to the phrase `"Enter Address"` can only be found as the weak path `11` leading to the composite `text`. A path to *compose*(`"Enter"`, `"Address"`), however, would not address an atomic phrase `"Enter Address"`, if contained in `Address.dsp`.

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 `Radio2.dsp` and `Set.gif` if they are not explicitly composed to a (sub-)media object. If the structure of `Address.dsp` is neglected, however, the sequence ⟨`Radio2.dsp`, `Set.gif`⟩ occurs in it. A path to this media object, *compose*(`Radio2.dsp`, `Set.gif`),

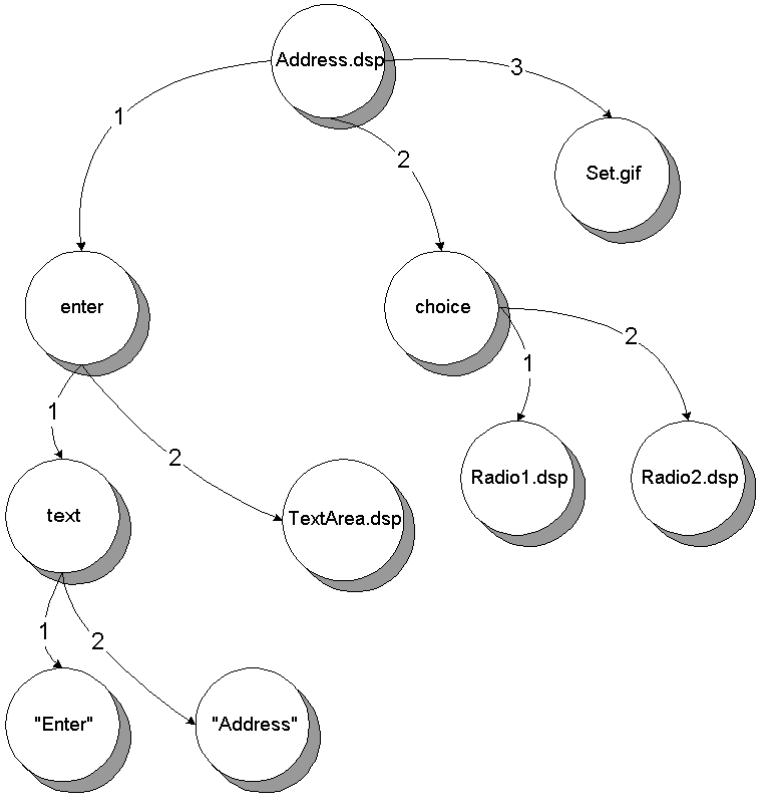

Fig. 4. Paths in a media object.

can be computed, however, if not only `Address.dsp` is considered but also its representative set, $\mathcal{C}_{\texttt{Address.dsp}}$. Then, each possible tree structure of `Address.dsp` is available as search space including at least a term of the form $\mathit{compose}(\ldots, (\mathit{compose}(\texttt{Radio2.dsp}, \texttt{Set.gif}))\ldots)$. Then we can find several paths to $\mathit{compose}(\texttt{Radio2.dsp}, \texttt{Set.gif})$ in different representations of `Address.dsp`. Annotating paths with information about the representative of a media object 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 a media object as an underlying search space aiming at purposely 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 an application. 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:

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

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 [2], p. 36, as follows:

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

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

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, ..., 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. A car cockpit as partially presented in Fig. 5 assembles instruments such as a speedometer and information screens. Additionally, we allow for armature to operate, for example, a CD player or a navigational device. Each information whether displayed textually or graphically on the cockpit is represented by a media object. We assume a car to be supplied with a digital screen such as a touch-screen or some other device allowing to interact with the cockpit. The cockpit may consist of a speedometer (`Velocity.dsp`), a rotation device (`Rotation.dsp`), as well as a navigation device (`Navigation.dsp`, `Address.dsp`) together with some informative texts linked to specific buttons.

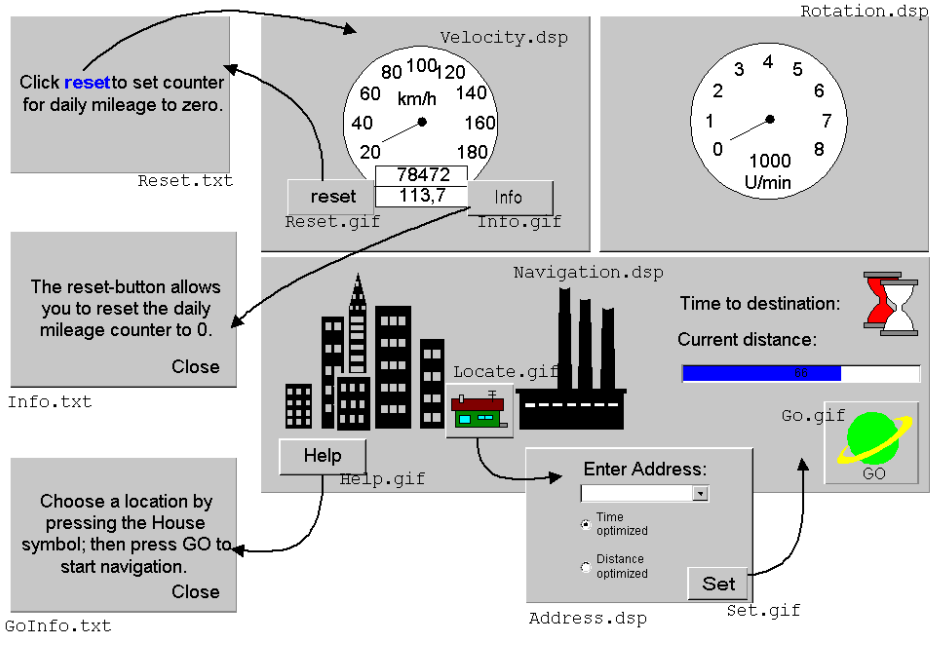

Fig. 5. A complex hypermedial car cockpit.

We only consider the media object `Velocity.dsp` since the link structure encompassing the entire cockpit is given analogously. It consists of a speedometer and a mileage counter, and is treated as an atom. Additionally, we want to attach two buttons to it, viz `Reset.gif` and `Info.gif`, and thus built a compositional media object. The two buttons shell form the beginning and ending of this media object, respectively. This is expressed by the term *compose*(`Reset.gif, Velocity.dsp, Info.gif`). The info button is linked to the text `Info.txt`, and the reset button is linked to the text `Reset.txt`. Therein, the word `reset` occurs as an atomic media object (we always assume texts to be composed of words) which in converse is linked to the media object `Velocity.dsp`. Traversing this link may set the daily mileage counter to zero. Handling this functionality is left to the media objects and is not important for specifying the link structure.

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

$$\begin{aligned}\mathcal{M} := \{&\texttt{Reset.txt}, \texttt{"reset"}, \texttt{Info.txt},\\ &\texttt{Velocity.dsp}, \texttt{Reset.gif}, \texttt{Info.gif},\\ &\mathit{compose}(\texttt{Reset.gif}, \texttt{Velocity.dsp}, \texttt{Info.gif})\}\end{aligned}$$

Note that the word `"reset"` is an atomic media object, and the compositional media object is a member of $\mathcal{M}$ as well. For simplicity again, we assume pictures to be atoms. We omit suffixes for better reading if they are clear from the context.

Each link starting in an occurrence of the word `"reset"` leads to the velocity display. The sink of this link is given by the beginning of `Velocity` (the end of it works equally well), or the second component of the compositional containing `Velocity`. 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. Hence, the set of positions is given by

$$\begin{aligned}\mathcal{P} := \{&\theta(\texttt{Reset}, \texttt{"reset"}), \alpha(\texttt{Velocity}), \alpha(\texttt{Reset}), \alpha(\texttt{Info}),\\ &\alpha(\mathit{compose}(\texttt{Reset, Velocity, Info})),\\ &\omega(\mathit{compose}(\texttt{Reset, Velocity, Info}))\}\end{aligned}$$

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 `"reset"`, $2_{\texttt{Reset.txt}}$, is thus given by its number within the text.

Written as paths, we have the following positions:

$$\begin{aligned}\theta(\texttt{Reset}, \texttt{"reset"}) &= \{2_{\texttt{Reset}}\},\\ \alpha(\texttt{Velocity}) &= 2_{\mathit{compose}(\texttt{Reset, Velocity, Info})}\\ &= \epsilon_{\texttt{Velocity}},\end{aligned}$$

$$\begin{aligned}
\alpha(\texttt{Reset}) &= \texttt{"Click"} = 1_{\texttt{Reset}},\\
\alpha(\texttt{Info}) &= \texttt{"The"} = 1_{\texttt{Info}},\\
\alpha(\mathit{compose}(\texttt{Reset, Velocity, Info})) &= 1_{\mathit{compose}(\texttt{Reset, Velocity, Info})}\\
&= \epsilon_{\texttt{Reset}},\\
\omega(\mathit{compose}(\texttt{Reset, Velocity, Info})) &= 3_{\mathit{compose}(\texttt{Reset, Velocity, Info})}\\
&= \epsilon_{\texttt{Info}}
\end{aligned}$$

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

$$\mathcal{L} := \{(2_{\texttt{Reset.txt}}, \epsilon_{\texttt{Velocity}}), (\epsilon_{\texttt{Reset.gif}}, 1_{\texttt{Reset.txt}}), (\epsilon_{\texttt{Info.gif}}, 1_{\texttt{Info.txt}})\}$$

Depending on the media object `Reset.txt`, the set of occurrences of the word `"reset"` may differ: we may obtain more than one link, if each occurrence is specified to be connected to the velocity display. The set may even be empty, if `"reset"` does not occur in `Reset.txt` at all.

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 Sec. 4.2, where the hyperdocument discussed in Example 3 on page 671 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, $\Sigma = \mathit{sig}(SP)$, and a set of formulas, $E = \mathit{axms}(SP)$, 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** $f^{\mathcal{A}}$: $A_{s1} \times \cdots \times A_{sn} \to A_s$ for each f: $s_1 \times \cdots \times s_n \to s \in \Gamma$.

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 $\mathit{Alg}(\Sigma, E)$.

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

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 $A'|_\Sigma$ is called Σ-**reduct** of $\mathcal{A}'$, if for each $s \in S$ the carrier-set $(A'|_\Sigma)_s$ is defined as A'_s, and for each $f \in \Gamma$ the operation $f^{\mathcal{A}'|_\Sigma}$ is defined as $f^{A'}$.

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

$$\mathit{sig}(\mathit{import}\ SP\ \mathit{into}\ SP') := \mathit{sig}(SP) \cup \mathit{sig}(SP')$$

$Mod(import\ SP\ into\ SP') := \{\mathcal{A} \in Alg(\Sigma_{import\ SP\ into\ SP'}) | \mathcal{A}|_{\Sigma(SP)} \in Mod(SP)\}$

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 Sec. 3.2 can be specified algebraically as shown in specification 1 on page 20.

SPECIFICATION 1. Compositional media objects.

$COMPOSITIONAL =$

sorts $MedObj$

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

axms

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

The sort *MedObj* represents the universe of media objects, $\mathcal{MO}$. The operation symbols follow the names given in Sec. 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 2 on page 21, and a specification for tuples, which is given in specification 3 on page 21.

SPECIFICATION 2. A specification for sets.

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

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

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

$$\begin{aligned}
&(11) && insert(insert(a,y),x) =_{set(s)} insert(insert(a,x),y),\\
&(12) && insert(delete(a,x),x)) =_{set(s)} insert(a,x),\\
&(13) && delete(empty,x) =_{set(s)} empty,\\
&(14) && delete(insert(empty,x),x) =_{set(s)} empty,\\
&(15) && (delete(insert(a,x),x) =_{set(s)} a) =_{bool} \neg(x \in_s a),\\
&(16) && (a =_{set(s)} b) =_{bool} (a \subset_s b) \wedge (b \subset_s a),\\
&(17) && (empty \subset_s a) =_{bool} true,\\
&(18) && (insert(a,x) \subset_s b) =_{bool} (a \subset_s b) \wedge (x \in_s b),\\
&(19) && (a \subset_s insert(b,x)) =_{bool} (a \subset_s b),\\
&(20) && ((a \subset_s delete(b,x)) \rightarrow a \subset_s b) =_{bool} true,\\
&(21) && (a \subset_s b \wedge b \subset_s c \rightarrow a \subset_s c) =_{bool} true,\\
&(22) && (x \in_s insert(a,x)) =_{bool} true,\\
&(23) && (x \in_s delete(a,x)) =_{bool} false
\end{aligned}$$

SPECIFICATION 3. A specification for tuples.

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

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

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

$$(24) \qquad \pi_{n,i}(tup^n(x_1, \ldots, x_n)) =_{s_i} x_i, \qquad i = 1, \ldots, n$$

$$(25) \qquad (t_1 =_{\langle s_1,\ldots,s_n \rangle} t_2) =_{bool} \bigwedge_{i=1}^{n} (\pi_{n,i}(t_1) =_s \pi_{n,i}(t_2)),$$

$$(26) \qquad t =_{\langle s_1,\ldots,s_n \rangle} tup^n(\pi_{n,1}(t), \ldots, \pi_{n,n}(t)),$$

$$(27) \qquad tup^n(x_1, \ldots x_n) \circ tup^m(y_1, \ldots, y_m) =_{\langle s_1,\ldots,s_{n+m} \rangle} tup^{n+m}(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

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_*: $bool \times s \times s \to s$ 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 [30], on pages 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 N for which $f(i)$ holds.

SPECIFICATION 4. An algebraic signature for hyperdocuments.

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

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

With these specifications, we can specify our hyperdocuments under consideration. Specification 4 on this page imports *COMPOSITIONAL* and introduces new sorts for positions and links. The operation symbols again follow those used in Secs. 3.3 and 3.4. The constant ϵ represents the empty path, and $error_{position}$ reflects the case when 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 *HDOC* provides the axioms for the signature given in $HDOC'$.

We consider Example 3 again and show how specification *HDOC* is used to formally describe the cockpits link structure (see Spec. 6) introduced in the example.

Example 4. First, we need a new sort, *cockpit*, for technical reasons. The sorts *MedObj*, *position* and *link* are already introduced through *HDOC*. Each media object is represented by a unary operation. The operations *linkWord*, *linkRes* and *linkInf* are used to specify the desired hyperlinks. The axioms fix their semantics and describe the compositional media object named *display*. *linkWord* specifies a set of links starting at each occurrence of a specific word in the text represented by *reset* and ending in the beginning of the velocity display represented by *veloc*. The operation ψ, defined in *HDOC*, pairs two positions to a link, whereas the operation

SPECIFICATION 5. An algebraic specification for hyperdocuments – contd.

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

$$ispath(compose^n(m_1, \ldots, m_n), m', i) = \exists m'' : MedObj \,. \quad m'' \;\partial\; part_n(compose^n(m_1, \ldots, m_n), i) \wedge m'' \approx m', \qquad i = 1, \ldots, n \tag{28}$$

$$path_{\approx}(compose^n(m_1, \ldots, m_n), m') = \mathsf{if}\ compose^n(m_1, \ldots, m_n) \approx m'\ \mathsf{then}\ \epsilon\ \mathsf{else}\ \mathsf{if}\ \exists i : nat.ispath(compose^n(m_1, \ldots, m_n), m', i)\ \mathsf{then}\ (\mu i.ispath(compose^n(m_1, \ldots, m_n), m', i)) \frown path_{\approx}(part_n(compose^n(m_1, \ldots, m_n), i), m')\ \mathsf{else}\ error_{position}, \qquad i = 1, \ldots, n \tag{29}$$

$$\alpha(m) = \epsilon, \tag{30}$$

$$\alpha(compose^n(m_1, \ldots, m_n)) = 1 \frown \alpha(part_n(compose^n(m_1, \ldots, m_n), 1)), \tag{31}$$

$$\omega(m) = \epsilon, \tag{32}$$

$$\omega(compose^n(m_1, \ldots, m_n)) = n \frown \omega(part_n(compose^n(m_1, \ldots, m_n), n)), \tag{33}$$

$$\theta(compose^n(m_1, \ldots, m_n), m') = \mathsf{if}\ compose^n(m_1, \ldots, m_n) \approx m'\ \mathsf{then}\ \{\epsilon\}\ \mathsf{else}\ \mathsf{if}\ \exists i : nat.ispath(compose^n(m_1, \ldots, m_n), m', i)\ \mathsf{then}\ \{i \frown p \mid ispath(compose^n(m_1, \ldots, m_n), m', i), p \in \theta(part_n(compose^n(m_1, \ldots, m_n), i), m')\}\ \mathsf{else}\ \{\,\}, \qquad i = 1, \ldots, n \tag{34}$$

$$(l \in \psi_{set(link)}(pset, p)) = (source(l) \in pset \wedge sink(l) = p) \tag{35}$$

$$source(\psi(p, q)) = p, \tag{36}$$

$$sink(\psi(p, q)) = q, \tag{37}$$

$$(l =_{link} l') = ((source(l) = source(l')) \wedge (sink(l) = sink(l'))) \tag{38}$$

$\psi_{set(link)}$ yields a set of links by pairing each element of a set of positions with exactly one other position. The former positions are used as different link sources, all leading to the same sink. The operation *linkRes* and *linkInf* specify links from the beginning and the end of *display* to the beginning of the texts represented by *reset* and *info*, respectively.

This specification can be implemented in *DoDL* as shown in Listing 1 on page 25. We use a class, `Cockpit`, to collect the involved media objects as its attributes, called document variables. They are introduced by the keyword `documents` and are of type `MedObj`, representing media objects. Within the typed binding-section, document variables are assigned media objects taken from some repository. Here, we can simply exchange them to generate arbitrarily many different hyperdocuments while still using the same specification of its link structure. This link structure is defined in class methods, introduced by the keyword `construct`. The method `display` is used to compose the media object referred to as *display* in Spec. 4. It returns the value of the method variable `comp`. The `compose`-method is predefined in *DoDL*. The method `createLinks` is responsible for constructing the desired links.

SPECIFICATION 6. A cockpit specification.

$COCKPIT =$ **import** $HDOC$ **into**
sorts $cockpit$
opns $reset : cockpit \to MedObj$,
$info : cockpit \to MedObj$,
$veloc : cockpit \to MedObj$,
$resBut : cockpit \to MedObj$,
$infBut : cockpit \to MedObj$,
$word : cockpit \to MedObj$,
$display : cockpit \to MedObj$,
$linkWord : cockpit \times MedObj \to set(link)$,
$linkRes : cockpit \times MedObj \to link$,
$linkInf : cockpit \times MedObj \to link$
vars $c : cockpit$
axms $display(c) = compose(resBut(c), veloc(c), infBut(c))$,
$linkWord(c, reset(c)) = \psi_{set(link)}(\theta(reset(c), word(c)), \alpha(veloc(c)))$,
$linkRes(c, resBut(c)) = \psi(\alpha(display(c)), \alpha(reset(c)))$,
$linkInf(c, infBut(c)) = \psi(\omega(display(c)), \alpha(info(c)))$

The methods `getBegin`, `getEnd`, `getOcc` and `setLink` are also predefined in *DoDL*. They correspond to α, ω, θ, and ψ as defined in Sec. 3. This implementation can be translated by a compiler into a program executed on a machine. Doing so, the program generates a hyperdocument currently encoded in HTML.

The advantages mentioned in the introduction become obvious here: by exchanging the values given in the binding, we are able to generated arbitrarily many hyperdocuments underlying a certain link structure. Rearranging the compositional, if desired, is simply done by changing method `display`. Alternatively, a compositional media object can be defined directly within the binding-section as shown in Listing 2. In this case, method `createLinks` has to be adapted. This is a matter of choice and depends on how flexible a media designer allows the system to change.

In the example, the choice of the text `Reset.txt` is worth noting. We can generate different hyperdocuments with different positions and different links simply by exchanging this text. Hence, it is not clear from a specification such as that shown in Spec. 6 which specific hyperdocument is obtained. Its link structure depends on the choice of media objects. They are given in an algebra as a specific carrier-set for sort *MedObj*. Implementing a hyperdocument in the way we do makes it mandatory to analyze its link structure in advance of the hyperdocument's incorporation, and specifying a hyperdocument in the way we do makes it possible to do so.

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

LISTING 1. A *DoDL*-listing.

```
class Cockpit is
  documents reset, word, info, veloc, resBut, infBut: MedObj;
  construct
    MedObj display(void){
      MedObj comp;
      comp = compose(resBut, veloc, infBut);
      return comp;
    };

    void createLinks(void){
            // linking the word
      reset.getOcc(word).setLink(veloc.getBegin());
            // linking the reset button
      (display().getBegin()).setLink(reset.getBegin());
            // linking the info button
      (display().getEnd()).setLink(info.getBegin());
    };
end Cockpit;

binding Cockpit is
  reset: text = Reset.txt;
  word: text = "reset";
  info: text = Info.txt;
  veloc: graphics = velocity.dsp;
  resBut: graphics = Reset.gif;
  infBut: graphics = Info.gif;
end;
```

LISTING 2. An alternative definition for compositionals.

```
binding Cockpit is
  reset: text = Reset.txt;
  word: text = "reset";
  info: text = Info.txt;
  veloc: graphics = velocity.dsp;
  resBut: graphics = Reset.gif;
  infBut: graphics = Info.gif;
  display: MedObj = compose(resBut, veloc, infBut);
end;
```

a certain set of media objects to which a link structure applies. 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 as given by Def. 13. All other sorts can be interpreted arbitrarily. We reconsider Example 3 on page 671, but now more formally.

Example 5. Let $\mathcal{C}$ be a $\Sigma_{COCKPIT}$-algebra, and let the carrier-set for *MedObj* equal the set $\mathcal{M}$ of Example 3:

$$\begin{aligned} C_{MedObj} = \{&\texttt{Reset.txt}, \texttt{"reset"}, \texttt{Info.txt},\\ &\texttt{Reset.gif}, \texttt{Info.gif}, \texttt{Velocity.dsp},\\ &\mathit{compose}(\texttt{Reset.gif}, \texttt{Velocity.dsp}, \texttt{Info.gif})\} \end{aligned}$$

Hence, certain documents are given by interpretation of the sort *MedObj*. With this carrier-set at hand, we can interpret the operations representing media objects:

$$\begin{aligned} \mathit{reset}^{\mathcal{C}}(c) &= \texttt{Reset.txt}, & \mathit{word}^{\mathcal{C}}(c) &= \texttt{"reset"},\\ \mathit{info}^{\mathcal{C}}(c) &= \texttt{Info.txt}, & \mathit{veloc}^{\mathcal{C}}(c) &= \texttt{Velocity.dsp},\\ \mathit{resBut}^{\mathcal{C}}(c) &= \texttt{Reset.gif}, & \mathit{infBut}^{\mathcal{C}}(c) &= \texttt{Info.gif} \end{aligned}$$

The interpretation of the compositional media object represented by *display* is term-generated:

$$\mathit{display}^{\mathcal{C}}(c) = \mathit{compose}^{\mathcal{C}}(\mathit{resBut}^{\mathcal{C}}(c), \mathit{veloc}^{\mathcal{C}}(c), \mathit{infBut}^{\mathcal{C}}(c))$$

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

$$\begin{aligned} \mathit{linkWord}^{\mathcal{C}}(c, \mathit{reset}^{\mathcal{C}}(c)) &= \psi^{\mathcal{C}}_{set(link)}(\theta^{\mathcal{C}}(\mathit{reset}^{\mathcal{C}}(c), \mathit{word}^{\mathcal{C}}(c)), \alpha^{\mathcal{C}}(\mathit{veloc}^{\mathcal{C}}(c))),\\ \mathit{linkRes}^{\mathcal{C}}(c, \mathit{resBut}^{\mathcal{C}}(c)) &= \psi^{\mathcal{C}}(\alpha^{\mathcal{C}}(\mathit{display}^{\mathcal{C}}(c)), \alpha^{\mathcal{C}}(\mathit{reset}^{\mathcal{C}}(c))),\\ \mathit{linkInf}^{\mathcal{C}}(c, \mathit{infBut}^{\mathcal{C}}(c)) &= \psi^{\mathcal{C}}(\omega^{\mathcal{C}}(\mathit{display}^{\mathcal{C}}(c)), \alpha^{\mathcal{C}}(\mathit{info}^{\mathcal{C}}(c))) \end{aligned}$$

The other operations introduced by *HDOC* are interpreted in $\mathcal{C}$ either following the axioms given in *HDOC*, or their interpretation is term-generated. For example, *compose* and α are interpreted as follows:

$$\begin{aligned} \mathit{compose}^{\mathcal{C}}(m_1, \ldots, m_n) \;=\; & \mathit{compose}(m_1, \ldots, m_n)\\ & \text{for each } m_i \in C_{MedObj}, i = 2, \ldots, n \end{aligned}$$

$$\alpha^{\mathcal{C}}(m) := \begin{cases} \epsilon^{\mathcal{C}}, & \text{if } m \text{ is atomic}\\ \{1\} \circ \alpha^{\mathcal{C}}(\mathit{part}^{\mathcal{C}}_n(m, 1)), & \text{else} \end{cases}$$

It is easy to prove that $\mathcal{C}$ is a model for specification *COCKPIT*. We can construct specification *HDOC* in such a way that its model class corresponds to the class of free term-algebras (cf. [10], Chap. 10.5) over the set of atomic media objects. We avoid this construction, however, to keep the presentation technically simple.

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 carrier-sets $C_{position}$ and C_{link} which equal the set of paths $\mathcal{P}$ and the set of positions $\mathcal{L}$ as given in Example 3, respectively:

$$\begin{aligned} C_{position} &= \{2_{\texttt{Reset}}, \epsilon_{\texttt{Velocity}}, 1_{\texttt{Reset}}, 1_{\texttt{Info}}, \epsilon_{\texttt{Reset}}, \epsilon_{\texttt{Info}}\}\\ C_{link} &= \{(2_{\texttt{Reset.txt}}, \epsilon_{\texttt{Velocity}}), (\epsilon_{\texttt{Reset.gif}}, 1_{\texttt{Reset.txt}}),\\ &\quad (\epsilon_{\texttt{Info.gif}}, 1_{\texttt{Info.txt}})\} \end{aligned}$$

Since we know that our definition of hyperdocuments given in Def. 13 on page 671 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 to the position of not specifying a hyperdocument's link structure. This specification, however, is crucial to compute a specific 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 Sec. 3.

In the remainder of this section, we discuss some properties of hyperlink structures and show how they can be checked on carrier-sets. Note that link structures depend 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 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 C_{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 the elements in C_{link}, or by using hashing if the structure of C_{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 there exist pairs of the form (p_m, p'_m).

Reachability is a major concern. We can check this property by following sequences of the form $\langle (p_m, p_{1m_1}), (p_{2m_1}, \ldots), \ldots, (\ldots, p'_{m'}) \rangle$ to see that a media object m' is reachable from m. Thereby, entire paths through a hyperdocument can be proven to exist. This is a useful feature for verifying guided tours where certain media objects need to be visited in a certain order.

Dangling links, however, do not occur in our approach and hence can not be checked on the specification level. Each media object participating in a hyperlink structure is under specified control. A link to a specified yet not existing position in one of these media objects is simply not computed when a specification is interpreted as shown in the previous example. Therefore, dangling links may only occur in implemented hyperdocuments. In this case, however, such a hyperdocument cannot satisfy any specification and is thus not a valid implementation. Nonetheless, to ensure a hyperdocument is free of dangling links, each media object hosting possible sinks must be made controllable. Since this is not always possible, for example if one needs to refer to an external web site, the concept of "void" media objects can be introduced into our approach. That is, an empty media object $\emptyset_m$ can be used as place holder for positions beyond specification scope. Our approach will then compute a position of the form $\epsilon_{\emptyset_m}$. In the carrier-set of *MedObj*, an URL can be given for $\emptyset_m$ such that a position is computed and a link is generated. Within the refinement process yielding a valid implementation, these positions are additionally considered, and a link creating method may return an error-message if the desired link cannot be created.

These and further properties can be formalized and tested mechanically depend-

ing on the structure of C_{link}. We are currently checking if relational methods (cf. [5]) can do a good service here, since the link set is a relation on positions. Formalized in relational algebra, we can formulate properties of this relation by relation-algebraic formulae. Together with the RelView-system (cf. [4, 3]), such properties can efficiently be checked in advance of a hyperdocument's implementation.

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, i.e. separated from specific 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 hyperlink structure in advance of the physical incorporation of a hyperdocument. This is mandatory for checking hyperlink structures especially when media objects change frequently and thus it is required to reconsider their linkage.

We generally use differently structured algebraic specifications; the simple ones shown in this paper are embedded into [11]. This allows for high-level specifications which can be implemented easily in an object-oriented fashion. By specifying a hyperdocument in the way we do, we enhance testing and maintenance of hyperdocuments. Since the structured algebraic specifications we use prepare an object-oriented implementation, we were able to think about an entire development process for hyperdocuments that is based on a "standard" software process. This process is elaborated in [12]. Implementing a hyperdocument in the way we do makes it mandatory to analyze its link structure in advance of the hyperdocument's incorporation, and specifying a hyperdocument in the way we do makes it possible to do so.

The approach presented here has been used in our HyCop project, the realization of a digital and hypermedial car cockpit (cf. [11], Chap. 13), as a large scale case study. The interfaces of position- and link-creating operations were expanded regarding this application. Links were given types such that a weakness of our approach reducing expressiveness was resolved, and a browsing behavior was specified to realize both interaction with the user and between media objects. Positions were equipped with attributes concerning layout. The implementation concepts used in the compiler-system were technically reconsidered and adapted to aspects such as implementation with Java and Java-Swing using threads and events to communicate, i.e. for sending data between media objects. In this case study [1], we generalized the tree-like structures of media objects to acyclic graphs. The notion of position is still valid then, but has to be reformulated regarding the new structure. This case study showed that the approach presented in this paper can be put to work, but issues regarding technically aspects therefore need to be con-

sidered. However, on the conceptual level the approach remained valid and useful.

Tool support for analyzing specified hyperlink structures is also mandatory and feasible, since the specification of hyperlink structures 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. In our HyCop project, specifying the structure of media objects and their linkage was deferred to a language based on XML such that some relevant properties such as link integrity could be checked by suitable parsing procedures. We noticed that in analogy to usual software development formal specification may be awkward but cannot fully be replaced by any informal approach, and that the refinement steps proposed in [12] lead to the right direction: whenever media objects may change and safety is a major concern, we are well-advised to deduce an implementation from our specification which is provably correct with respect to the desired properties.

Acknowledgements

The author gratefully thanks Prof. Dr. E.-E. Doberkat for his constructive remarks on this paper.

References

1. K. Alfert, A. Fronk, and Ch. Veltmann, "Endbericht der Projektgruppe 415: Konzeption und Implementierung eines digitalen und hypermedialen Automobilcockpits", Memorandum 138, Lehrstuhl für Software-Technologie, Fachbereich Informatik, Universität Dortmund, 2003.
2. F. Baader and T. Nipkow, *Term Rewriting and All That*, Cambridge University Press, 1998.
3. R. Berghammer and Th. Hoffmann, "Modeling sequences within the relview system", *Journal of Universal Computer Science* **7**(2) (2001) 107–123.
4. R. Berghammer and Th. Hoffmann, "Relational depth-first-search with applications", *Information Sciences* **139**(3–4) (2001) 167–186.
5. C. Brink, W. Kahl, and G. Schmidt (eds.), *Relational Methods in Computer Science*, Advances in Computing Science. Springer, 1997.
6. G. H. Collier, "Thoth-II: Hypertext with explicit semantics", in *Hypertext '87*, November 1987, pp. 269–288.
7. H. M. Deitel, P. J. Deitel, T. R. Nieto, T. M. Lin, and P. Sadhu, *XML — How to Program*, Prentice Hall, 2001.
8. E.-E. Doberkat, "A language for specifying hyperdocuments", *Software — Concepts and Tools*, 17:163–172, April 1996.
9. E.-E. Doberkat, "Using logic for the specification of hypermedia documents", in J. Balderjahn, R. Mathar, and M. Schader (eds.) *Classification, Data Analysis and Data Highways*, Springer, 1998, pp. 205–212.
10. H. Ehrig, B. Mahr, F. Cornelius, M. Grosse-Rhode, and P. Zeitz, *Mathematisch-strukturelle Grundlagen der Informatik*, Springer, 1999.

11. A. Fronk, *Algebraische Semantik einer objektorientierten Sprache zur Spezifikation von Hyperdokumenten*, PhD thesis, Universität Dortmund, 2001, Shaker Verlag, 2002.
12. A. Fronk, "An object-oriented approach to design, specification and implementation of hypermedia documents based on usual software development", in *JUCS Special Issue: Hypermedia — State of the Art 2002* **8** (2002) 892–912.
13. A. Fronk, "An approach to algebraic semantics of object-oriented languages", in *Proc. 7th Brazilian Symposium on Programming Languages*, Brazil, 2003.
14. A. Fronk and J. Pleumann, "Der *DoDL*-Compiler", Memorandum 100, Universität Dortmund, Fachbereich Informatik, Lehrstuhl für Software-Technologie, June 1999.
15. P. Gloor, *Elements of Hypermedia Design: Techniques for Navigation & Visualization in Cyberspace*, Birkhäuser, 1997.
16. F. Halasz and M. Schwartz, "The dexter hypertext reference model", *Commun. ACM* **37**(2) (1994) 30–39.
17. W. Hall, H. Davis, and G. Hutchings, *Rethinking Hypermedia: The Microcosm Approach*, Kluwer, 1996.
18. L. Hardman, D. C.A. Bulterman, and G. van Rossum, "The Amsterdam hypermedia model: Adding time and context to the dexter model", *Commun. ACM* **37**(2) (1994) 50–62.
19. D. Lowe and A. Ginige, "Matilda: A framework for the representation and processing information in multimedia applications", in *3rd Int. Interactive Multimedia Symposium*, Perth, Australia, 1996.
20. D. Lowe and W. Hall, *Hypermedia & the Web — An Engineering Approach*, Wiley & Sons, 1999.
21. H. Maurer and N. Scherbakov, "Document linking and embedding: A new hypermedia structuring paradigm", in P. Brusilovsky, P. Kommers, and N. Streitz (eds.), *Multimedia, Hypermedia and Virtual Reality: Models, Systems and Applications*, Lecture Notes in Computer Science (LNCS), Springer, 1996, pp. 17–27.
22. S. R. Newcomb, N. A. Kipp, and V. T. Newcomb, "The hytime hypermedia/time-based document structuring language", *Commun. ACM* **34**(11) (1991) 67–83.
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*, December 1991, pp. 313–328.
25. D. Schwabe and G. Rossi, "OOHDM. An object-oriented hypermedia design model", Technical report, Pontifcia Universidade Catlica do Rio (PUC-Rio), Departamento de Informatica, 1994.
26. D. Schwabe and G. Rossi, "The object-oriented hypermedia design model", *Commun. ACM* **38**(8) (1995) 45–46.
27. P. Seyer, *Understanding Hypertext: Concepts and Applications*, Windcrest/MacGraw-Hill, 1991.
28. P. D. Stotts and R. Furuta, "Petri-net-based hypertext: Document structure with browsing semantics", *ACM Transactions on Information Systems* **7**(1) (1989) 3–29.
29. 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) (1998) 1–30.
30. M. Wirsing, "Algebraic specifications", in J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, Elsevier, 1990, pp. 675–788.