A formalization of the Amsterdam Hypermedia Model[*]

Jacco van Ossenbruggen, *Lynda Hardman*, *Anton Eliëns*

email: jrvosse@cs.vu.nl, eliens@cs.vu.nl
Vrije Universiteit, Fac. of Mathematics and Computer Sciences
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

email: Lynda.Hardman@cwi.nl
Multimedia and Human Computer Interaction, CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

A1.1 Introduction

In order to be able to compare the way documents are handled in various hypermedia systems and to be able to define interchange and interoperability standards we need to specify the behavior of such systems and their underlying document model in a non-ambiguous manner.

For hypertext systems, we already have such a non-ambiguous description in the form of the Dexter hypertext reference model [8] and its formal specification in the Z specification language [7]. The Dexter model describes atomic, link, anchor and composition structures in hypertext documents. This provided the hypertext community with a way of comparing the documents created by already existing hypertext systems and to design new systems which followed the (encompassing) Dexter model more closely.

As the use of dynamic media, such as audio and video, in documents increases, and as linking parts of these documents becomes more common, there is a need to be able to compare these more complex hypermedia structures. The Amsterdam hypermedia model (AHM) goes towards describing a hypermedia document including temporal and spatial relationships among constituent media

[*]This appendix is a preliminary result of work which will be an integral part of the PhD thesis of Jacco van Ossenbruggen, expected to appear in 1998 at the Vrije Universiteit, Amsterdam.

elements, and also pays attention to defining the behavior of links among groups of (dynamic) media items. While Dexter does not specify the internal structure of the information which describes how a component is to be presented at run time, the AHM explicitly defines that part of the presentation information which describes the spatio-temporal layout.

The description of the AHM has, until now, relied on informal descriptions and its (partial) implementation in the CMIFed system. For the model to be more useful to the hypertext community it needs to be described in a precise way. A more formal description of the AHM should identify the exact boundaries of the model, and facilitate the comparison the AHM with other hypermedia models.

This appendix formalizes the AHM model as described in chapter 3 of this thesis. The Object-Z specification language [3, 4, 5] is used in order to be able to present the formal part of the AHM in an incremental and modular way. Object-Z is an object-oriented extension of Z, the language originally used in [7] to formalize the Dexter model.

The formal specification has been developed in close cooperation with the author of this thesis. The specification process helped to abstract from the implementation details of CWI's authoring and play-out environment CMIFed, and to keep the model as generic as possible. Furthermore, the specification process helped to find inconsistencies and design flaws in earlier versions of the model. Parts of the model are not implemented in the CMIFed system, and a formal approach proved to be useful to check especially these parts of the model.

However, the formal specification has not been developed to prove correctness of the model, nor to prove the correct behavior of systems implementing the model. Instead, the specification gives a precise description of the model, which facilitates a deeper understanding of the more complex concepts of the AHM, and allows comparison with other hypermedia models.

The specification given is based upon the description of the AHM as described in this thesis and differs significantly from the AHM as described in [9].

A1.2 Background

The Amsterdam Hypermedia Model provides an abstraction of, and an extension to, the hypermedia model implemented by the CMIFed hypermedia authoring environment. In contrast to many of the hypertext systems which formed the basis of the Dexter model, CMIFed is originally a multimedia system, extended by hyperlink support at a late stage in its development. This firm background in multimedia explains the central role of temporal relationships in the composition mechanisms of the model, the way the behavior of hyperlinks is related to these structures and the explicit modeling of spatio-temporal layout.

When compared to the Dexter model, the main extensions of the AHM are a specific semantics for composite components, its notion of link context and explicitly defined spatio-temporal layout. The formal specification given in the following sections will focus on these three topics.

Composition structures In contrast to the pure abstract composition facilities of the Dexter model, the AHM defines two specific composition mechanisms: temporal and atemporal composition.

Temporal composition allows the grouping of media items by placing them on the same time axis. This type of composition is common in multimedia systems. Examples include parallel and sequential compositions. By using temporal composition exclusively, the resulting hypermedia document represents a strictly linear multimedia presentation. Hyperlinks can only be used to jump back and forward within the same document, or to start the presentation of another document. Temporal composition does not allow the author to create a hyperlink to parts of the presentation which are optional, and which would not be played if the user never selected the corresponding link.

To allow the inclusion of optional material, the AHM introduces the notion of atemporal composition. Atemporal composition allows grouping of elements which represent alternatives which are accessible by means of hyperlinking. Media objects grouped within an atemporal composite can be part of the main linear stream of the presentation if their initial activation state is $PLAY$ or $PAUSE$. This state can also have the value $INACTIVE$. Objects that are initially paused or inactive require explicit user interaction to be played.

For example, imagine a hypermedia presentation which includes two text entries of a glossary. There are two reasons for not including the entries within a temporal composite, but to use an atemporal composite, combined with an $INACTIVE$ initial activation state. Typically, the entries should only be presented after an explicit request from the user, so both entries are only accessible by link traversal. If they were included in the document by temporal composition only, they would always be presented. Another reason for not using a temporal composite is the absence of obvious temporal relations. While there is a clear structural relation between the two entry components and the glossary component, there is neither a temporal relation between the glossary and the entry components nor a mutual temporal relation between the two entry components. Note that the entries may be displayed in a pop-up window, and in this case there is no spatial relationship between the components. The current version of AHM, however, does not discriminate between spatial and aspatial composition. Temporal and atemporal composition within the AHM will be defined in section A1.6.

Link context In a hypermedia document an end-user can navigate through information by following links defined within the document structure. In an environment with dynamic media, it becomes more important to define the scope of the document affected by the link. To be able to define the scope of a hyperlink

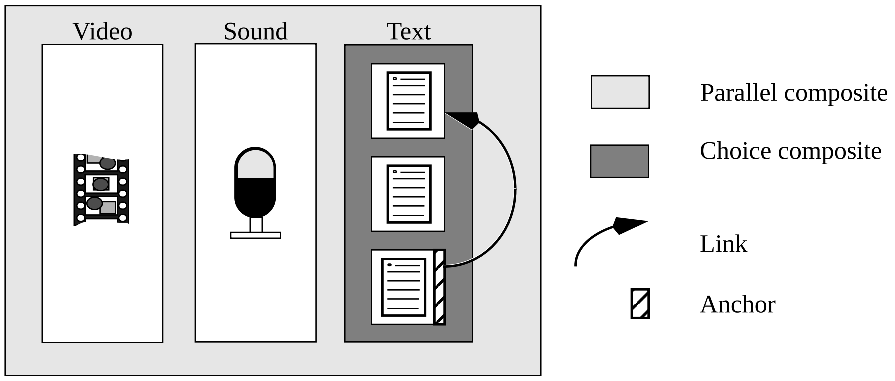

Figure 1: The link between the text atoms may affect the sound and video atoms.

in a declarative way, the AHM introduces the notion of link context [10]. Link contexts make explicit which part of the document stops playing when a link is followed from an anchor and how much of the destination is presented.

For instance, in figure 1, the effect on the sound and video presentation of traversing a hyperlink between two text atoms depends on the link contexts. If the source context is the third text item, and the destination is the first one, the sound and video presentation will continue with no interruption. However, if the source and destination context are defined to be the parallel composite, the presentation of both the sound and video node will be restarted on link traversal. The notion of link context is formalized in section A1.5.

Layout specification The Dexter model provides an adequate way to model structural relationships among the components of a hypermedia document by means of the composite component.

In the Dexter model, all spatial and temporal relationships among components are assumed to be hidden in the presentation specification, arising from the set *PresentSpec*. The presentation specification is the main interface between the storage and the runtime layer. While the internal structure of the *PresentSpec* is considered to be beyond the scope of the model, Dexter makes heavy use of this concept. Each component in the storage layer has a *PresentSpec* to store presentation information local to the component. Additionally, each link-end specifier uses a *PresentSpec*, for instance to store information on how the target should be displayed in the case a link is followed. Note that the link as a whole — being a component itself — also has a *PresentSpec*. To make the situation even more complex, the runtime layer may add an extra presentation specification in order to be able to reflect runtime knowledge in the specification of a component.

Yet, even this large number of presentation specifications proves to be insufficient in some cases [9]. More importantly, however, we think that the ability to express temporal and spatial relationships between components is too important to be omitted from a hypermedia reference model. As the use of dynamic media will increase, comparing the techniques used to express these relationships becomes an essential part of comparing different hypermedia systems. Additionally, it is useful to have commonly accepted abstraction mechanisms and terminology addressing exactly these topics. For example, one of the main objectives for developing the HyTime standard was the need within the SGML-community for standardized methods of (spatio-temporal) alignment. As such, we see common abstractions for spatio-temporal alignment as a requirement for interoperable hypermedia systems.

A1.3 Preliminaries

Before we start with the specification of the Amsterdam Hypermedia Model, we define the Dexter concepts we reuse in the AHM[1]. Additionally, we need to introduce some basic classes which reflect some (relatively small) differences between the AHM and the original Z specification of the Dexter model.

Presentation Specifications We explicitly discriminate between the information describing temporal and spatial relationships from other presentation information (modeled by *PresentSpec* as in Dexter). Following Dexter, we consider the inner structure of a *PresentSpec* beyond the scope of our model. In the following sections, spatio-temporal presentation information is modeled by subclassing *AhmPresentSpec*. All objects which need such a layout specification, need to store other presentation information as well. As a consequence, we include a plain Dexter *presentSpec* in our *AhmPresentSpec* to be able to store style information.

$[\mathit{PresentSpec}]$

AhmPresentSpec

$\mathit{style} : \mathit{PresentSpec}$

Anchoring The AHM extends the Dexter anchor by adding a style specification and semantic attributes. In Dexter, anchor style can be modeled in the link-end *Specifier*, but by locating the style information in the anchor, multiple link-ends

[1]We could have used the object-oriented facilities of Object-Z to reuse these parts of the Dexter specification. To make this appendix a self-contained document, we explicitly copied the required definitions of the Dexter specification into the AHM specification.

can share the same anchor style for the same anchor, while link-ends may still override the style specification provided by the anchor. Adding semantic attributes to anchors allows knowledge-oriented applications to store meta data associated with the anchors, and make anchors subject to querying and other information retrieval processing. We leave the Dexter notions of $AnchorValue$ and $Attribute / Value$ pairs unchanged.

$$[Attribute, Value]$$

$$[AnchorValue]$$

$$\begin{array}{|l}
\hline
\textit{AhmAnchor} \\
\hline
anchorStyle : PresentSpec \\
attributes : Attribute \rightarrowtail\!\!\!\!\!\!+ \; Value \\
anchorValue : AnchorValue \\
\hline
\end{array}$$

Components We introduce the $AhmComponent$ as a base class for the AHM atom, link and composite components described in section A1.6. All three component classes differ from their Dexter counterparts in having anchors of the $Anchor$ type described above.

$$\begin{array}{|l}
\hline
\textit{AhmComponent} \\
\hline
attributes : Attribute \rightarrowtail\!\!\!\!\!\!+ \; Value \\
anchors : \mathrm{seq}\, AhmAnchor \\
\hline
\end{array}$$

Specifications Dexter uses the $ComponentSpec$ to indirectly specify a component. The AHM also applies the advantages of this indirect addressing mechanism to media items, anchors and channels. In this way, one can refer to these objects by means of a database query as is already possible for components in the Dexter model. The specifications arise from the following sets:

$$[ComponentSpec, MediaItemSpec, AnchorSpec, ChannelSpec]$$

Resolver functions are needed to map the specifications to the specified objects. The channel resolver will be introduced in section A1.4.1. We follow the Dexter convention of representing the raw media data by the given set $Atomic$.

$$[Atomic]$$

$$
\begin{array}{|l}
mediaResolver : MediaItemSpec \pfun Atomic \\
compResolver : ComponentSpec \pfun \downarrow AhmComponent \\
anchorResolver : AnchorSpec \pfun \downarrow AhmAnchor
\end{array}
$$

Note that the resolvers no longer return identifiers (such as Dexter's Uid and $AnchorId$). We consider the use of explicit identifiers and accessor functions superfluous since we can use the implicit object identifiers of the Object-Z language.

A1.4 Spatio-Temporal Layout

Spatial layout is modeled differently from temporal layout in the AHM. Spatial layout definitions are defined by $Channel$s. Components can share the same spatial layout by using the same $Channel$. Temporal layout definitions are modeled by synchronization arcs ($SyncArcs$). As noted before, temporal relations also play an important role in document composition.

A1.4.1 Spatial relationships

In the AHM, spatial relationships between components are defined by the use of channels, which are abstract output devices for playing the contents of components. When the document is played, channels are mapped onto physical output devices, so they can be effectively used for resource allocation purposes. Channels may additionally define other (default) presentation attributes for all associated components. For example, an author is able to change the font of all captions in the document, by changing the font of the "caption-channel", instead of changing the presentation specification of all individual caption components.

In the model, spatial constraints are supported by the $Channel$ attribute of the components. Channels are defined by a presentation specification and a resource allocation information field.

$[ResourceSpec]$

Channel

$$
\begin{array}{|l}
parent : ChannelSpec \\
style : PresentSpec \\
resourceSpec : ResourceSpec \\
attribute : Attribute \pfun Value
\end{array}
$$

$$
\begin{array}{|l}
channelResolver : ChannelSpec \pfun Channel
\end{array}
$$

A1.4.2 Temporal relationships

Temporal relationships among the components in the AHM are represented by synchronization arcs. Temporal constraints indicate a preferred delay and allowable deviations. The constraints can be $ADVISORY$, meaning that realization of the constraint at runtime is desirable, but not strictly necessary, or $HARD$, meaning that violating the constraint would be an error. The initial values default to an (advisory) delay of zero.

$$SyncType ::= HARD \mid ADVISORY$$

$TemporalConstraint$

$preferredTime : \mathbb{R}$
$minimimBeforeTime : \mathbb{R}$
$maximumAfterTime : \mathbb{R}$
$type : SyncType$

INIT

$preferredTime = minimimBeforeTime = maximumAfterTime = 0$
$type = ADVISORY$

A synchronization arc is defined by references to the anchors of the arc's source and destination, followed by the temporal constraint between these components. Synchronization arcs are used to denote temporal constraints among descendants of a composite document, and are considered to be part of the composite's presentation specification (see also section A1.6.3). Constraints may be defined between the two intervals associated with the anchors. Note that any of the thirteen possible temporal relations defined by Allen [2] can be described by at most two synchronization arcs (see also chapter 3 of this thesis). Additionally, constraints may be defined on specific points in the document as well. For instance, a synchronization arc may be used to synchronize a video frame with an audio sample. In that case the anchors resolve to an individual frame or sample.

Note that we explicitly do not overload the Dexter link for specifying temporal constraints because hyperlinks are considered primarily to describe semantic relationships (although they can be used for navigation purposes). In contrast, synchronization arcs cannot be used for describing semantic relationships, nor for navigation, but are used for describing temporal presentation information. Both links and synchronization arcs, however, use the same media-independent anchoring mechanism to address their end points.

$SyncArc$

$$
\begin{array}{l}
source : (ComponentSpec \times AnchorSpec) \\
destination : (ComponentSpec \times AnchorSpec) \\
constraint : TemporalConstraint \\
\hline
compResolver(first(source)) \neq compResolver(first(destination))
\end{array}
$$

$\textit{INIT}$

$$
constraint.\textit{INIT}
$$

A1.5 Link Context

The declarative aspects of the concept of link context can be easily formalized by deriving a new class from the Dexter *Specifier*. We need a flag which indicates whether the source context needs to be continued, paused or deactivated:

$$SourceActivation ::= CONTINUE \mid PAUSE \mid DEACTIVATE$$

The destination context can be activated in a playing or a paused state:

$$DestinationActivation ::= PLAY \mid PAUSE$$

We reuse the Dexter specification for modeling *Direction*:

$$Direction ::= FROM \mid TO \mid BIDIRECT \mid NONE$$

The presentation specifier for the link end specifier contains both flags, and inherits from the *AhmPresentSpec*:

$SpecifierPresentSpec$

$AhmPresentSpec$

$$
\begin{array}{l}
srcActivation : SourceActivation \\
dstActivation : DestinationActivation
\end{array}
$$

$\textit{INIT}$

$$
\begin{array}{l}
srcActivation = CONTINUE \\
dstActivation = PLAY
\end{array}
$$

For each link end (i.e. specifier), the context component of the link is specified by an attribute of type *ComponentSpec*. The context component is typically a composite containing the link end component. For example, if the source of a link is an anchor in a subtitle component, the associated context is likely to be the composite containing the subtitle along with the video and sound track component.

LinkSpecifier

$$\begin{array}{l}
\mathit{presentSpec} : \mathit{SpecifierPresentSpec} \\
\mathit{anchor} : \mathit{ComponentSpec} \times \mathit{AnchorSpec} \\
\mathit{context} : \mathit{ComponentSpec} \\
\mathit{direction} : \mathit{Direction} \\
\hline
\mathit{direction} \neq \mathit{NONE}
\end{array}$$

INIT

$$\begin{array}{l}
\mathit{context} = \mathit{first}(\mathit{anchor}) \\
\mathit{presentSpec}.\mathrm{INIT} \\
\mathit{direction} = \mathit{FROM}
\end{array}$$

By default, the specifier is initialized to represent the most simple case, i.e. where the context component equals the component containing the anchor.

The context's role (whether it is a source or destination context) depends on the direction of the associated specifier (FROM or TO respectivly). Actually, the context can act as both source and destination context (if $\mathit{direction} = \mathit{BIDIRECT}$) so in general, the contexts of a link can only be determined at run-time. Note that Dexter's use of NONE has been criticized [6] because of its undefined semantics. In AHM, we disallow a NONE direction.

By making the context itself not a part of the presentation specification, we claim that context adds structural (and indeed semantic) information to a hyperlink and is considered to be more than "just" presentation information. Furthermore, by requiring the context of a link-end to be a (composite) component, contexts themselves are closely related to the document structure. To promote anchor reuse, our notion of context is defined on the link level, and not on the anchor level, as is used in MacWeb [11].

A1.6 Components in the AHM

Keeping the above descriptions of spatio-temporal relationships in mind, we can now formalize the various components of the AHM. We describe the AHM atomic, link and composite component, focusing on the structure of the spatio-temporal information within the components. Additionally, we discuss the difference between the Dexter and AHM components and their specifications.

A1.6.1 Atoms

The AHM atomic component mirrors its Dexter counterpart, but makes its spatio-temporal characteristics explicit. We expect all spatio-temporal arithmetic to be carried out using real numbers (the associated unit used, e.g. seconds, milliseconds, pixels or centimeters, is outside the scope of the model). The duration and layout information of the atomic component is described in its presentation specification, since it provides layout information which needs to be interpreted in the runtime layer.

Presentation Specification of Atoms The AHM atom's temporal characteristics are reflected by its duration (stated by the author or as an intrinsic property of the media item itself). Its spatial layout is defined by the associated channel. An atom may modify its channel's layout definition by defining an (smaller) extent within the extent defined by the channel. This is reflected by the extent and position attributes. Style information is modeled by the inherited style attributed.

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AtomPresentSpec}} \\
\hline
\mathit{AhmPresentSpec} \\
\begin{array}{|l}
\hline
\mathit{duration} : \mathbb{R} \\
\mathit{channel} : \mathit{ChannelSpec} \\
\mathit{position} : \mathbb{R} \times \mathbb{R} \\
\mathit{extent} : \mathbb{R} \times \mathbb{R} \\
\hline
\end{array} \\
\hline
\end{array}$$

Anchoring in Atoms The concept of anchoring for atomic components needs to be extended to include a duration. For continuous media items, the media dependent $\mathit{AnchorValue}$ can be expected to define the duration of an anchor (for example, by defining a range of frames for a video fragment). But for static media such as text, we need to define the interval in which the anchor is active. Additionally, we have extended the Dexter anchor with attributes to store semantic information (these can be used for information retrieval or knowledge representation purposes) and an anchor style presentation specification. These additions are inherited from the $\mathit{AhnAnchor}$ base class:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{AtomicAnchor}} \\
\hline
\mathit{AhmAnchor} \\
\begin{array}{|l}
\hline
\mathit{startTime} : \mathbb{R} \\
\mathit{duration} : \mathbb{R} \\
\hline
\end{array} \\
\hline
\end{array}$$

The Atomic Component The AHM atomic component contains the presentation specification described above, and a list of anchors. The actual media content is referred to by a system dependent media item reference (this can be a filename, URL or database query) and a media dependent anchor value denoting which part of the media item is used. In this way, the model is independent of the granularity of the server providing the media items (e.g. an author does not need to create a new image file if only a part of that image needs to be included in the presentation).

$AhmAtom$

$\upharpoonright(Init, attributes)$

$AhmComponent$

$presentSpec : AtomPresentSpec$

$anchors : \operatorname{seq} AtomicAnchor$

$content : MediaItemSpec \times AnchorValue$

A1.6.2 Links

The link is — following the Dexter model — a component with a sequence of link-end specifiers derived from *Specifier*. It can be used to define links of arbitrary arity. Because the AHM does not associate spatio-temporal information with the link component itself, there is no need to specify a new presentation specification class for link components. Other style information, for instance to display the link in a link browser, is specified using a Dexter *PresentSpec*. Being a component, links can be end points of other links, so a link needs anchors just as the other components do. The link component inherits its attributes and anchors from AhmComponent. Its class schema adds a style (e.g. to display the link in a link browser), a duration (of a possible transition from source to destination), a position (the position of the destination relative to the source anchor or mouse click) and a list of specifiers. Note that the link contains only one duration and position attribute, which is not sufficient for links containing multiple destinations. This problem needs to be solved in a next version of the model.

AhmLink

$\upharpoonright(Init, attributes, anchors)$

$AhmComponent$

$style : PresentSpec$

$duration : \mathbb{R}$

$position : \mathbb{R} \times \mathbb{R}$

$specifiers : \mathrm{seq}\, LinkSpecifier$

$\# specifiers \geq 2$

Constraining the number of end point specifiers to be at least two might prove to be too restrictive, especially in an open collaborative work environment [6].

A1.6.3 Composites

The AHM discriminates between two types of composition: temporal and atemporal. All children of a temporal composite share the same time line and need to be synchronized using synchronization arcs. In contrast, the children of an atemporal composite are scheduled on different time lines and cannot have any temporal relationships.

Anchoring in Composites The AHM addresses the underspecification of anchors for composite components [6] by defining the anchor value to be a list of references to anchors defined by the descendants of the composite. See figure 2. This can be used to group anchors by building a hierarchical structure of composite anchors. Semantic attributes and style information can be attached to each anchor.

CompositeAnchor

$AhmAnchor$

$anchorValue : \mathrm{seq}(ComponentSpec \times AnchorSpec)$

Presentation Specification of Composites Because of their different temporal properties, the two composite types need different presentation specifications. The temporal presentation specification contains a list of synchronization arcs defining the temporal relations among the children of the composite. By specifying a duration a user can override the intrinsic duration of the composite (the playing environment may scale or clip the children in order to achieve this):

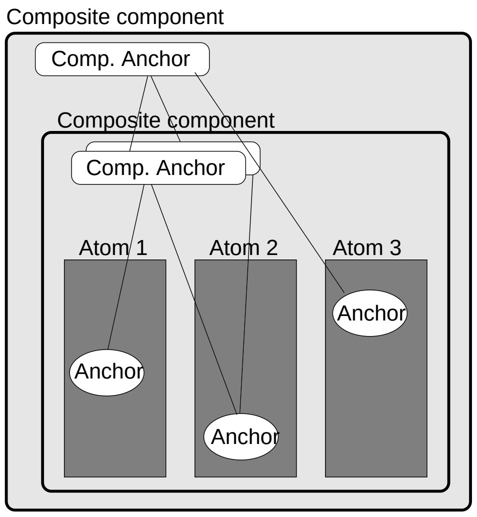

Figure 2: Composite anchoring hierarchy

TemporalPresentSpec

AhmPresentSpec

$duration : \mathbb{R}$
$syncArcs : \text{seq}\, SyncArc$

Since there are no temporal relationships between the children of an atemporal composite, the atemporal presentation specification does not contain any synchronization arcs. Instead, it specifies the initial activation state of each of the children. This state can be play, pause or inactive. Inactive children can only made active as a result of link traversal:

$$ActivationState ::= PLAY \mid PAUSE \mid INACTIVE$$

AtemporalPresentSpec

AhmPresentSpec

$initialStateOfChildren : \text{seq}\, ActivationState$

The Composite Component The *AhmComposite* serves as an abstract base class for the two composites:

$AhmComposite$

$\upharpoonright(Init, attributes)$

$AhmComponent$

$anchors : \mathrm{seq}\ CompositeAnchor$
$children : \mathrm{seq}\ ComponentSpec$
Δ
$descendants : \mathbb{P}\downarrow AhmComponent$

$$\begin{aligned}&\forall a : \mathrm{ran}\ anchors \bullet \forall avalue : \mathrm{ran}\ a.anchorValue \bullet \exists d : descendants \bullet\\&\quad compResolver(first(avalue)) = d \land\\&\quad anchorResolver(second(avalue)) \in \mathrm{ran}\ d.anchors\end{aligned}$$

Note that the state invariant requires that the anchors of the composite refer to existing anchors of the composite's descendants (the formal definition of *descendants* is left out for reasons of brevity) The temporal composite extends the *AhmComposite* with this presentation specification:

$TemporalComposite$

$AhmComposite$

$presentSpec : TemporalPresentSpec$

The specification of the atemporal composite mirrors the definition of its temporal counterpart. It requires the specification of an initial state for all its children:

$AtemporalComposite$

$AhmComposite$

$presentSpec : AtemporalPresentSpec$

$\#children = \#presentSpec.initialStateOfChildren$

A1.7 Hypermedia

Finally we define the *Hypermedia* class. The composition structure in AHM, as in Dexter, specifies a directed, acyclic graph. The AHM differs, however, by requiring that the graph should have a unique root element. If a graph has multiple potential root elements, these can always be combined together into a single root element using atemporal composition.

The first state invariant follows the constraints of the Dexter hypertext formalization in order to ensure accessibility of the components. It states that every component should be accessible by the external component resolver (ran $compResolver = components$). Furthermore, every component needs to be a descendant of the unique $root$ component ($components = root.descendants$).

The Dexter hypertext specification contains several consistency constraints which can only be ensured in a "closed" hypermedia system. In particular, Dexter requires all links to refer to existing components and anchors, *within* the system. As a result, deleting a component involves deleting all links resolving to the deleted component. Since these constraints can never be ensured in a open environment (such as the WWW), they have been left out in the following specification.

Hypermedia

$root : AhmComposite$
Δ
$components : \mathbb{F} \downarrow AhmComponent$

$components = \varnothing \vee components = root.descendants$
$\mathrm{ran}\ compResolver = components$

INIT

$components = \varnothing$

A1.8 Conclusion

The abstractions concerning the mechanisms used to describe spatio-temporal relationships and the context of a hyperlink are important features of a hypermedia reference model. The Amsterdam hypermedia model provides extensions to the Dexter model that address these issues. Previous descriptions of the model gave only informal descriptions of these abstractions, and so we have expressed them here as part of a formal description of the model using the specification language Object-Z. In the process of formalizing the model, we have obtained the refined version of that presented in [9], as presented in this thesis, where a number of flaws have been fixed.

In order to formalize the operational behavior of the additions to the storage layer as presented in this appendix, the formalization of the Dexter runtime layer needs to be extended The specification of the Dexter runtime layer, as given in in [7] focuses on the mechanics of link traversal. A specification of the AHM runtime layer should include the effect of context upon link traversal, and model the temporal interdependencies between the possible state transitions within the model.

References

[1] ACM. *Proceedings of ACM Hypertext '93 (Seattle)*, November 1993.

[2] James F. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–844, November 1983.

[3] Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z Specification Language: Version 1. Technical Report 91-1, Software Verification Research Centre, Department of Computer Science, The University of Queensland, Australia, April 1991. The most complete (and currently the standard) reference on Object-Z. It has been reprinted by ISO JTC1 WG7 as document number 372.

[4] Roger Duke and Gordon Rose. Modelling Object Identity. Technical Report 92-11, Software Verification Research Centre, Department of Computer Science, The University of Queensland, Australia, 1992.

[5] Roger Duke, Gordon Rose, and Graeme Smith. Object-Z: a Specification Language Advocated for the Description of Standards. Technical Report 94-45, Software Verification Research Centre, Department of Computer Science, The University of Queensland, Australia, December 1994.

[6] K. Grønbæck and R. Trigg. Design Issues for a Dexter-Based Hypermedia System. *Communications of the ACM*, 37(2):40–49, February 1994.

[7] F. Halasz and M. Schwarz. The Dexter Hypertext Reference Model. In *NIST Hypertext Standardization Workshop*, pages 95–133, January 1990.

[8] F. Halasz and M. Schwarz. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, February 1994. Edited by K. Grønbæck and R. Trigg.

[9] 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.

[10] L. Hardman, D.C.A. Bulterman, and G. van Rossum. Links in hypermedia: the requirement for context. In *Proceedings of ACM Hypertext '93 (Seattle)* [1], pages 183–191.

[11] Jocelyne Nanard and Marc Nanard. Should Anchors Be Typed Too — An Experiment with MacWeb. In *Proceedings of ACM Hypertext '93 (Seattle)* [1], pages 51–62.