CSA402

Lecture 14

The Dexter Hypertext Reference Model

Reference

Halasz, F. and Schwartz, M. 1994. 'The Dexter Hypertext Reference Model', in *Communications of the ACM*, 37(2), February, 1994, 30-39.

Overview

- A Reference Model for hypertext that:

  can be used to compare existing hypertext systems;

  that can be used to design new hypertext systems;

  that can be used to devlop interchange and interoperability standards

- Not the cartoon character

Background

- Between 1988 and 1990, a number of hypertext technology leaders met to *define* hypertext and hypertext standards, and the DHRM was developed

- DHRM is one of the most popular hypertext reference models because it is based on graph theory, whereas others are based on set theory, petri-nets, etc.

- DHRM is a *reference model* and not an *implementation* of a reference model, although it has been implemented

- We will not be talking about the *implementation*

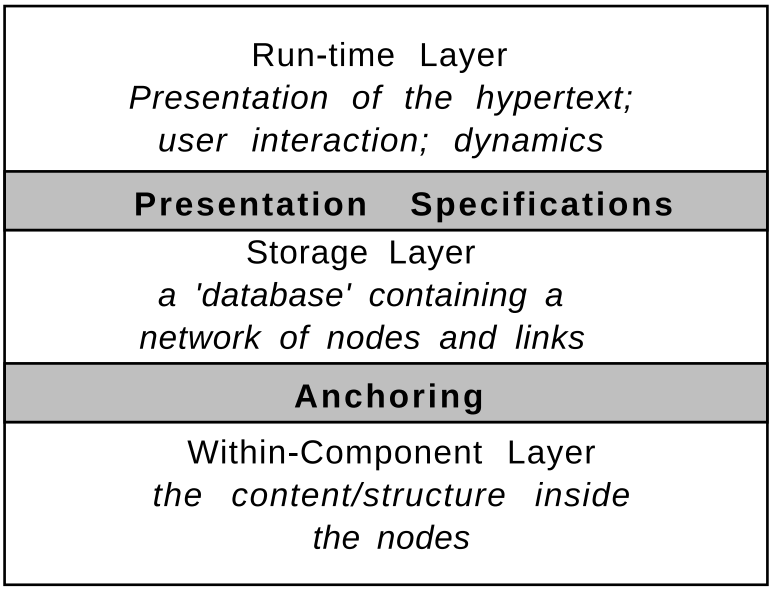
Overview of the Model

- DHRM divides a hypertext system into three layers

  Run-time Layer

  Storage Layer

  Within-component Layer



Run-time Layer
Presentation of the hypertext; user interaction; dynamics

Presentation Specifications

Storage Layer
a 'database' containing a network of nodes and links

Anchoring

Within-Component Layer
the content/structure inside the nodes

- DHRM focuses mainly on the Storage Layer, which models the basic node/link network structure of hypertext

- The Storage Layer does not concern itself with what is contained in the node - that is the function of the within-component layer

- Moreover, the within-component layer does not attempt to provide a model for the different types of data that can be contained within components - it is assumed that other reference models will do this and that those reference models will be used in conjunction with DHRM to capture the entirety of the hypertext

- However, DHRM does specify the interface between the Storage Layer and the within-component content and structure, to provide an addressing mechanism

- In DHRM, this is called *anchoring*, and allows links to have source and destination anchors.

- Links can be span-to-span, as well as document-to-document, and their combinations (e.g., document-to-span)

- The Run-time Layer provides tools to access, view and manipulate the hypertext network

- Once again, the tools which could be included in the run-time layer are too diverse to be captured by a generic model, so DHRM describes only a 'bare-bones' model

- The run-time layer captures the essentials of the dynamic, interactional *aspects* of hypertext systems, without covering the *details* of the user interaction with the hypertext

Simple Storage Layer Model

- The Storage Layer describes a hypertext as a finite set of components together with two functions: *resolver* and *accessor*
- The resolver and accessor functions are jointly responsible for retrieving components
- A component can be *atomic*, a *link*, or *composite* (composed of many non-self referential components)
- A component has a globally unique identity (UID), not just within a specific hypertext implementation, but across the entire universe of discourse
- The accessor function is responsible for accessing a component given its UID
- The resolver function is responsible for determining the UID(s) of satisfying components given some other method of reference

- E.g., difference between IP and DNS
- E.g., difference between URL and search engine
- E.g., link to the component which contains the statement "The King of France wears a wig"
- The resolver function takes a *component specification* and returns one or more UIDs which can be fed to to accessor function
- A component specification can result in 0, 1 or more UIDs, but for every component there is at least 1 specification which will resolve to the UID for that component

Links in the Storage Layer

- To implement span-to-span linking, more than just the UIDs of the components are required

- It is also necessary to identify substructures within the component

- Needs to be independent of the actual data type contained in the component

- The addressing technique used in DHRM is *anchoring*

- An anchor contains two parts:

  anchor id
  anchor value

- Anchor id identifies the anchor

- Anchor value identifies some location, region, or substructure within the component

- The anchor value is interpretable only by the application(s) responsible for handling the content/structure of the component

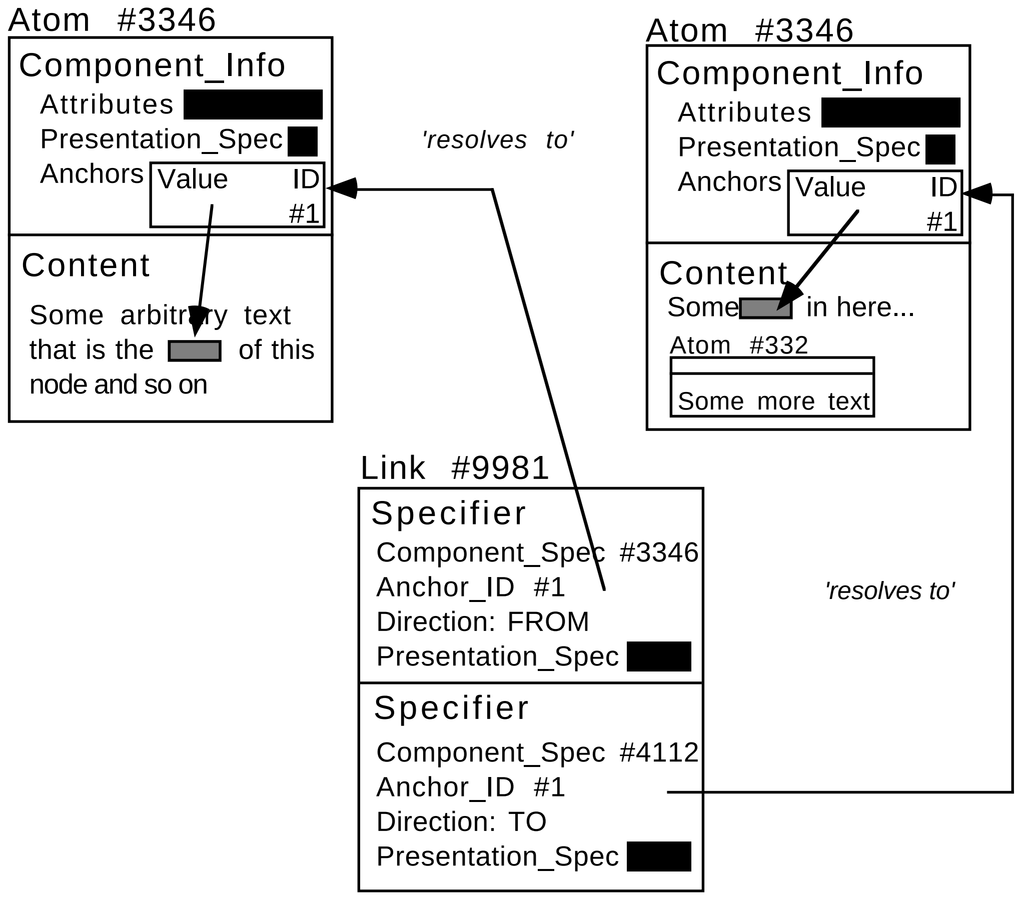
- The mechanism of the anchor id can be combined with the component specification mechanism to provide a way of specifying the end-points of a link

- In DHRM, this is called a *specifier*

- Apart from the component specification and the anchor id, a specifier also contains a *direction* and a *presentation specification*

- A link's specifier specifies a component and an anchor point within the component which acts as the end-point of the link

- The link's direction specifies whether the anchor point is the link source (FROM), destination (TO), both source and destination (BIDIRECT), or NONE



Atom #3346

Component_Info

Attributes

Presentation_Spec

Anchors

Value ID #1

Content

Some arbitrary text that is the of this node and so on

'resolves to'

Atom #3346

Component_Info

Attributes

Presentation_Spec

Anchors

Value ID #1

Content

Some in here...

Atom #332

Some more text

Link #9981

Specifier

Component_Spec #3346

Anchor_ID #1

Direction: FROM

Presentation_Spec

Specifier

Component_Spec #4112

Anchor_ID #1

Direction: TO

Presentation_Spec

'resolves to'

- The presentation specification forms part of the interface between the storage and run-time layers - discussed later

- Links can have arbitrary arity (min. 2), with at least one having a direction of TO or BIDIRECT

- A component was previously described as atomic, link, or composite

- These are actually *base components*

- A component is a base component together with *component information*

- The component information describes the properties of the component, apart from its content

- Component information contains:

  Sequence of anchors
  Presentation specification
  A set of arbitrary attribute/value pairs

- The attribute/value pairs can be used to associate keywords and type information with the component

- The storage layer also defines a set of operations that can be used to access/modify the hypertext

- Examples:

  CreateComponent
  DeleteComponent
  ModifyComponent
  LinksToAnchor
  LinksTo

Simple Run-Time Layer Model

- The most important function of the run-time layer is the presentation of a component to the user

- This is called the *instantiation* of a component

- When a component is requested, a 'copy' is cached in the instantiation. The cached copy can be viewed and/or edited, and the altered cache is 'written' back to the storage layer

- Each instatiation is assigned a unique within-session identfier (IID)

- The instatiation of a component also results in the instantiation of its anchors, called *link markers*, which are a visible manifestation of the anchors in the displayed document

- At any given moment, a user can be viewing/editing any number of instatiations

- The run-time layer keeps track of the mapping between components and their instantiations through an entity called a *session*

- The interaction cycle of a session is

  Open session: user initialises interaction with hypertext
  Present component: user creates an instantiation
  Realise edits: user modifies the component based on edits to the instantiation
  Unpresent component: user destroys the instantion
  Close session: user terminates interaction with hypertext

- The session entity contains:

  the hypertext being accessed
  a mapping from IIDs to components
  a history
  a run-time resolver function
  an instatiation function
  a realiser function

- The run-time resolver function is the run-time version of the storage layer's resolver function

- It maps specifiers into component UIDs

- The run-time resolver function (RTRF) is a superset of the storage layer's equivalent function (STRF)

- RTRF can refer to information in the run-time session, to which the storage layer does not have access

- E.g., a reference to "the most recently accessed component named 'xyz'"

The *instantiator* function

- The core of the run-time model
- Input to the instantiator is a component UID and a presentation specification
- The instantiator returns an instantiation of the component as part of the session
- The presentation specification specifies *how* the component is to be presented by the system during this instantiation
- The component also has a presentation specification as part of its information
- This represents the component's own notion about how it is to be presented
- The instantiator function must decide how to resolve differences between the two presentation specifications
- The act of following a link (*follow link*) calls the *present component* operator, which in turn calls the instantiator.

The *realiser* function

- The 'inverse' of the instantiator function
- Takes an instantiation and creates a new component in the storage layer

Conformance with DHRM

- DHRM describes a significantly more powerful hypertext system than existed beack in 1990, and indeed, which exist today

- Main differences between DHRM and typical hypertext implementations are:

  multiway links
  composite components
  dangling links

- DHRM is revised into sets of models, including a minimal model, and optional mechanisms within more complex models

Can the Dexter Hypertext Reference Model be used to describe an Adaptive Hypertext System?