Toward a Dexter-based model for open hypermedia: Unifying embedded references and link objects

Kaj Grønbæk
Computer Science Deptartment
Aarhus University
Ny Munkegade, Bldg. 540
DK 8000 Aarhus C
Denmark

Randall H. Trigg
Xerox PARC
3333 Coyote Hill Rd.
Palo Alto, CA 94304
USA

ABSTRACT
The Dexter Hypertext Reference model is well suited to modelling anchor-based hypermedia systems and static hypermedia structures. But it is less clear that Dexter is adequate for systems whose linking is based on embedded references like the World Wide Web (WWW), nor for modelling the dynamic aspects of contemporary hypermedia systems like DHM and Microcosm. This paper proposes a new Dexter-based extensible object-oriented model designed to cover a broader spectrum of the features of contemporary hypermedia systems. The model introduces two new concepts, LocationSpecifiers and ReferenceSpecifiers, which let us model links as references embedded in documents as well as links as objects in separate databases. This suggests the idea of new systems that could support both styles as one step toward integrating global networked information sources with application-bridging systems on local hosts. In addition, our model is better equipped to handle dynamic hypermedia structures. As an example, a model of Microcosm's Generic Link is given which extends that important concept in useful ways.

Keywords: open hypermedia, link objects, embedded links, Dexter hypertext reference model, dynamic hypermedia, generic links

1. INTRODUCTION

The goal of open, integratable hypermedia systems is as pertinent today as it was when Norm Meyrowitz issued his "call to arms" at the first hypertext conference (Meyrowitz, 1989). Over the years since then, the investment by researchers and product designers in non-monolithic integrating approaches to hypermedia has steadily increased. In our minds, one of the landmark events of this period was the formulation and presentation of the Dexter Hypertext Reference Model (Halasz & Schwartz, 1990). Dexter explicitly recognized the importance of existing "content" (and to a lesser degree "content editors") in the work contexts that hypermedia was hoping to support. Indeed, the three layers introduced by Dexter included, in addition to *Storage* and *Runtime*, a *Within-Component* layer meant to model the technological/-material environment existing prior to and to some degree independent of the hypermedia substrate.

Dexter captured the understanding that successful application of hypermedia sometimes requires connecting new hypermedia structures being offered with existing content and software (as opposed to expecting such software to be part of the hypermedia facility). Toward that end, Dexter proposed the key concepts of *anchor* and *presentation spec*. Anchors model the joining of hypermedia structures (like links) with existing content, while pspecs model the storage of bits of runtime behavior (often UI-related) in hypermedia structures.

Since 1991 we have been involved in developing open hypermedia systems based on the ideas of the Dexter model. We developed an object-oriented application framework, called the Devise Hypermedia framework (Grønbæk & Trigg, 1994), implementing and extending the generic Dexter concepts. One of our primary contributions was to make link services available to third party applications.[1] In addition, we

Hypertext '96, Washington DC USA
© 1996 ACM 0-89791-778-2/96/03..$3.50

[1] In this, we were inspired by Amy Pearl's (Pearl, 1989) Sun Link Service, and the Microcosm work at Southhampton University (Hall, Davis, Pickering, & Hutchings, 1993).

offered a richer set of composites and new views of dangling links and link directionality.

Based on this development experience, we have begun to reformulate the model behind our framework in general implementation-independent terms. It is worth emphasizing, though, that a formal articulation of a framework (like the Dexter-based one proposed here) is just one part of an open systems effort, along with, for example, providing concrete support for inter-operability and customization. Our reformulation is meant to delineate a hypermedia design space capturing what we deem to be essential to successful hypermedia. We are informed in this effort by: (1) the document-related work of real people both with and without hypermedia technology; (2) our own experience building useful systems; and (3) other successful hypermedia projects and the issues they have raised. See, for example, (Malcolm, Poltrock, & Schuler, 1991), (Grønbæk, Kyng, & Mogensen, 1993), (Hall, et al., 1993), (Kacmar & Leggett, 1991).

Our goals in this effort overlap with but are not identical to the original Dexter goals. On the one hand, we share the goal of *coverage*, the ability of the model to capture well known actual systems, whether products or research prototypes. Like Dexter, the model should also be *general* and *simple*. That is, it should elegantly express those concepts that are most crucial to what hypermedia brings to the table. It should be *leading* in the sense that it suggests new directions for hypermedia development. And finally it should be *concrete*, offering a useful starting point for designers of hypermedia systems (something Dexter didn't particularly emphasize). Here we want the model to be valuable not just for those designing systems from scratch, but also for those supporting integration, say, by employing hypermedia links to connect diverse documents stored and/or edited in sometimes "un-touchable" third party applications (Haan, Kahn, Riley, Coombs, & Meyrowitz, 1992), (Grønbæk & Trigg, 1994), (Davis, Hall, Heath, Hill, & Wilkins, 1992).

At a conceptual level, our approach can be compared to that of the HyTime SGML-based hypermedia standard (DeRose & Durand, 1994). While we have been working on data models for hypermedia systems, HyTime is best thought of as a representation and interchange language for hypermedia documents. Nonetheless, parts of the HyTime standard cover similar conceptual ground. We claim that our object oriented model is simpler and more directly applicable to the design of hypermedia systems. See the appendix at the end of this paper for a brief comparison of the two approaches.

It is important to emphasize that our extended Dexter model does not address hypermedia *application* design ala for example, RMM (Isakowitz, Stohr, & Balasubramanian, 1995) or HDM (Garzotto, Paolini, & Schwabe, 1993). Rather it provides a data model for hypermedia *system* design, or as is so often the case in practice, hypermedia-based system integration. However, we believe that our model is compatible with these application design models. For example, HDM's three link categories ("perspective," "structural," and "application") are mappable onto our extended version of Dexter's link components.

Among other hypermedia system data model approaches, we share the goals of HB1 (Schnase, Leggett, Hicks, & Szabo, 1993) to support integrating third party applications. However, the two models are based on very different approaches, HB1 being process oriented (links and anchors are modeled as operating system-like processes), whereas we take an explicitly object-oriented approach. In our model, the process-like behavior of links and anchors arises primarily from the invocations of computed links (or as we will see, of computed locSpecs).

In the remainder of this paper, we present our extended version of the Dexter model, pointing out changes both substantive (primarily architectural) and terminological. We include detailed justifications of the two new concepts we are proposing, location specifiers and reference specifiers. We hope it will be clear that our model is an extension of Dexter, that is, that we can still model the systems addresseded by Dexter. Moreover, we demonstrate how we improve on Dexter by offering more intuitive models of systems like HyperCard and their embedded "go to" style links. Finally, we'll argue that our framework is general enough to cover three features of post-Dexter hypermedia systems not addressed by the original Dexter effort:

- representations of links (or more precisely of "pointers") like WWW URL's which can exist outside the hypermedia;
- dynamic aspects of hypermedia including various forms of computed links (e.g. WWW URLs that point at programs which construct destination documents on the fly); and
- open, integrating aspects like Microcosm's "generic" links.

Like any good model, ours also suggests new approaches. In particular, we propose combining features of embedded link systems like WWW with "hands off" linking ala Microcosm and DHM.

2. THE EXTENDED DEXTER MODEL

This section presents an object-oriented description of our extended Dexter-based model for hypermedia. Our focus here is on the Storage Layer together with one key runtime topic, the scheme for link traversal. We explicate the new model from the "inside out." That is, we start with two new basic concepts, LocSpec and

RefSpec, and from these reconstruct classic Dexter notions like anchor, component, link and composite. (We assume the reader has at least a passing familiarity with these Dexter concepts.) Section 3 argues for the benefits of this new conceptualization.

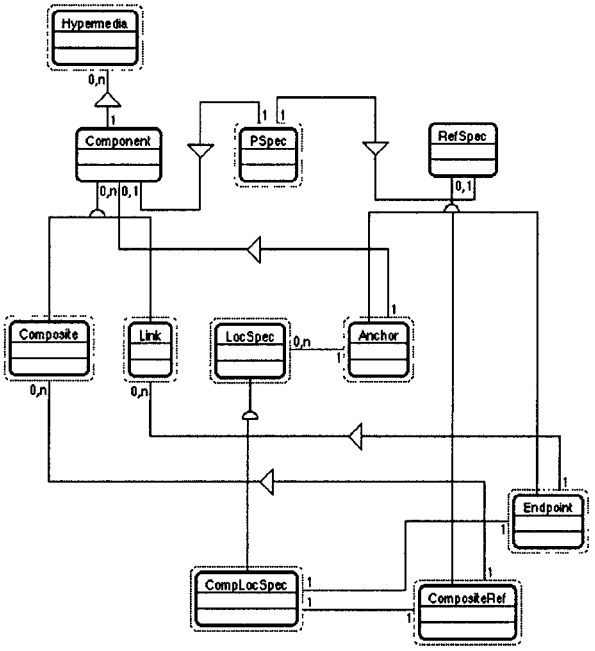

Figure 1: Extended Dexter model in object-oriented notation

Our extended Dexter data (Storage) model is depicted in object-oriented design notation in Figure 1. Lines with triangles denote aggregation relations. For example, a Link object is an aggregation of zero or many Endpoints implying that "dangling" links with zero or one endpoint are legal as are links with many endpoints. Lines with half circles denote inheritance relations. For example, Link and Composite are both specializations of the general Component class.[2] Lines with only numbers at the ends denote associations, e.g. an Anchor object is associated with exactly one LocationSpecifier (LocSpec) whereas a LocSpec can both be free floating (0-association) and associated to many anchors (n-association). Details on the model are given in the following sections.

2.1 Location specifiers

Basic to any hypermedia system is the ability to specify a location in a body of electronic material. We model this basic capability with an entity called *LocationSpecifier* (LocSpec). Present in link endpoints and composite references, it identifies a component or some location (e.g. a component anchor) within a component's contents. LocSpecs also appear in component anchors where they identify (linkable) locations within the parent component's contents. Finally, locSpecs can exist outside the hypermedia altogether (usually by virtue of being textually encoded). Section 3 motivates general locSpecs as well as this "free floating" variety.

LocationSpecifier:

Attributes:

- Object ID (e.g. Anchor ID, a HTML defined name, Persistent Selection ID, etc.)
- Within-component structure descriptor (e.g. position, span, video frame, chapter, paragraph)
- Within-component computation specification (a search criterion like a text string or format description, or an executable script)

The simplest kind of LocationSpecifier is used in component anchors, where legal locations are restricted to those within the anchor's parent component. Such locSpecs specify locations in any of three ways: by absolute (or named) identification, by specifications of within-component structure (e.g., a chapter in an SGML-tagged document), and by specifying a search criterion or script to be applied over the component's contents. These are implicitly 'and'ed together, that is, those of the three that are specified must all be true of any candidate component or its contents. Collectively, we call these three *content locator* attributes.[3]

ComponentLocationSpecifier CLASS LocationSpecifier:

Attributes:

- Component ID (unique within a hypermedia or across the world, e.g. a URL)
- Cross-component structure descriptor (i.e. hypermedia structure, e.g. enclosing composites)
- Cross-component computation specification (a query over component attributes, e.g. creation date; or an executable script)

The class ComponentLocationSpecifier (CompLoc-Spec) is a specialized version of locSpec that identifies locations across a hypermedia; in such cases, the component as well as a location in its contents needs

[2]The single border around the Component and RefSpec boxes indicates that they represent "abstract" classes, i.e. instances may only be created from their subclasses.

[3]Here and in what follows, the descriptions of object classes list Attributes (slots) and Procedures (methods). We understand these to be implicitly extensible. That is, that any class may be extended with new attributes and/or procedures.

to be specified. Specializing the basic LocSpec class, CompLocSpec adds three ways of specifying components analogous to the three means of specifying within-component locations: component naming, positions in cross-component structures like composites, or computations or queries over component attributes. We call these the *component locator* attributes.[4]

The LocSpec class replaces Dexter's anchor value, while the CompLocSpec class subsumes Dexter's Anchor ID and the ComponentSpec of Dexter's Specifiers.

Most common cases of referencing in hypermedia are simple instances of these classes. For example, the compLocSpec for a typical link endpoint or composite reference would have a fixed component ID (as component locator) and anchor id (as content locator). For a more complicated example consider a computed link endpoint that locates occurrences of the string "DAIMI" in components within the "DEVISE Proposals" composite whose dates are within 1994. The compLocSpec might look like:

INSTANCE of ComponentLocationSpecifier:

Attributes:

- Object ID: None
- Within-component structure descriptor: None
- Within-component computation spec: Query with text string: "DAIMI"
- Component ID: None
- Cross-component structure descriptor: ID of "DEVISE Proposals" composite
- Cross-component computation spec: Search criterion: Creation date = 1/94 - 12/94

That same location specifier could act as a Microcosm "generic link." That is, if the user selects a string of text containing "DAIMI" in any component that meets the Component locator criteria, then the other endpoints of the link are opened. For more on how we model "generic links," see Section 5.

[4]A simple matching policy is to require that all of the component locator and content locator attributes (if specified) resolve for a locSpec to match a given location. Unfortunately, this policy can pose problems for the handling of dangling links: if the objectID named in a locSpec is unresolvable, the user might still want the locSpec to resolve to the enclosing component. Thus we may need to allow the locSpec as a whole to match even when its content locator part cannot be resolved. (For a further discussion of dangling links, see Grønbæk and Trigg, 1994.)

Our locSpec can be used to model the WWW's URL addressing mechanism. URLs typically pick out a fixed web page by encoding a server protocol, host name, and pathname (ComponentID). They can also locate particular entry points if those have been predefined in the page's html definition (ObjectID). Finally, they can invoke a program on a server and pass it textstring arguments (Computation spec). Missing from the current URL paradigm is a means of explicitly representing structural filters for searches. (This is due in part to the inability of the WWW to represent explicitly structures larger than a page.)

2.2 ReferenceSpecifiers

As mentioned above, locSpecs can exist independent of the hypermedia. When appearing in the components of a hypermedia, however, they are packaged as RefSpecs (corresponding to Specifiers in Dexter).[5] Every refSpec is contained in some component in which it represents either an anchor, a link endpoint, or a composite reference.[6] Thus refSpecs include, in addition to a locSpec, a *Parent ID* identifying the refSpec's parent component (in some implementations, this is simply a pointer to that component). RefSpecs also include a presentationSpec attribute, which will be extended (i.e. specialized) in the anchor, endpoint, and compositeRef subclasses.

ReferenceSpecifier:

Attributes:

- ParentId (Identifies the link or composite that this RefSpec belongs to)
- RefSpec ID (Unique within enclosing component)
- LocationSpecifier
- PresentationSpecifier (determines how the refSpec is to be displayed)

Procedures:

- EqualLocSpec
- RefsTo

The EqualLocSpec operation accepts another locSpec as argument and determines whether it matches the

[5]Implementations of the framework ought to offer multiple ways of embedding the locSpec information. For example, a designer might choose to "inline" the locSpec information in the refSpec to save the overhead of an extra object reference. On the other hand, if a complex locSpec is to be shared among several refSpecs, a pointer to a locSpec object might be more efficacious.

[6]Note that the refSpec for a transient anchor (Section 2.5) might be short-lived, but it still must have a parent component.

locSpec belonging to this refSpec. This operation is used during link following (Section 2.8). The RefsTo operation returns the set of all refSpecs in the hypermedia that resolve to this refSpec. This could be used, for example, to compute the collection of links and composites that point at a given anchor.

In this model, we require that each refSpec live in a single parent component. However, there is no such requirement on locSpecs (i.e. locSpecs do not include a parentID). Thus multiple refSpecs can share the same locSpec, enabling, for example, reuse of locSpecs that contain complex search queries.

RefSpecs form the basis for the concepts of *anchor*, *endpoint*, and *compositeRef*. Anchors can be associated with any component, while endpoints belong to links and compositeRefs to composite components. The characteristic behavior of hypermedia, following links and opening composites, depends on the ability to match component anchors with endpoints and compositeRefs (see Section 2.8).

2.3 Anchors

RefSpecs that live in components and identify locations within those components are called *anchors*. As we have seen, an anchor's location specifier includes one or more of the following attributes: an ObjectID, e.g. the id of an emacs "mark" or Microsoft Word "bookmark"; a representation of within-component structure, e.g. position, span, video frame, chapter, paragraph; and/or a computation specification (say, a query text string, "yellow rectangle").

Anchor CLASS ReferenceSpecifier

Attributes:

- PresentationSpecifier (determines how the anchor is highlighted in the component contents)

Sometimes redundancy among the three kinds of locSpec content locators is valuable. For example, an unmarked anchor[7] which would otherwise not have a structure descriptor (rather, say, a text string to search for), might nonetheless save a position in the structure descriptor as a means of short-cutting the search through long documents. Likewise, "specific" anchors (ala Microcosm) which would otherwise only have a positional structure descriptor, might save textual "context" for the link as a Computation specification query in order to test for broken links.

Transient anchors are constructed at run-time, usually based on a selection in some component instantiation not associated with an existing anchor, and typically exist for the duration of a followLink operation. Nonetheless, like any refSpec, they belong to some particular component. (See Section 4.2 for a further discussion of the issues surrounding transient anchors.)

The anchor's PSpec attribute (inherited from refSpec) dictates how, whether, and when the anchor should be displayed (often involving some form of highlighting).

2.4 Endpoints and CompositeRefs

Endpoint CLASS ReferenceSpecifier

Attributes:

- Type (One of: *SOURCE, DESTINATION, BOTH.*)
- PresentationSpecifier (determines how the endpoint is displayed in a graphical link viewer)

Procedures:

- SetType
- GetType

Endpoint is a specialization of RefSpec used in link components. Endpoint includes a type attribute whose values (so far) include: *SOURCE, DESTINATION*, and *BOTH*. Like Dexter's "link directionality" constants, these determine how the endpoint will behave when following links.[8] SOURCE corresponds to Dexter's FROM value, and DESTINATION to Dexter's TO value. BOTH means that the endpoint behaves as both a SOURCE and a DESTINATION at the same time. The Endpoint concept subsumes Dexter's Specifier.[9]

---

[7]Previously, we have discussed the difference between *marked* and *unmarked* anchors (Grønbæk & Trigg, 1994). Marked anchors are associated with particular objects inserted in the contents of the document (e.g. buttons in Microcosm, or NAME tags in html) and thus are modelled by the ObjectID attribute of the refSpec's locSpec. Unmarked anchors require searching the contents of the component (say, by keyword) and are modelled by the locSpec's Computation specification attribute.

[8]Endpoint types are further described in Section 3.1, including the rationale for dropping Dexter's fourth type of "link directionality", NONE. Other endpoint types might, however, be useful. For example, a HIDDEN type might denote "blind" link endpoints that aren't presented during follow operations, but are fully inspectable as a part of the link. Assigning HIDDEN to an endpoint would enable temporary hiding of endpoints in a complex link relation.

[9]As we have mentioned, the attributes of any of these classes can be extended. For example, Endpoint could be extended with a "correspondance" attribute, to support the HyTime notion of pairing source and destination link endpoints during traversal (DeRose & Durand,

We can map between Microcosm's (Hall, et al., 1993) concepts and ours: Microcosm's "button" can be modeled as a source endpoint with component locator and objectID; "specific" as a source endpoint with component locator and position (as within-component structural descriptor); "local" as an endpoint with a component locator and within-component search query; and "generic" as an endpoint with empty component locator and a within-component search query. An important difference is that our model makes these distinctions at the endpoint rather than link level. (See also Section 5.)

For the WWW, URLs correspond directly to our locSpecs. To our knowledge, there is nothing in HTML (version 2.0) corresponding to a refSpec, and thus no analogue to link endpoints. Rather, locSpecs are embedded directly in web pages, or "float" as text strings (e.g. in email messages or on paper).

CompositeReference CLASS ReferenceSpecifier

Attributes:

- PresentationSpecifier (determines how the compositeRef is displayed in a composite viewer)

CompositeReference is a specialization of RefSpec used in composite components. CompositeRef has no direct analogue in Dexter since Dexter does not support composites that contain other components (rather, only base components). Both endpoints and compositeRefs inherit a pSpec from the refSpec class. Normally, endpoints (unlike the anchors they point to) are not displayed directly. Nonetheless, graphical representations of them may appear in the user interface and in such cases, the pSpec is relevant. (An example of a "link instantiation editor" can be found in the DHM system, Grønbæk & Trigg, 1994.)

2.5 Components

In Dexter, the notion of *component* models what is often called a "node" in hypermedia parlance and provides the hypermedia "wrapping" for a document or piece of information. Following Dexter, our components include a presentation specifier and a structured collection[10] of anchors as well as some kind of content.

1994). The attribute's value would be an ID or pointer to the appropriate corresponding endpoint object.

[10] Structured collections include e.g. lists and hash tables.

The ParentID attribute is only filled if the component is *included* in some composite.[11] The RefsTo operation returns all refSpecs in the hypermedia which resolve to this component or to locations within this component.

Component:

Attributes:

- Parent ID (ID of parent composite if this component is included rather than referenced by a composite)
- Component ID (unique within a hypermedia or across the world[12])
- PSpec (attributes governing default presentation features of this component)
- Anchors (Structured collection of refSpecs)
- Content (data object or structured collection of RefSpecs)

Procedures:

- CreateAnchor
- DeleteAnchor
- ScanAnchors
- RefsTo

Dexter identified three basic kinds of components: *atomic*, *link*, and *composite*, distinguished primarily by their component contents and their behavior. In our object-oriented framework, links and composites are modeled as specializations of a basic (atomic) component class.

2.6 Links and composites

In the original Dexter model, links and composites were distinguished from atomic components by the types of their contents ("base components"). In contrast, in our framework, the endpoints and compositeRefs that distinguish link and composite components supplement rather than replace the content of an atomic component. Thus, for example, a link

[11] Whether a component should be included in or referenced by a composite depends on the application; in general, if the component ought to be deleted when the parent composite is deleted, then the component should be included in the composite. In that case, the component has a non-null parentID.

[12] A discussion of the relative merits of globally unique component identifiers is beyond the scope of this paper. Suffice it to say that uniqueness within the hypermedia is often sufficient, assuming that the containing hypermedia can be uniquely identified. The choice of identifier scope depends in part on whether the system is to support copying, moving and sharing components across hypermedias.

component can have both data contents and a set of endpoints referencing other hypermedia objects. This allows us to model constructs like the Fat Link (Jordan, Russell, Jensen, & Rogers, 1989) in a straightforward way.

LinkComponent CLASS Component:

Attributes:

- Endpoints (structured collection of RefSpecs augmented with directionality)

Procedures:

- AddEndpoint
- RemoveEndpoint
- ScanEndpoints

CompositeComponent CLASS Component:

Attributes:

- CompositeRefs (structured collection of RefSpecs)

Procedures:

- AddCompositeRef
- RemoveCompositeRef
- ScanCompositeRefs

Similarly, our CompositeComponent extends the basic component class by adding a set of compositeRefs (refSpecs), and thus can model Composites that have data contents in addition to the CompositeRefs referencing other hypermedia components. This allows us to model constructs like the NoteCards Filebox (Halasz, Moran, & Trigg, 1987) and KMS frames (Akscyn, McCracken, & Yoder, 1988), which include both text and hierarchical links. And because our compositeRefs are full fledged refSpecs, composites can point within the components they contain. Thus a TableTop or GuidedTour composite can highlight parts of the contents of its components during a presentation.

In our terminology, the WWW has only a single component type called the *page* (reminiscent of KMS frames). Pages support "permanent selections" (HTML NAME tags) which can be pointed at by URLs. Something like the effect of composite components can be obtained using pages full of URLs. However, true structuring composites aren't supported, that is, pages can't be nested; all components are in effect equally accessible in a "flat" pool.

2.7 Hypermedia

Finally, a *hypermedia* (called a *hypertext* in Dexter) includes an ID, a pSpec, and a structured collection of components. Alternatively, we could have used the term *hypermedia structure*, since the applications themselves often take responsibility for storing the content of components. That is, the hypermedia normally consists of "wrappers" for application documents and data objects, and structures for linking and organizing that content. (Applications can however, choose to store data with the hypermedia.)

Hypermedia:

Attributes:

- Hypermedia ID (unique across the world)
- PSpec (default presentation specification for components in this hypermedia)
- Components (structured collection of Components)

Procedures:

- CreateComponent
- DeleteComponent
- ScanComponents

2.8 Navigating through a hypermedia

Though we haven't the space to fully develop the runtime implications of our model, we do need to outline the basic scheme for link traversal. In our model, "following a link" usually involves these five steps:

1. Starting from a "selection" in a component's content, find an appropriate anchor for that selection or create a transient anchor for that component with an appropriate locSpec.
2. *LocSpecResolver*: Lookup link endpoints whose locSpecs match the locSpec in (1).
3. *RefSpecMapper*:Use the refSpecs from (2) to determine the set of refSpecs (other endpoints) that need to be opened. Typically these are "destination" endpoints for those links whose "source" endpoints matched in (2).
4. *RefSpecResolver*: Resolve the locSpecs belonging to the refSpecs from (3) to a set of components and locations within components (sometimes these will be anchors).
5. Display the set of "destinations" from (4) using the relevant pspecs; namely, a unification of pspecs from the endpoints of (2) and (3), from the destination component(s)/anchor(s) of (4), and from the hypermedia itself. In the simplest case, this involves displaying the destination components and highlighting any destination anchors.

Partial versions of this process happen when moving from anchors *to* (but not *through*) the links that point at them (steps 1 and 2), or by starting from link components rather than from source anchors (steps 3,

4, and 5).[13] Note also that although the above scheme is focused on link traversal, we can support something similar for composites. That is, one could start from an anchor pointed to by a composite's compositeRef, follow back into that composite, and say, open the composite's other compositeRefs. It is not clear what use this "traversal" of a composite might have. In any case, for the purposes of simplification, we restrict the discussion to link traversal in what follows.

In order to further demonstrate the role of locSpecs and refSpecs, we provide a bit more detail on steps 2, 3, and 4. Step 2 makes use of a primitive operation called *LocSpecResolver* which takes a locSpec (either free floating or embedded in a refSpec, but in any case referring within this hypermedia) and returns the set of matching refSpecs. The matching test is handled by the refSpec's EqualLocSpec procedure (see Section 2.2).[14] In practice, locSpecResolver needs to be implemented as a fast hashtable lookup. Thus all non-transient refSpecs in a hypermedia must be registered in a table which is kept up to date as refSpecs are created, modified and deleted. In the table, locSpecs (possibly belonging to anchors) are mapped to matching refSpecs belonging to endpoints. From the refSpecs returned by LocSpecResolver, it is a simple matter to follow the ParentID to reach the relevant link component.

Step 3 depends on an operation called *RefSpecMapper*, which given one (or possibly multiple) refSpecs that are endpoints for a link, returns a set of refSpecs corresponding to a subset of the link's (other) endpoints. So, for example, the refSpec of a "source" link endpoint might map to the refSpecs of that link's "destination" endpoints. In general, RefSpecMapper needs to take a "forward/backward/anyward" parameter which determines the "directionality" of the traversal, say, from source to destination link endpoints.

Step 4 uses an operation called *RefSpecResolver*, which given a locSpec follows the "instructions" implied by the locSpec attributes to either look up the resulting component or component anchor, or perform the computation dictated by the locSpec's component locator and content locator attributes. The result is a collection of components and/or locations within components.

In step 5, the destinations of the links are displayed according to the relevant pSpecs. In addition, there may be attributes of the endpoint pSpec capturing what Hardman calls "context" (Hardman, Bulterman, & Rossum, 1994). For example, for a "replacing link," when moving *from* an endpoint, a "context" attribute determines what part of the source component (or its enclosing composite hierarchy) should be replaced by the destination. The value of the attribute might be a composite spec (resolving to some composite enclosing the endpoint). Similarly, the destination endpoint of the link might specify a composite to replace when arriving at that endpoint.

More specifically, there are two ways of following such "replacing" links. When "pushing" through the link, the context specified by the source endpoint is carried over the link and replaces the context specified by the destination endpoint. When "pulling" through the link, the context specified by the source endpoint is replaced by the context specified by the destination endpoint.[15] This models Intermedia's "warm links" used to support annotations in collaborative authoring (Catlin, Bush, & Yankelovich, 1989) as well as Xanadu's "transclusions" (Nelson, 1995).

3. THE CASE FOR LOCSPECS

3.1 Revisiting a Dexter conundrum

One of the tasks for the Dexter group was to model systems with embedded "go-to" links (e.g. HyperCard) as well as those with links reified as objects stored outside of the nodes of the hypermedia (e.g. Intermedia). The Dexter group solved this problem in part by operationalizing the notion of link "directionality." A direction was associated with each link endpoint.[16] In addition to TO and FROM modelling the standard link source and destination, they used BIDIRECT to model bidirectional "connection" links in which no traversal direction was necessarily privileged (this was the case

[13] Simply resolving a free-floating locSpec to the component locations to which it refers (as in "link following" on the WWW) corresponds to steps 4 and 5.

[14] The exact behavior of this matching needs to be worked out - or left open in precisely specified ways. For example, if a potentially matching refSpec has a non-empty objectID, then must the given locSpec also have a non-empty objectID which matches in order for the refSpec to match? Alternatively, we might stipulate that if the refSpec provides a search query but no objectID, and if the query matches the locSpec, then it doesn't matter whether the locSpec also has an objectID. In other words, a "generic" link endpoint could match a fixed ("button") anchor.

[15] In both cases ("pushing" and "pulling"), the required "context" specified in the endpoint's pspec could be implemented as a free-floating locSpec. The situation gets trickier, however, when scaling up to links with multiple endpoints. Consider, for example, a search conducted on the WWW. The page resulting from the search often consists of links to the top 10 hits together with the textual context of the hit "pulled" onto the search page. Here we see the "pulling" and composing into one page of the "contexts" of multiple destinations of a computed link (the search query).

[16] Not associating the direction with the link as a whole was a novel idea at the time.

in Intermedia). Finally, a fourth direction, NONE, was used to model the source "endpoints" of embedded HyperCard-like links. Though elegant in the sense that a single concept captured two very different kinds of linking, it was unclear what the notion of directionality had to do with the question of link embeddedness. Moreover, the need for a distinction between non-directional and bi-directional link endpoints was never adequately justified.[17]

We believe the operative distinction is not one of directionality, but rather objectification. What makes HyperCard "links" different is that they are not reified for the hypermedia system, but are captured as statements in a HyperTalk script. Our model captures this by distinguishing free floating descriptions of hypermedia locations (locSpecs), from descriptions packaged as objects available to the hypermedia system (refSpecs). The HyperTalk go-to statement is modeled by a locSpec identifying a destination stack and card. In contrast, a bidirectional Intermedia link is modeled by a pair of refSpecs (endpoints) embedded in a link component as well as anchors in each of the linked components.

3.2 Human-readable locSpecs

Among hypermedia systems that support embedded linking, two kinds can be distinguished. The link can be "human-readable" (in practice, this usually means that the link's destination is specified as an ascii text string) or it can be available only through a graphic interface (and the script behind it), for example, via a click'able button. HyperCard and KMS (Akscyn, et al., 1988) are examples of the latter, Augment (Englebart, 1984) and the WWW are examples of the former. In the WWW, the link's presence is indicated by a graphic feature (click'able underlining), and in this way is similar to HyperCard. However, the heart of the link, its URL, is represented as an ascii text string which can live independently of the hypermedia. It is common practice in the internet community, for example, to exchange WWW URLs via email.

The free floating nature of such "pointers" is quite possibly an unsung source of the WWW's success. Not only can users save URLs as "bookmarks" for future use (supported earlier in Document Examiner (Walker, 1987)), but they can also be passed along to friends. And even constructed from scratch. For example, looking for a home page for a company named Acme, one might profitably try the URL, http://www.acme.com. In Section 6, we argue that future systems ought to combine the advantages of free floating locSpecs with the stability and searchability of reified refSpecs.

[17]For a discussion of other problems with Dexter's notion of link directionality and an approach to addressing them, see Grønbæk & Trigg (1994).

4. THE CASE FOR REFSPECS

4.1 The traditional arguments

Two reasons are traditionally given for the importance of link objects stored apart from hypermedia "nodes" or documents. First they enable one to answer structural questions about the hypermedia, for example, which other components are linked to this one? Access to explicit link structures makes it possible, for example, to construct and maintain graphical browsers over parts of the hypermedia. The second reason is especially important in distributed, collaborative environments. Storing links externally to the documents allows the interlinking of information in write-protected files, or on read-only media like CD-ROMs. (DeRose and Durand (1994: 118-119) make comparable arguments in distinguishing between "inline" and "out-of-line" links.)

As we have pointed out, the WWW currently does not support refSpecs. With ongoing work on security and web-based access permissions, it may soon be possible to support small groups of collaborators sharing group-writable web pages. It is unclear, however, how web users could interlink remote pages (which they can't modify) in such a way that the resulting links could be made visible to future readers of those pages.

Just as important is the current lack of access provided by the WWW to the implicit structures formed by embedded URLs. This may change soon, however. Though graphical structure browsers have not to our knowledge been made available on the web, it seems likely that "partially consistent" link maps will soon appear. The web "crawler" programs that today index most of the text on the web, also have access to URLs. Thus graphical browsing of WWW implicit structures using "approximate" representations may be just around the corner.

Another kind of justification for refSpecs concerns the management of dynamic structures.

4.2 Dynamic structures

Many modern hypermedia systems support some notion of *computed link*. The destination of such a link is not represented explicitly, but is computed at the time the link is followed. In some cases, the result may be an existing document or anchor within a document; in others, traversing the link may construct the destination document on the fly. The most common example of the former are links that perform searches across the hypermedia. Our notion of locSpec can explicitly represent such computations.

Less common has been support for computed links that construct their destinations. Although users have recently begun to experience this phenomenon on the WWW. A question that arises concerns the status of the newly constructed destination document. When

does it "vanish," if ever? Can one link to it, and if so, will it persist? In the WWW, the precise fate of such transient pages depends on the particular browser and server implementations; usually they last until flushed from a local cache. Saving a "bookmark" for such a page saves only the specification of the page, not its current contents. Accessing that URL later will recompute the page. Contrast this with DHM where the transient page (say, a "search composite") lasts until the last link to it is removed. Such pages can also be optionally rendered permanent in the hypermedia (Grønbæk & Trigg, 1994).

This suggests another reason for supporting refSpecs. Users could retain the product of running a computed link simply by creating a "proper" link to it, that is, one represented and maintained explicitly in the system. Today on the WWW, for example, one must download the source of the transient page and use it to construct a new permanent page, usually on one's local host. If DHM-like "garbage collectible" pages are to be supported over the WWW, however, issues of page storage will have to be addressed.

5. EXAMPLE OF USING THE EXTENDED DEXTER MODEL

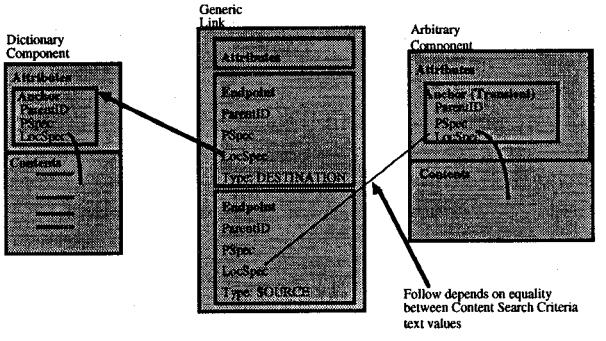

Figure 2: Modeling Microcosm Generic Link

This section provides an example of mapping a dynamic hypermedia structure, Microcosm's Generic link (Hall, et al., 1993), to our extended Dexter model. A Generic link (Figure 2) is accessible from the contents of arbitrary components; traversing the link results in the presentation of a fixed destination endpoint in a given component. For example, the text string "hypertext" (regardless of where in the hypermedia it appears) could act as the source for a generic link with destination in a dictionary component. We model the Generic link as a link object with two endpoint objects, one with Type SOURCE and a LocSpec with 'Computation spec' query set to the text string "hypertext." The other endpoint object has Type DESTINATION and a locSpec with Component ID set to the dictionary component. The Object ID of the destination locSpec picks out an anchor whose LocSpec in turn locates the explanation of the word "hypertext" in the dictionary contents.

Following a generic link from an arbitrary component is performed by generating a transient anchor on the fly in the component's instantiation. The transient anchor gets a locSpec with its 'Computation spec' query set to the selection in the component contents, i.e. "hypertext". The transient anchor is passed to the standard FollowLink procedure. Note furthermore, that this procedure makes use of the hashtable described in Section 2.8 in which the Generic link's SOURCE refSpec will already have been registered. Thus our hashtable serves the role of a Microcosm *filter*.

Note that according to our model, the generic'ness of a Generic link is not associated with the link as a whole, but rather with its source endpoint. This suggests a simple generalization of generic links. Consider a link similar to the one in Figure 2, but having another generic source endpoint whose locSpec has the string "hypermedia" as 'Computation spec' query, and another destination endpoint identifying an anchor in the dictionary component for a "hypermedia" entry. This link would enable accessing both explanations of the two related concepts "hypertext" and "hypermedia" from textual selection of either of the words in an arbitrary component.

6. CONCLUSION

The Extended Dexter model described here covers systems that represent and store links as objects separate from the components (documents) they connect. At the same time, it models systems whose "links" are embedded in the contents of documents, or which float freely outside the scope of the hypermedia. Although the original Dexter model covered similar territory, the concepts of locSpec and refSpec we have introduced capture the two styles of linking in a more intuitive, integrated, and object-oriented manner.

Furthermore, our model covers modern forms of dynamic hypermedia structures like computed links and "generic" links which have appeared since the original Dexter model. Indeed, our model suggests modifications or generalizations to such facilities for future hypermedia designs.

Finally, we believe our model takes a step toward more open hypermedia systems. RefSpec-based anchors and link endpoints support the integration of third party application documents in hypermedia networks, while text-encoded locSpecs capture a critical feature of the global distributed hypermedia made popular by the WWW.

The logical next step, we believe, is to encourage systems that combine both styles. Consider for example, a designer wishing to support the integration

of global hypertext (ala WWW) with the local applications and interlinking protocols supported on the workstations of her organization. We expect the need for such projects to grow in the future. Their success may well depend on the ability to smoothly integrate link objects on the local or enterprise side, with embedded and free floating references on the wide-area side.

References

Akscyn, R., McCracken, D., & Yoder, E. (1988). KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations. *Communications of the ACM, 31*(7), 820-835. July.

Catlin, T., Bush, P., & Yankelovich, N. (1989). InterNote: Extending a Hypermedia Framework to Support Annotative Collaboration. In *ACM Hypertext'89 Proceedings*. New York: ACM Press, 365-378.

Davis, H., Hall, W., Heath, I., Hill, G., & Wilkins, R. (1992). Towards an Integrated Information Environment with Open Hypermedia Systems. In *ACM Hypertext'92 Proceedings*. New York: ACM Press, 181-190.

DeRose, S. J., & Durand, D. G. (1994). *Making hypermedia work: A user's guide to HyTime*. Boston/Dordrecht/London: Kluwer.

Englebart, D. C. (1984). Authorship provisions in Augment. In Proceedings of *the IEEE COMPCON*. San Francisco, California, IEEE, 465-472

Garzotto, F., Paolini, P., & Schwabe, D. (1993). HDM - A model-based approach to hypertext application design. *ACM Transactions on Information Systems, 11*(1), 1-26. January.

Grønbæk, K., Kyng, M., & Mogensen, P. (1993). CSCW challenges: Cooperative design in engineering projects. *Communications of the ACM, 36*(4), 67-77.

Grønbæk, K., & Trigg, R. H. (1994). Design issues for a Dexter-based hypermedia system. *Communications of the ACM, 37*(2), 40-49. February.

Haan, B. J., Kahn, P., Riley, V. A., Coombs, J. H., & Meyrowitz, N. K. (1992). IRIS Hypermedia Services. *Communications of the ACM, 35*(1), 36-51.

Halasz, F., & Schwartz, M. (1990). The Dexter Hypertext Reference Model. In J. Moline, D. Benigni, & J. Baronas (Eds.), *Proceedings of The Hypertext Standardization Workshop*. Gaithersburg, MD: National Institute of Standards, 95-133. January.

Halasz, F. G., Moran, T. P., & Trigg, R. H. (1987). NoteCards in a Nutshell. In *Proceedings of ACM CHI+GI'87*. New York: ACM Press, 45-52.

Hall, W., Davis, H., Pickering, A., & Hutchings, G. (1993). The Microcosm Link Service: An Integrating Technology. In *Proceedings of ACM Hypertext'93*. New York: ACM Press, 231-232.

Hardman, L., Bulterman, D. C. A., & Rossum, G. v. (1994). The Amsterdam Hypermedia Model: Adding time and context to the Dexter model. *Communications of the ACM, 37*(2), 50-62. February.

Isakowitz, T., Stohr, E. A., & Balasubramanian, P. (1995). RMM: A methodology for structured hypermedia design. *Communications of the ACM, 38*(8), 34-44. August.

Jordan, D. S., Russell, D. M., Jensen, A.-M. S., & Rogers, R. A. (1989). Facilitating the Development of Representations in Hypertext with IDE. In *ACM Hypertext'89 Proceedings*. New York: ACM Press, 93-104.

Kacmar, C. J., & Leggett, J. J. (1991). PROXHY: A process-oriented extensible hypertext architecture. *ACM Transactions on Information Systems, 9*(4), 399-419. October.

Malcolm, K. C., Poltrock, S. E., & Schuler, D. (1991). Industrial Strength Hypermedia: Requirements for a Large Engineering Enterprise. In *Proceedings of ACM Hypertext'91*. New York: ACM Press, 13-24.

Meyrowitz, N. (1989). The missing link: Why we're all doing hypertext wrong. In E. Barrett (Ed.), *The Society of Text*. Cambridge Mass: MIT Press, 107-114.

Nelson, T. H. (1995). The heart of connection: Hypermedia unified by transclusion. *Communications of the ACM, 38*(8), 31-33. August.

Pearl, A. (1989). Sun's link service: A protocol for open linking. In *Proceedings of Hypertext '89 Conference*. New York: ACM Press, 137-146. November.

Schnase, J. L., Leggett, J. J., Hicks, D. L., & Szabo, R. L. (1993). Semantic data modelling of hypermedia associations. *ACM Transactions on Information Systems, 11*(1), 27-50.

Walker, J. H. (1987). Document Examiner: Delivery Interface for Hypertext Documents. In *ACM Hypertext'87 Proceedings*. New York: ACM Press, 307-323.

Appendix: HyTime comparison

HyTime (DeRose & Durand, 1994), is a landmark hypermedia standard adopted by the ISO after the first publication of the Dexter model. Unlike Dexter's "data model" approach, HyTime is primarily intended as a representation and interchange language for hypermedia documents. (Dexter-based models do not prescribe markup-based representations, though Dexter-compliant systems are expected to be able to output HyTime-compatible textual representations of a hypermedia.) The following table mapping HyTime concepts to ours, is based on our reading of DeRose & Durand, especially Chapters 7 and 8.

HyTime concept	Our Dexter-based concept
Anchor (p. 110): Any data item which can be unambigously located. Anchor is used as a general term and not as a declared entity in HyTime.	≈ "Locator": an abstract term refering to any referencable entity in the hypermedia. Note that in our model the term 'anchor' designates a certain kind of refSpec appearing inside components.
locator (p. 67): a structure that references an anchor. Can be used to locate "things" that lack explicit IDs. The HyQ query language (Chapter 9) supports computed links and aggregates.	≈ LocSpec: specifies a location in any kind of data, and can be human-readable and/or free-floating outside the hypermedia system. Can locate by means of object ID, structural properties, and/or a script or query
cLink (p.112) = Contextual link: the clink itself is the source end while the destination end is given by the embedded address. (Similar to an HTML link tag: <A HREF= "URL">...</A>)	≈ A locSpec embedded in the text of a document.
iLink (p.115) = Independent link: an entity separate from document contents which includes a set of linkends. The number of linkends is fixed for each type of link.	≈ Link (subclass of component): an object separate from document contents containing a collection of 0 or more EndPoints. There is no restriction on the number of endpoints. Thus new endpoints can be added at runtime without having to change the link's type.
Linkend (p. 110): an SGML IDREF or any Location address	≈ Endpoint: includes a reference to a locSpec. A given locSpec may be referenced by many EndPoints (or other refSpecs), but a given EndPoint belongs to only one link.
Anchor Role (p. 115): Direction + number of endpoints	≈ Direction attribute in an EndPoint object. Note that the number of endpoints can be changed dynamically.
Link end terms (p. 113): holds information for the application to use for presentation of an anchor	≈ PSpec: information about the presentation of data from linked components.
≈ An SGML document (p. 39)?	Component: an abtract superclass for objects which "wrap" data objects or documents, making them anchorable for links. Note that both Links and Composites are subclasses of Component and can thus be linked.
≈ Aggregate (with "aggloc=aggloc") (p. 153): a collection of locators referencable as a single entity.	Composite: a subclass of components which may reference or contain (by inclusion) other components. The contents may be specified either intensionally (computed) or extensionally (explicit membership).

HyTime is an open standard in that it does not require that all interlinked documents be represented in SGML. In particular, HyTime includes a "notloc" mechanism (p. 147) which can be used to specify locations in special media or data notations. As the authors state (p. 148), "[notloc] can help bring the benefits of HyTime's standardization and portability to a wider range of applications than might otherwise be possible." We agree that standard notations can eventually make the job of hypermedia system developers easier. However, rather than designing or promoting document representation standards, SGML-based or otherwise, our primary concern is with the interlinking of documents in their existing and evolving formats. In effect, we are in "notloc mode" most of the time. For us, successful open hypermedia services depend more on the openness of applications for communication and API programming than on the formats they use to represent data objects on files. Indeed, we believe that the potential for hypermedia will be furthered by keeping standards for document linking separate from those for document representation. In any case, we welcome further discussion with HyTime advocates on the relative merits and complementarities of our approaches.