Link Architecture for a Global Information Infrastructure

by

Jeffrey R. Van Dyke

Submitted to the Department of
Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements
for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June, 1995

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, Massachusetts 02139

This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract No. DABT63-92-C-0002.

Link Architecture for a Global Information Infrastructure

by

Jeffrey R. Van Dyke

Submitted to the Department of Electrical Engineering and Computer Science
on May 30, 1995
in partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The notion of a link to represent an explicit relationship or association between entities has been utilized by numerous hypertext systems to provide a variety of capabilities, including quotation, navigation, annotation and knowledge structuring. The link mechanism described herein provides the ability to relate entities in a global information infrastructure, the Information Mesh.

The implementation of a link architecture shows the feasibility of a minimum mechanism to provide a rich set of relationship expressions as an element of a global information infrastructure. Mesh objects are shown to require a composite object mechanism and enhancements to their substructure interface. Mesh link endpoints allow the description of an object, some aspect of an object or a component of an object. The resulting Mesh link implementation provides first-order linking in an extensible and flexible architecture.

Thesis Supervisor: Dr. Karen R. Sollins
Title: Research Scientist, Laboratory for Computer Science

Acknowledgments

Special thanks to my thesis advisor, Karen Sollins for her guidance and support. Her vision of an Information mesh provided the foundation for this effort. This thesis would not have been possible without her patient encouragement and ocassional prodding.

Thanks to Benvenido Vélez-Rivera and Alan Bawden for their detailed explanations of the Mesh kernel and Mesh object system. Their efforts provided a springboard to the implementation of the link mechanism described in this thesis.

Thanks to Mitch Charity and Chris Lefelhocz for reading and commenting on various portions of this document.

Thanks to the members of the Advanced Network Architecture group at the MIT Laboratory for Computer Science. Dave Clark, John Wroclawski, Janey Hoe, Lewis Girod, Tim Shepard, Tim Chien, Matt Condell, Robert McInnear, Lisa Trainor and Garret Wollman all created an atmosphere of motivation and support.

Finally, a deepfelt thank-you to my parents, Roger and Diane Van Dyke, and brother, Bryan, for their love and care.

Contents

This page intentionally left blank.

Chapter 1

Introduction

This thesis examines a *link* mechanism for describing object relationships in a long lived, global information infrastructure, the Information Mesh. The resulting Mesh link implementation provides a rich and flexible mechanism to relate information in the Mesh. Minimum link, object and system capabilities necessary to support such capabilities are described.

1.1 Background

The Information Age has created a need to manipulate a vast and ever increasing amount of data. As an example, consider the Internet: the traffic related to information manipulation has increased tremendously in the past few years [1]. Corresponding to this growth has been an increasing need for tools to manage information, particularly a mechanism to connect and relate knowledge.

The notion of connecting and relating knowledge has been a compelling vision since at least 1945 when Vannevar Bush suggested the implementation of a vast knowledge base [6]. These ideas have been further developed in hypertext systems, such as Xanadu [17], Aquanet [16] and World Wide Web [1], where links are utilized to explicitly represent a relationship or association between entities.

Hypertext links provide a powerful mechanism to relate information. In the World Wide Web, links provide a means of information navigation. Xanadu utilizes links for quotation, navigation, annotation and commentary. Aquanet links are utilized to represent and discuss knowledge structures. Thus, hypertext link utilization includes: navigation, quotation, annotation and knowledge representation.

1.2 Link Architecture

We describe a link mechanism to describe object relationships in a long lived, global information infrastructure. Our framework for this effort is the Information Mesh Project: an effort to provide a minimum set of universal commitments necessary to provide a long-lived global architecture for network-based information reference, manipulation and access. The Information Mesh Object System provides Mesh objects as the nodes of Mesh links.

The overall goal is to describe a minimum link mechanism which provides a flexible and rich set of relationship expressions. One result of this effort is a description of the minimum system, node and link capabilities necessary to support Mesh links.

1.3 Organization

The examination of Mesh links begins with a description of several representative hypertext systems in Chapter 2. System requirements, node capabilities and link characteristics are described in this section. These characteristics and the concluding observations are utilized throughout the remaining chapters. Chapter 3 describes the overall Information Mesh, the Mesh kernel and the Mesh Object System. The system requirements of the Information Mesh are described and the capabilities of Mesh objects are described. Chapter 4 examines enhancements to the Mesh Object System to better utilize Mesh objects as nodes of Mesh links. Chapter 5 describes a Mesh link architecture and demonstrates the flexibility of Mesh links in several examples. Chapter 6 summarize the overall results and open issues.

Note that security and privacy issues will not be examined except where they directly affect overall link design.

Chapter 2

Related Work

The notion of a *link* to represent an explicit relationship or association between entities has been utilized by hypertext to provide a variety of capabilities, including quotation, navigation, inclusion, annotation and knowledge structuring. In this chapter, we examine a variety of hypertext systems: Memex, Xanadu, the World Wide Web, Aquanet and the Dexter Hypertext Reference Model. We examine their use of links and describe how they confront hypertext issues, including:

1. **System issues:** How do system characteristics enhance or limit linking?
   - **minimum requirements**: basic system expectations and requirements
   - **scalability**: mechanisms to deal with large system issues
   - **flexibility**: provisions for a variety of hypertext nodes and links
   - **security**: mechanisms to prevent unauthorized access
   - **privacy**: mechanisms to ensure privacy
2. **Node attributes:** How do nodes support linking?
   - **naming**: identification of nodes
   - **typing**: describing node characteristics, semantics and invariants
   - **substructure interface**: exposing node substructure for linking
   - **composites**: combining nodes
   - **versioning**: supporting node changes

3. **Link issues:** How are links exposed to the overall system?
   - **link utilization**: overall use and characteristics of a link
   - **link relationships**: ability of link to "talk about" or express relationships between entities (including other links)
   - **link independence**: ability to exist separate from nodes
   - **endpoint capabilities**: what can links associate?

Note that we focus on the issues of scalability, node typing (as a means of achieving flexibility among other things), substructure interface, endpoint capabilities and overall link utilization.

2.1 Memex

The notion of relating a vast domain of information using some associated structure was first described in Vannevar Bush's vision of the Memex: "A device in which an individual stores his books, records, and communications, and which is mechanized so that it may be consulted with exceeding speed and flexibility. It is an enlarged intimate supplement to his memory [6]."

Bush's work was distinguishable for its inclusion of an association mechanism: the examination of one item in the system would suggest another. Bush envisioned this mechanism working in a fashion similar to the human brain: "With one item in its grasp, it snaps instantly to the next that is suggested by the association of thoughts, in accordance with some intricate web of trails carried by the cells of the brain [6]."

It is largely agreed that one outgrowth of Bush's vision was *hypertext*, an information management mechanism in which data is stored in nodes connected by links.

2.2 Xanadu

Ted Nelson's Xanadu Project [17] is an influential examination of a large hypertext system. Xanadu utilizes links to provide "a connection between parts of text or other material... made by individuals as pathways for the readers exploration.. [17]." The overall goal of Xanadu is a distributed system of documents connected by links.

Xanadu documents are the fundamental unit of storage. Indeed, everything in the Xanadu system is a document. Xanadu documents "may contain text, graphics, links... or any combination of these... [17]." Xanadu documents provide no information hiding or abstraction layer; they expose their entire structure and contents, along with associated versioning information, in a manner allowing Xanadu links to relate any portion of a document.

Xanadu links are composed of "end-sets". Each end-set indicates "spans" or regions of text in Xanadu documents. Thus, Xanadu supports *span-to-span* linking by allowing its links to relate regions of text. The typical Xanadu link is a three end-set structure: a "from-set" which is an arbitrary collection of spans specifying the source of a link, a "to-set" which specifies the destination of a link, and a set specifying the link type or relationship being expressed.

Xanadu links are contained in nodes: "Each link resides in one place, the document that contains it. Links, just like text, are owned. Every link is part of a particular [document] and has an owner [17]." Links can relate other links by connecting to the link portion of a document.

Xanadu links maintain associations across document versions. "Essentially, the link seizes a point or span (or any other structure) in the [document] and holds onto it. Links may be refractively followed from a point or span in one version to corresponding places in any other version. Thus a link to one version of a [document] is a link to all versions [17]." Unfortunately, the mechanism for a link to "hold" onto a node across versions is dependent on specific characters remaining invariant: "a link is attached... to specific characters and simply stays with these characters wherever they go [17]." Note that this mechanism will break under a variety of conditions, including wording changes.

The greatest weakness of the Xanadu system is its expectation of complete availability of certain system information: "..every change must be known throughout the network the instant it happens [17]." In particular, Xanadu expects that all links to any particular Xanadu document will always be determinable. "The reader should be able to ask, for a given document, 'What connects here from all from other documents?' – and be shown all these outside connections without appreciable delay [17]."

In summary, Xanadu provides a link mechanism to allow reader exploration of documents. Xanadu documents expose their entire substructure in a manner allowing links to relate any portion of a document. Xanadu links are composed of end-sets and are contained in nodes. Xanadu links maintain associations across document versions and provide a mechanism for determining all links to a particular Xanadu document.

2.3 World Wide Web

The World Wide Web [1] is perhaps the best known, most widespread and most successful example of a distributed hypertext system. The Web allows navigation via links across the Internet and between documents. The incredible growth and success of the World Wide Web has exhibited the power of a distributed hypermedia system connecting various sites using "links".

The overall World Wide Web (WWW) paradigm is documents connected by links. WWW links exist in the WWW documents that are their sources. Each WWW link specifies a relationship between two entities: the document in which the link is contained, and an identified destination document. WWW documents are specified in HTML [5].

World Wide Web documents are identified through the use of location (a Universal Resource Location or URL [2]) rather than in a location independent manner such as Universal Resource Names [21]. [1] This prevents the relocation of Web objects – they can not be moved from the location described by their URL. For transmission purposes, WWW document content is specified using Internet Media Types [18].

WWW documents emphasize human browsing, and do not explicitly encode semantics. There is, for instance, no mechanism to specify that a specific page is

[1] The latest draft of HTML 3.0 [19] proposes the addition of the (optional) URN attribute to describe the universal resource name for an HTML document.

an individual person's "home page" other than to imply it in the text or assume it from the URL associated with the Web page. In a related manner, HTML provides minimum mechanisms to assist browsers in presenting new markup structures. If a particular HTML markup is encountered for which the Web browser lacks knowledge, there is little or no fall-back; the Web browser can either ignore the markup or display the ASCII text representation. In short, there is no standard way to classify a Web document; any meaning must be determined from the accompanying HTML which, at least presently, provides minimum capabilities for such descriptions.[2]

Documents expose internal content for linking through the use of an *anchor*. In the WWW an anchor specified section of the WWW document is the source and/or destination of a WWW link. An anchor HREF attribute specifies the beginning of a link. An anchor NAME attribute specifies an identifier whose reference allows the anchor to be the target of a link. Anchors can nested, but can not overlap one another. Anchors are limited in that they are merely arbitrary portions of document – there is no document typing mechanism which would allow associating anchors with a certain document type (such as the aforementioned "home page") or document substructure.[3] In short, WWW anchors lack the ability to be associated in some formal manner with a generalized structure, such as a particular document type.

WWW links are one-way, two-ended and document-based. Links always describe a relationship between exactly two documents; there are no mechanisms to relate more than two entities. Links must be contained in one of the two documents they associate. The Web provides a mechanism to allow servers to add links to documents "by those who do not have the right to alter the body of a document [4]", but servers are not required to provide this functionality. There are no mechanisms to link two documents if the servers of both documents refuse the additional links. WWW links are not first class and therefore can not exist independently of the documents they link.

[2] The latest draft of HTML 3.0 [19] suggests the utilization of a "ROLE" attribute which is a listing of SGML name tokens "that define the role this document plays [19]."

[3] The latest draft of HTML 3.0 proposes the addition of an ID mechanism to associate document elements with anchors. Further, the draft suggests the addition of a CLASS mechanism to subclass HTML elements.

WWW link types are specified by a relationship name. However, the present use of WWW link relationship names is extremely limited. This is particularly frustrating because early WWW documentation[3] suggested several link names to describe "relationships between documents" and "relationships about subjects of documents". The current HTML 2.0 draft [4] has minimal discussion of link relationships, merely stating: "Relationship names and their semantics will be registered by the W3 Consortium. The default value is void." The latest draft of HTML 3.0 [19] suggests an expanded use of link relationships to provide specific navigation buttons or equivalent mechanisms.

WWW links can become "dangling" links. For example, a referenced WWW document may rename or remove the necessary anchor. Worse, the referenced document may move or be removed in a manner that "breaks" its prior URL. There is no mechanism for a referenced WWW document to expose invariants in anchors to allow a link to ensure it is less likely to become "dangling".

In summary, the WWW allows navigation via links across the Internet and between documents. WWW documents are identified by location, emphasize human browsing and expose internal content for linking through the use of anchors. WWW links are one-way, two-ended, document-based and can become "dangling" links. WWW link types are specified by a relationship name.

2.4 Aquanet

Aquanet [16] is a hypertext knowledge structuring tool designed to allow users to graphically represent information and explore its structure. Aquanet allows users to interpret and organize ideas using Aquanet's linking structure to connect and express ideas. Overall, Aquanet provides an examination of utilizing hypertext facilities in the realm of knowledge representation.

Aquanet objects (both nodes and links) are typed, structured frame-like entities. Every Aquanet object is an instance of some type. A type's definition specifies slots, type(s) of objects that can fill each slot, and the graphical appearance of the object. Type definitions are organized into a multiple inheritance hierarchy. "Aquanet objects of a given type include not only the slots defined by their type but also the slots that they inherit from their supertype(s) [16]." The inheritance rules of the Aquanet type hierarchy are taken directly from the Common Lisp Object System specification [14].

Aquanet nodes and links are distinguished by their use of slots. Node slot values are a named set of contents restricted to primitive datatypes such as text, images, numbers, strings, etc. Link slot values may be primitive datatypes or other Aquanet objects. Aquanet links can be viewed as containing named and typed endpoints.

Aquanet links are utilized as part of the definition, development and display of "knowledge structures". [4] As an example, an "Argument relation" is expressed as an Aquanet link containing three slots: the Conclusion, the Grounds and the Rationale. Each slot can be filled by either a "Statement node" (an Aquanet object containing a text slot) or another Argument relation.

In summary, Aquanet utilizes a type hierarchy to describe object types and multi-ended links to provide enhanced knowledge structuring capabilities.

2.5 Dexter

The Dexter Hypertext Reference Model provides an abstract model of hypertext systems which describes the entities and mechanisms which allow users to create, manipulate and examine hypertext [12]. The overall goal of Dexter is two-fold. First, Dexter formalizes some of the hypertext notions we have examined, thus providing a vocabulary that can be utilized to describe a particular hypertext system's functionality and characteristics. Second, Dexter provides a model of the important abstractions found in a wide variety of hypertext systems, and thus necessary to incorporate into a flexible link mechanism.

In this section, we examine Dexter in considerable detail. First, we examine

[4] The term knowledge structure refers to "..an interconnected network of information-bearing nodes that are used to represent the primitive objects and their interconnection in some domain of discourse [16]."

the Dexter storage layer which contains components that serve as nodes and links. We separately examine the composite information and base components which together construct all Dexter components. Additionally, we describe Dexter's storage layer functions and runtime layer. Finally, we describe Dexter invariants and summarize Dexter limitations.

2.5.1 Dexter Storage Layer

The Dexter storage layer models the node/link network structure of hypertext. It is composed of a database of data-containing components interconnected by relational links. The storage layer focuses on the mechanisms by which link and non-link components are 'glued-together' to form hypertext networks.

The fundamental entity in the storage layer is a *component*. Components are what are typically thought of as 'nodes' and 'links' in a hypertext system. The storage layer of Dexter doesn't attempt to model the overall content and structure of components, but treats components as largely generic containers of data. Despite the overall indifference to component contents, Dexter requires that each component expose *component information* and utilize a *base component*. Component information is described in Section 2.5.2 and base components are described in Section 2.5.3.

Also associated with the storage layer are two functions: a *resolver* function and an *accessor* function. Together they are jointly responsible for retrieving components from the storage layer based on the specifications of the components. The exact nature of these mechanisms is described in Section 2.5.4.

2.5.2 Dexter Component Information

Dexter requires that each component in the storage layer expose *component information*. Component information describes certain properties of the component and provides a fundamental interface to the component.

Component information includes: unique identification, anchoring, presentation specification and attribute/value pairs.

- *Unique Identifier*

  Each Dexter component has a unique identifier (UID) assumed to be "uniquely assigned to components across the entire universe of discourse [12]."

- *Anchors*

  Each Dexter component contains a sequence of *anchors* that index into the component. Dexter anchors provide an indirect addressing mechanism for specifying the internal structure of a component in a manner which does not depend on knowledge of the internal structure of a document. Dexter links utilize anchors to relate component substructure.

  A Dexter anchor consists of two parts: an *anchor id* and an *anchor value*. The *anchor id* is an identifier which uniquely identifies an anchor within the scope of the component it occupies. The *anchor value* is an arbitrary value that specifies some location, region, item or substructure within a component. The anchor value is interpretable only by the applications responsible for handling the content/structure of the component. Dexter anchors can overlap.

  Anchors allow Dexter to support linking across component versions. As a component changes over time, the anchor value changes to reflect modifications to the internal structure of the component, "[t]he anchor id, however, remains constant, providing a fixed referent that can be used to specify a given structure within a component [12]."

- *Presentation Specification*

  The *presentation specification* is a primitive value containing information about how the node contents should be presented to the user. Presentation specifications are described in more detail in Section 2.5.5.

- *Attribute-Value Pairs*

  Finally, Dexter components provide the ability to set and retrieve arbitrary attribute/value pairs. The attribute/value pairs can "be used to attach any arbitrary property (and its value) to a component. For example, keywords can be attached to a component using multiple 'keyword' attributes [12]."

Note that Dexter does not provide a formal component type model. Some component attributes can be determined by examining attribute-value pairs, but no formal type system mechanism is specified. Some descriptions of Dexter suggest modeling a component type system by "adding to each component a 'type' attribute with an appropriate type specification as its value [12]."

2.5.3 Dexter Base Components

Dexter *components* are composed of a *base component* together with the *component information* described in Section 2.5.2. The *base components* in the Dexter storage layer are: *atomic components*, *composite components* and *links*.

Atomic Components

Atomic components are the finest grain members of the storage layer. Atomic components are largely opaque objects; the storage layer knows little about the contents of atomic components or the "within-component" layer. Atomic components may contain chunks of text, graphics, images, etc.

Composite Components

Composite components are constructed out of other components. The composite relationship is restricted to a directed acyclic graph (DAG) of base components; no component may contain itself either directly or indirectly and composites are only composed of *base components*.

Finally, it is not clear how the linking mechanism is provided with composite components. Dexter does not describe how anchors are related to composites; no mention is made of how anchors should refer to base components in a composite.

Links

Links associate Dexter components by describing a relationship between components. Dexter links describe their relationship using a sequence of two or more *specifiers*. Each specifier describes the entities being related, the direction of the relationship and the presentation mechanism by which to display the entities. Dexter links are first class and Dexter links can relate Dexter links.

Dexter utilizes composites to model hypertext systems in which links are not independent, but are embedded in nodes. An example of this application of composites is the KMS [20] hypertext system: "All links in KMS are *embedded* within the frame (component) containing the source anchor. Since links are also components in the Dexter model, it may be argued that a frame in KMS is actually a *composite component* [15]."

Dexter utilizes *specifiers* to describe the link relationship. The specifier structure contains: a component specification, an anchor id, a direction and a presentation specification.

- *component specification* provides a description of the component being linked. This description can be utilized by the storage layer's resolver function to produce a set of component UIDs matching the description.

- *anchor id* specifies the anchor to be utilized in the resolved component.

- *direction* encodes link endpoints as FROM, TO, BIDIRECT or NONE. Dexter allows duplicate direction values with the constraint that at least one specifier have a direction of TO or BIDIRECT.

  There are many different notions of directionality. Grønbaek and Trigg [11] have identified at least three types: semantic direction, creation direction and traversal direction. Dexter does explicitly utilize a particular notion of directionality; Dexter provides directionality as a mechanism to support directionality semantics in existing hypertext systems with Dexter's two-way links. For example, Dexter models a one-way link system (such as HyperCard [10]) by using two-way links with the source end having a direction value of NONE and the other end having a direction value of TO[5]

- *presentation specification* is a primitive value that helps the runtime layer determine how the associated descriptor should be presented to the user. We will discuss the presentation specification in more detail in the discussion of Run-Time issues in Section 2.5.5.

Note that for a particular specifier, the component specification allows the return of a set of UIDs, but the other aspects of a specifier structure are single valued and statically determined. This implies that all components resolvable from a particular component specification must support the same anchor id and presentation.[6]

2.5.4 Dexter Storage Layer Functions

As we have previously mentioned, the storage layer utilizes a resolver and accessor function to retrieve components.

- *Accessor Function*

  The accessor function of the hypertext is responsible for "accessing" a component, given its UID. That is the accessor function is responsible for retrieving the component corresponding to a given UID.

- *Dexter Resolver Function*

  The resolver function must be able to produce all possible valid component UIDs for any given description or "component specification".

  Dexter remains silent on the mechanism and implementation of resolver functions, including the domain and syntax of specifications, but justifies their need:

[5] HyperCard links can only be traversed from source to destination. "This is because HyperCard links are implemented as 'GO' statements in a script in the link's source component. This also means that links cannot normally be seen from their destination cards [11]."

[6] In [25], Penzo, Sola and Vitali propose modifications to Dexter to support dynamic determination of anchor ids.

"The use of UIDs as a basic addressing mechanism in hypertext may be too restrictive. Rather, when [the component specification described in a specifier of a] link is followed, the specification must be 'resolved', if possible, to a UID (or set of UIDs) which then can be used to access the correct component(s)."

2.5.5 Dexter Runtime Layer

The runtime layer specifies the tools for a user to access, view and manipulate the node/link network structure. The runtime layer tools can treat components as more than generic containers of data – utilizing the actual contents.

The runtime layer utilizes the *presentation specification* values associated with components and link specifiers to determine how a component should be presented to an end user. "Thus, the way in which a component is presented to the user can be a function not only of the specific hypertext tool that is doing the presentation (i.e., the specific run-time layer), but can also be a property of the component itself and/or of the access path (link) taken to that component [12]." Thus, the runtime layer is the layer at which dynamic mechanism is determined, while the storage and component level mechanisms previously described implement hypertext as an essentially passive data structure.

2.5.6 Dexter System Invariants

The Dexter model requires that several invariants be maintained at all times by the hypertext system. These invariants are expected to be implemented in a fashion to ensure they are maintained when creating, modifying or utilizing components.

Among the Dexter invariants are:

- Link specifiers must have at least one specifier with the direction of TO or BIDIRECT. Thus, all links must point to some component.
- The accessor functions must be an invertible mapping from UIDs to components. This implies that every component must have exactly one UID

- The resolver function must be able to produce all possible valid UIDs. This implies that any possible component descriptions must be resolvable to a complete set of component UIDs.

- Composite components must contain no cycles in the component/subcomponent relationship. Thus, no component may be a subcomponent (directly or transitively) of itself.

- Links may not be 'dangling'. The specifiers of a link must always resolve to a set of components containing the associated anchor id. Any component changes must be reflected in links. Thus, any Dexter-based hypertext system must ensure that any component changes result in the immediate update and modification of links to reflect the changes.

2.5.7 Dexter Limitations

Dexter is limited in several respects.

1. The Dexter system invariants ignore large distributed system issues, such as unavailability. For instance, the need to prevent 'dangling links' ignores the difficulty of providing and maintaining such information across a widely distributed system.

2. Dexter does not explicitly provide a component typing mechanism. Some component attributes can be determined from examination of the component information such as attribute-value pairs, but there is no formal mechanism to associate a component type with invariants such as the anchors available.

3. Dexter anchors are little more than arbitrary identifiers of values. Dexter provides no mechanism to associate formally a particular set of anchors with a particular type of component. Nor is there any way to specify certain content characteristics with particular anchor ids. Finally, Dexter anchors do not provide any context; Dexter assumes that all component anchors are valid at all times.

4. Dexter link specifiers are limited in dynamic endpoint component determination – the component specification portion permits the dynamic determination of a set of UIDs, but the other specifier portions are single valued and statically determined. Thus, all components resolvable from a particular component specification must support the same anchor id and presentation.

5. Dexter provides only limited motivation for link directionality. Dexter directionality is motivated as a mechanism to support directionality semantics in existing hypermedia systems, but, as shown by Grønback and Trigg, it is insufficient "to model the ways people interpret link direction in practice [11]."

2.6 Observations

Several observations about the overall characteristics of the previously described hypertext systems:

1. Scalability is often ignored.

   Dexter and Xanadu require links and other system information be completely available – an unrealistic expectation for distributed systems. The World Wide Web's association of documents with location limits the ability to relocate documents.

2. No consensus on typing mechanisms to associate characteristics and invariants with nodes and links. Typing mechanisms include:

   - *no typing*

     Xanadu provides no node types. The lack of a node type means that there is no mechanism to associate attributes tightly with a document.

   - *single value*

     The WWW utilizes a single value, a relation name, to express link types. Single value types are usually selected from a standard set supplied by the system or maintained by some authority. For example, WWW relationship names are registered by the W3 Consortium. A mechanism to allow individual users to designate a new value as a type is sometimes provided, but such a mechanism is usually limited.

Single value types generally do not allow partial knowledge of a particular type: either a type is recognized or it is not. All single value relationships must be made explicit by some entity; there are no implied relationships between values.

- *hierarchical types*

  Aquanet nodes and links are instances of a specified type in a type hierarchy. Hierarchical types provide a mechanism to relate a new type to prior types through the placement of the new type in the inheritance tree. Careful choices of inheritance allow a new type to reveal details about its characteristics and capabilities.

  One limitation of hierarchical types is the difficulty in selecting a position in the hierarchy to add new types. It is sometimes desirable to place a new type at multiple locations in hierarchy.

- *attribute-value pairs*

  The World Wide Web and Dexter provide an attribute-value mechanism for nodes and links. Attribute-value pairs, while not strictly a typing mechanism, utilize a set of attributes to describe node and link characteristics. These characteristics are expressed by associating attribute names with values.

  As with singular values, attribute value pairs must be limited to a standard set. A user can relate a new "type" to prior types by appending a new attribute to existing, well understood attributes. Unfortunately, most attribute-value systems do not provide a mechanism to prevent attribute naming conflicts. Further, individual attribute values suffer the same recognition problems as single value (either recognized or not recognized).

Each of these typing mechanisms has limitations. No typing prevents the exposure of document invariants. Single value mechanisms limit the expressive capabilities of individual users. Hierarchical types limit type associations by requiring a single position in the hierarchy. Attribute-value pairs have naming conflicts which limit expressive capability. These limitations emphasize the need for an extensible typing mechanism.

3. No consensus on node substructure exposure. Substructure exposure mechanisms include:

   - *no substructure exposure*

     The object is completely opaque with no generalizable mechanisms to allow link associations. No examined hypertext provided such substructure exposure, but Aquanet only allows linking at the granularity of individual nodes. A lack of substructure exposure limits linking capability – node substructure can not be linked.

   - *entire content exposure*

     Node contents are completely exposed for linking – but not necessarily with any content invariants. Xanadu nodes expose their complete structure with no invariants, with a resulting linking schema which depends on character matching.

   - *arbitrary anchors*

     Anchors provide a mechanism by which links can "reach inside" nodes and "hold" onto node substructure. WWW and Dexter nodes provide arbitrarily named "anchors" with no mechanism to specify context or semantics. Anchors provide invariants, allowing node contents to change while providing a consistent interface. However, the lack of a mechanism to specify anchor characteristics limits anchors to be utilized as arbitrary identifiers of substructure regions.

   - *syntactic anchors*

Anchors explicitly associated with a particular HTML syntactic structure is a suggested addition to HTML in the current version 3.0 draft [19]. It is not clear if the present proposal allows for expressing semantic content.

A link's ability to reference node structure is limited by the mechanisms provided by the nodes being linked. If nodes expose substructure invariants, either through anchors or some other mechanism, then a link can "hold" onto those invariants across mutations. It is unclear which mechanism is the best method by which nodes should expose their contents for linking. Limited anchor capabilities suggest the need for more formal structures.

4. No consensus on link endpoint capabilities. Link endpoint capabilities include:

- *no substructure linking*

  No substructure linking implies that link endpoints connect at the granularity of nodes. As an example, Aquanet links relate entire objects, not object substructure. A lack of substructure linking limits the power of links to express relationships between nodes which involve substructure.

- *substructure linking*

  The WWW and Dexter links utilize "anchors" for substructure linking. The WWW links use statically specified link endpoints. Dexter provides dynamic determination of link endpoints through the use of specifiers. Substructure linking is limited by the exposure of node substructures.

- *computed linking*

  Links may utilize computations on nodes for linking. Such approaches are useful when the item to be linked is not exposed by the node as an anchor or equivalent invariant structure. One example is Xanadu's mechanism of linking to nodes through the use of a computation involving invariant characters – presumably some form of character matching. Equivalent linking schemas might utilize character offsets or word counting to specify the endpoint of a link. The problem with computations is that they fail in the presence of mutable objects. This is particularly true for nodes which do not expose characteristics or invariants through some typing mechanism.

Clearly, a powerful link endpoint mechanism would utilize exposed substructure invariants, yet provide the capability to utilize computations on nodes.

5. No consensus on minimum link characteristics and capabilities, including:

   - *multi-dimensional links*

     Xanadu, Aquanet and Dexter links can be relate more than two entities. The WWW restricts links to two-ended structures.

   - *directionality*

     Xanadu expects a distinguishable FROMSET and TOSET. Dexter, in contrast, marks individual endpoints as either TO, FROM, BIDIRECT or NONE. WWW has implicit directionality from the markup in a document. Aquanet does not have link directionality.

   - *presentations*

     Dexter links provide a "presentation specifier" with both the link and each endpoint. Aquanet utilizes a graphical appearance specification associated with node and link types to designate the presentation of Aquanet objects. The WWW utilizes HTML as a markup language to describe presentations.

   - *independent links*

     Aquanet and Dexter links are independent hypertext entities. The WWW and Xanadu require that links be embedded in a hypertext node.

   - *named endpoints*

     All Aquanet endpoints are named. Some WWW and Xanadu links are named. Dexter does not name its link specifiers.

Clearly, hypertext systems employ a variety of different link characteristics. It is not clear which mechanisms are absolutely necessary.

We will utilize these insights to provide a reference for discussing the attributes and implementation of a global information infrastructure linking mechanism: Information Mesh Links.

2.7 Summary

In this examination of nodes, links and system attributes, we have described how node attributes support linking, how link relationships are exposed to the overall system, and how particular system requirements impact link capabilities. In particular, we observed the overall lack of consensus on the issues of node and link typing, substructure exposure, endpoint capability and overall link characteristics. Associated with these observations, we noted the need for a scalable hypertext system providing extensible typing and a formal mechanism for substructure exposure. We described the need to determine minimum link capabilities. Further, we discussed the need for a powerful endpoint mechanism utilizing exposed substructure invariants yet providing the capacity to utilize computations on nodes.

Chapter 3

Information Mesh Project

The Information Mesh Project represents a new paradigm for networked systems which supports the vision of widespread information sharing and structuring. The central idea of the Information Mesh is that the network exists primarily to maintain relationships among nodes of information. The fundamental activity of network applications thus becomes constructing, manipulating and using these relationships.

The implementation of this vision has been centered around the notion of supporting networked *Mesh objects* interconnected by *links*. The overall goal is to understand the minimum set of information services necessary to support such a model and push them into the networking infrastructure. The result should shield applications from having to manipulate transport level protocols.

Work for this project has resulted in the creation of a *Mesh kernel* and *Mesh object system*. The Mesh kernel provides information naming, discovery and relocation. The Mesh object system utilizes the notion of *roles* to provide flexible, evolvable objects in the Mesh. Roles provide an extensive typing mechanism to describe object behavior (*actions*) and object structure (*parts*). *Mesh links*, a mechanism to express relationships between Mesh objects, are described in Chapter 5.

In this chapter, we describe the overall goals, constraints and requirements for the Information Mesh. We describe the Mesh kernel and Mesh object system.

3.1 Goals

The Information Age has created a need to manipulate a vast and ever increasing amount of data. As an example, consider the Internet: the traffic related to information manipulation has increased tremendously the past few years [1]. Indeed, the explosive growth and success of the World Wide Web, Gopher and other Internet information navigators, combined with the recent commercialization of the Internet, can only lead to increasing growth. Corresponding to this growth has been an increasing awareness that current information manipulation tools are inadequate to an already vast information base.

The Information Mesh attempts to address the problem of inadequate information management tools by providing a networking substrate in which information manipulation is an attribute of the network, not the individual application. The hope is that "much as traditional applications utilize a database system, the Mesh will become the primitive abstraction around which applications are built [8]."

The overall vision of the Information Mesh Project is to provide a long-lived global architecture for networked-based information reference, manipulation and access as a ubiquitous substrate for distributed and network applications and domain-specific knowledge bases. The implementation of this vision is expected to contain objects interconnected by relationships or links in a universal and long-lived information base.

3.2 Constraints

The constraints to meet the vision of a Mesh of objects can be summarized as universality, ubiquity, heterogeneity, longevity, evolvability and resiliency.

- *Universality*

  The Information Mesh vision of "a single model for information identification, location and access as a substrate for distributed system and applications [22]."

implies that the Mesh must be universal; it must provide agreement on referencing objects and do so in a highly scalable manner.

- *Ubiquity*

  The Information Mesh must support "network-based applications accessing information that is distributed both physically through the net and administratively across regions of differing management policies [22]."

- *Heterogeneity*

  The Information Mesh should be prepared for changes in communications media, transport protocols and networked-applications. It must support a broad set of protocols and applications, both those implemented and likely to be implemented.

- *Longevity*

  The Mesh must support long-lived information; it can not require that information be reformatted and it must support both old and new formats. Objects must be constructed in a manner that realizes that the same object may exist for hundreds of years.

- *Evolvability*

  The Mesh must be able to provide for changing semantics, syntax, structures and utilization of information. The Mesh must be able to provide capabilities for information to be utilized in new and unexpected forms. The Mesh must support new network services. It must provide for information moving both in physical location and ownership.

  Mesh objects must be made available in a manner that realizes that they may change location, ownership and behavior. Thus, we must ensure that Mesh mechanisms do not expect an object to remain constant.

- *Resiliency*

The Mesh must provide resiliency in the face of unreliability. The Mesh will exist in many situations of unreliability where it will be unable to locate or access information. Thus, the Mesh must be designed from the start to provide mechanisms to deal with unavailability.

3.3 Implementation Requirements

The goals and constraints of the Information Mesh imply several implementation requirements: minimal agreement, minimal coordination, and flexibility.

- *Minimum Agreement*

  The need for minimal agreement comes from the pragmatic understanding that "we can not depend on any universal agreement on issues like a best way to find information, the internal structure of information or how information is internally manipulated by programs [24]." Thus we must minimize the requirements imposed on Mesh entities.

- *Minimum Coordination*

  The need for minimum coordination of information flows from the need for resilience and ubiquity. The Mesh needs to be highly scalable with diverse mechanisms to find, represent and manipulate information. These goals are best met if the overall coordination between these capabilities – and any other core Information Mesh services – are designed to minimize the required coordination.

- *Flexibility*

  The need for flexibility is a result of the need for heterogeneity, longevity and evolvability. The Mesh needs to support a wide set of global information architectures. Further, the Information Mesh should be "flexible enough to encompass new network services as they evolve. It should also support a broad set of expectations from applications as well as administrative controls.[22]"

These constraints imply that the Mesh must be implemented with the constraints of minimal universality, but with an eye towards minimum coordination and enormous flexibility. Thus, we must minimize the set of required Mesh functionality while still providing the sufficient flexibility to build a wide range of services on top of the Mesh.

Note that the Information Mesh does not directly deal with security and privacy issues except where they affect design decisions.

3.4 Information Mesh Kernel

The first step in realizing the Information Mesh Project was the implementation of the Information Mesh kernel [24]. The Information Mesh kernel addresses several of the concerns raised by the Project. In particular, the kernel provides information naming, discovery and relocation as a powerful and evolvable component of the Mesh.

The Information Mesh kernel's naming is provided through the use of globally unique identifiers described as *points*. Information about these points are stored in sets of attribute-value pairs called *factoids*. Information is located through a flexible and evolvable locating mechanism that utilizes meta-information about where points have been seen or discussed in the Mesh. Finally, the kernel provides a generic procedure dispatch mechanism.

The Information Mesh kernel ensures minimum coordination by ensuring that information identification (points) is decoupled from location and retrieval. In particular, points contain at most hints about location. The overall kernel is designed to have minimum constraints on data representation and location to provide a flexible information infrastructure.

3.5 Information Mesh Object System

The Information Mesh object system[23] provides the Mesh with a powerful means to create and utilize *Mesh objects* – the chief feature of which is the capability of objects to *play* a variety of *roles*. Roles describe object behavior by specifying *actions*, *parts* and *makers*. *Implementations* provide objects with a concrete representation of a role capability.

3.5.1 Mesh Objects

Mesh objects are identified through the use of *oids*. Oids provide a naming scheme that ensures that objects can be uniquely specified throughout the global network. Our current implementation utilizes the kernel's *ports*, but we eventually expect to provide a more general identification mechanism, such as URNs [21].

Object behavior is built around the notion of a *role*. A role is a specification of an abstract behavior and structure, similar to an object class. An object *plays* a particular role if it behaves in the manner described by that role. To understand the interaction of *roles* and *plays*, imagine how an individual plays several roles in life such as parent, teacher, leader, follower, etc. This notion captures the key notion that objects can play multiple roles and that the roles played can change or evolve through time. Roles are further described in Section 3.5.2.

All Mesh objects play the *object-role*. The object-role provides a starting point for all dialogs with Information Mesh objects. Since all Mesh objects must play the object-role, we are guaranteed that the required object-role actions are answerable by any Mesh object. Objects playing the object-role can answer questions about which roles they can play, allow the addition of new roles to play, and describe the implementation objects for a role played by the object. The object-role's actions and parts are detailed in Appendix 7.

3.5.2 Roles

Roles are composed of *actions*, *parts* and *makers*. Actions specify the abstract behavior of a role. Parts specify the static abstract structure of a role. Makers specify the abstract mechanisms necessary for creation. Taken together, these three characteristics (actions, parts and makers) constitute the necessary characteristics for an object to *play* a particular role. [1] We will examine *actions* and *parts* in more detail in Sections 3.5.4 and 3.5.5. Since *makers* are not important to this discussion, we will not investigate them further.

Roles are arranged into an inheritance hierarchy such that if an object plays a particular role, it also plays all of that role's super roles. The inheritance rules for roles are based on the hierarchy rules present in the Common Lisp Object System specification [14]. The single root of all role inheritance is the *object-role* which provides role playing capabilities as described in Section 3.5.1.

Roles serve as an extensible object typing mechanism. Roles provide invariants in object interface – objects playing a role agree to perform the actions, parts and makers specified by the role. Furthermore, role inheritance provides user extensible typing. That is, a user specified role's position in the hierarchy determines a subset of the user-specified role's features (because role inheritance specifies that if an object plays a particular role, it plays all of that role's super roles). Thus, one can determine a subset of a user-specified role's features from its position in the role hierarchy.

Roles provide flexibility and evolvability through the ability of objects to play multiple roles. Objects can play multiple roles simultaneously or even different roles at different times; the nature of an object can evolve in time by making the same object play new roles through its existence. Thus, newer applications can have access to old objects via their old roles at the same time that newer applications can access the same information by using newer roles [23].

Roles are first class Mesh objects; a role is a Mesh object which describes the actions, parts and makers necessary for an object to play a particular role. Mesh objects which provide such services are said to be playing the *role-role*.

3.5.3 Implementations

Implementations provide Mesh objects with the ability to 'play' a role by describing a concrete representation of a particular role's actions, parts and makers. Mesh object

[1] We use 'role' and 'plays' instead of 'class' and 'instance' to capture the notion that as objects evolve through time they may exhibit diverse natures by playing a variety of roles.

may utilize multiple implementations. It is the job of implementations to actually figure out how to implement new nature on old objects.

Implementations are independent of roles. In theory, every object playing a particular role could utilize a different implementation. Alternatively, every object implementing a role could utilize the same implementation. In practice, it is likely that implementations will be packaged and distributed by a variety of information providers. Implementations provide an implementation inheritance mechanism such that if a particular implementation doesn't provide a description of some concrete role capability, the super implementations are examined for the capability.

Implementations are first class Mesh objects; an implementation is a Mesh object containing concrete *methods* for actions, parts and makers. Presently methods are represented using portable lisp code.

3.5.4 Actions

Actions specify role behavior; they specify the form of interactions with any object playing a role. In this manner, actions specify the interface to methods. For all roles played, the following actions are special in that they are always answerable by an object for whatever role they are asked:

(**actions-supported** *object role*) Required for all roles

Returns the list of actions that the object supports when playing the role in which asked.

(**supports-action?** *object role action-name*) Required for all roles

Returns true if the object supports an action named *action-name* when playing the role in which asked. Returns false otherwise.

Note that roles allow optional actions which are not required to be implemented. Hence, the answer to 'supports-action?' must be true for *required actions* and may be true for *optional actions*. The result for optional actions depends on both the implementation and the particulars of the object of which the question is asked.

Optional actions are utilized for a variety of reasons. One compelling reason is to allow slightly different capabilities among implementations of roles, for instance, an implementation of a role which allows object mutations and an implementation which does not allow object mutations. Such a mechanism is particularly useful for inherited roles where it is not always desirable to permit super role mutable actions. Optional actions also allow objects to provide certain actions only at certain times.

3.5.5 Parts

Parts expose the abstract structure of an object playing a role; they specify an interface to object structure. Parts provide an ability to expose invariants in terms of object structure. Parts are divided into two portions: *part-names* and *part instances*. Part-names are described by the role. Part instances are created and utilized by Mesh objects and exposed through several universal actions. Part instances can be specified through the use of a part-name and *selector*.

Part-names are relatively static structure names. In the original object implementation, part-names are simply identifiers specified by a role. All possible part names for a particular role can be statically determined.

Part-names may be either *required* or *optional*. Objects must implement the parts associated with required part-names. As with actions, the existence of part names is answerable by all Mesh objects regardless of the role. The 'parts-supported' action enumerates the currently available part-names and the 'supports-part?' action determines the existence of a particular part-name.

(**parts-supported** *object role*) Required for all roles

Returns the list of part-names that the object supports when playing the role in which asked.

(**supports-part?** *object role part-name*) Required for all roles

Returns true if the object contains a part-name when playing the role in which asked. Returns false otherwise. Must return true for all required parts and may be true for optional parts.

Associated with each *part name* are *part instances*. Part instances are the Mesh mechanism to expose a dynamic model of object substructure. Part instances may overlap or even contain one another; they can be dynamically created and destroyed. It is important to note that part instances do not have to form an enumerable set. Thus, it may not be possible to know all selectors for a particular part-name. Part instance utilization is determined by the object which contains them.

Part instance existence can be determined through the utilization of the 'has-part-instance?' action. Note that there is no 'supported-part-instances' action to enumerate the part instance selectors (because of the potential innumerable nature of part instances).

(**has-part-instance?** *object role part-name selector*) Required for all roles

Returns true if the object contains an instance of the part specified for the given selector.

Selection of part instances is largely provided by specifying a part-name and a selector. A *selector* could be a range, words in a document object, etc., but this is not exposed by the role. Selectors always specify a particular instance, but part instances can be constructed in a manner such that their selection indicates the utilization of several part instances. Thus, a part instance can be a set of instances. Regardless, the original Mesh object system does not provide a mechanism to expose the contents of part instances. We will examine an enhancements to provide such capability in Section 4.3.

Another limitation of the original object system is the limited capability to expose the selectors available for part instances. There is not, for instance, a mechanism to enumerate (if possible) the set of instances for a particular part name. Nor is there a mechanism to statically expose part instance selector criteria in the role declaration. One result of this limitation is that there is no mechanism to declare that a particular part-name can have only one part instance associated with it. Indeed, there is no mechanism to expose part instances available for any part-name, nor to specify the range of potential selectors. This is not entirely surprising as the part instance set – and valid selectors – might be large, arbitrary or unspecifiable.

In summary, part support is achieved through three mechanisms: the declaration of part-names in the role, the runtime determination of optional part name existence and the ability to determine the existence of a particular part-instance through the 'has-part-instance?' action. Not initially associated with parts is the capability for part content manipulation or part instance selection exposure.

3.6 Summary

The overall vision of the Information Mesh Project is a long-lived global architecture for network-based information reference, manipulation and access. One component of this vision is the notion of Mesh objects interconnected by links. The constraints to meet this vision can be summarized as universality, ubiquity, heterogeneity, longevity, evolvability and resiliency. The Information Mesh requirements for base mesh capabilities are minimum agreement, minimum coordination and maximum flexibility.

The Information Mesh object system provides a means to create and utilize Mesh objects. Mesh objects are identified through the use of oids. Mesh object behavior is built around the notion of a role. A role is an abstract specification for object behavior. Roles describe abstract functionality (actions) and abstract structure (parts). An object is said to "play" a role if it behaves in the manner described by that role. Roles serve as an extensible object typing mechanism–providing flexibility and evolvability to Mesh objects.

Mesh objects expose their substructure through the utilization of parts. Parts are composed of part-names and part instances. Part-names are static names for object structure. Part instances are the Mesh mechanism to expose a dynamic model of object substructure. Selection of part instances is provided by specifying a part-name and selector. The original Mesh object system does not provide a mechanism to expose the contents of part instances, nor a mechanisms to expose selector characteristics for a part.

Note that unfulfilled from the original vision of the Information Mesh is a link mechanism to describe relationships among Mesh objects. In the next chapter, we will examine modifications to Mesh objects to better support Mesh links.

Chapter 4

Mesh Objects as Linkable Nodes

The Information Mesh vision of objects interconnected by links requires an examination of Mesh objects as nodes for linking in the Mesh. In this chapter, we examine Mesh objects using the criteria described in Chapter 2. Namely, we examine Mesh capability to provide:

- naming
- typing
- substructure interface
- composite objects

For capabilities already provided by the Mesh, we review the implementation and describe any limitations or necessary enhancements. For capabilities not provided by the Mesh, we describe implementation options, their associated limitations and the chosen implementation. Finally, we describe several examples of hypertext nodes implemented utilizing Mesh objects and the described enhancements. Note that versioning, which is important but not central to our overall discussion of Mesh links, is described in Appendix 8

4.1 Naming

Nodes in a hypertext system need to be named or distinguished in some manner. As shown in Section 3.5, the Information Mesh ensures that all Mesh objects are associated with a globally unique object identifier or *oid* which provides object identification and naming. Oids contain no semantics about object capability, location, versioning or typing.

Note that the Mesh does not have a mechanism similar to Dexter's Resolver function (described in Section 2.5.4) to produce oids from an object specification. However, such a mechanism could be implemented as a Mesh service. We consider such a mechanism to be outside the scope of Mesh links.

4.2 Typing

Node typing provides a mechanism to describe node semantics and invariants. Chapter 2 detailed a variety of hypertext node typing mechanisms including: no typing, single value typing, hierarchal types and attribute-value pairs. This examination made clear the need for an extensible typing mechanism.

The Information Mesh object system utilizes roles as its typing mechanism. Roles provide a powerful typing mechanism sufficient for Mesh objects to function as hypertext nodes. In particular, roles provide object invariants and user extensible typing. Role flexibility was previously described in Section 3.5.2. The usefulness of roles as a node typing mechanism is strengthened by the observation that roles can support all of the typing models described in Chapter 2. More specifically, single value and attribute-value typing can be provided through object parts and hierarchical types can be provided through role inheritance.

4.3 Substructure Interface

As described in Chapter 2, links are limited by the substructure interface provided by nodes. For example, Dexter links are limited by the anchors exposed by Dexter components. Substructure interfaces provide invariants that links can hold onto across node modifications. The lack of substructure exposure or invariants clearly limits link capability.

In the Information Mesh object system, object substructure is formalized into *parts*. Parts provide a mechanism to expose object structure in a manner similar to hypertext node anchors, but in a more systematic and generalizable manner. The Mesh object system provides the capability to declare part names, determine part-name presence through 'has-part?' and 'parts-supported', and determine the existence of part instances through the 'has-part-instance?' action.

Selector exposure and content manipulation were not provided in the original object system implementation. We describe modifications to provide these capabilities.

4.3.1 Selector Exposure

The Mesh object system utilizes selectors to specify part instances and determine their existence. However, there is no mechanism to specify selector characteristics in a role declaration of a part-name. We describe a mechanism utilizing role declarations to specify selector characteristics and specialized actions which can utilize such declarations.

Role declaration of part selector characteristics allows one to describe part instance capabilities for a specified part-name. Thus, role declarations of selector characteristics constrain the set of possible part selectors for a specified part-name. We describe selector characteristics by providing a *selector type* with each part-name in a role declaration. We provide the following selector types:

unspecified characteristic of selectors is unspecified

unary-of one part instance (part selector is ignored)

set-of part instances are grouped into one unordered
(no selection necessary)

named-of part instances are named with identifiers determinable at run-time.

ordered-of part instances are ordered and determinable at run-time.

The declaration of part selector types allows the use of specialized actions for certain selector types. In particular, parts utilizing a 'named-of' or 'ordered-of' selector type can (optionally) provide run time capabilities to create and remove part instances. 'Unary-of' and 'set-of' selector types ignore the selector for any part instance manipulation actions, such as the content manipulation actions described in Section 4.3.2.

Part-instance-names ('named-of' actions)

(**part-instance-names** *object role part-name*) Optional for all roles

Enumerates the selectors for part instances associated with the specified part name. Returns false if there are no part instances associated with part-name. Requires that the part-name be declared in the role as utilizing a 'named-set-of' selector type.

(**add-named-part-instance!** *object role part-name instance-name contents*) Optional for all roles

Allows one to add a named part instance to the specified part-name. Requires that part-name be declared as utilizing a 'named-set-of' selector type.

(**remove-named-part-instance!** *object role part-name instance-name*) Optional for all roles

Allows one to remove a specified part-instance. Requires that the specified part-name be declared in the role as utilizing a 'named-set-of' selector.

Parts declared with a 'named-of' selector type can be utilized as both an anchoring mechanism (named selectors serve as anchor identifiers and instance contents serve as anchor values) and attribute-value pairs (named selectors serve as attribute names and instance contents serve as values).

Part-instance-range (‘ordered-of’ actions)

(**part-instance-range** *object role part-name*) Optional for all roles

Returns range of part instances in integers. Requires that part-name be declared in the role as utilizing an ‘ordered-of’ selector type.

(**set-part-instance-range!** *object role part-name low high*) Optional for all roles

Sets range of part instances. Any instances outside of range are removed. Requires that part-name be declared in the role as utilizing an ‘ordered-of’ selector type.

(**set-ranged-part-instance!** *object role part-name value contents*) Optional for all roles

Sets a particular value in range to contents. Requires that part-name be declared in the role as utilizing an ‘ordered-of’ selector type.

Part instances do not necessarily form a discrete set. Thus, while we can always determine existence from ‘has-part?’, there is no guarantee that we can provide a selector type more specific than ‘unspecified’.

In summary, the selector type mechanism provides the ability to expose a minimum set of selector characteristics. We expect it will be necessary to provide a variety of additional selector types and actions.

4.3.2 Content Manipulation

In the original Mesh object system, parts expose the abstract structure of an object, but there is no generalizable mechanism to manipulate part content in a manner similar to “slots” in some object systems.

Part instance content extraction is provided by the optional action, ‘extract-contents’. Part instance modification is provided by the optional action, ‘set-part-instance-value!’.

(**extract-part-instance** *object role part-name selector*) Optional for all Roles

Returns contents of a specified part-instance.

(**set-part-instance-value!** *object role part-name selector value*) Optional for all Roles

Allows the setting of a specified part-instance

Regarding content manipulation, two items are notable. First, implementors may choose to provide only specific part instance manipulation capabilities, for instance, if part contents are not to be exposed for security reasons. Second, the setting and extracting of values requires a mechanism to describe the nature or type value of a particular part instance. In the following link role discussion, we will allow the declaration of part "types" to describe the nature of the part instance. [1]

4.3.3 Substructure Interface Summary

A substructure interface, while not strictly necessary for Mesh linking, enhances the capability of Mesh links. We examine the result of not providing certain substructure capabilities.

- *no part instances*

  A Mesh object may choose not to expose any substructure – with a resulting reduction in link capability. For example, if an object does not provide part instances then one can only link to the whole Mesh object. In this example, the lack of part instances limits the expressible relationships because no object substructure is exposed.

- *no selector exposure*

  Not exposing a criterion for reasonable selectors at the Mesh level reduces the capability of entities examining an object to determine a suitable link. Again, such capability is not strictly necessary but providing selector criterion exposes object semantics.

- *no general part manipulation*

  Not providing part content manipulation limits the ability of someone unaware of an object's semantics. Otherwise, one could examine an object and its part

[1] In our present system, these part types are ignored.

content to make some determination about part semantics. Again, part content manipulation is not strictly necessary but providing content extraction and mutation capability increases the exposure of substructure semantics.

In summary, none of the substructure interface capabilities are strictly necessary or required. Mesh objects may choose to provide only a subset of these substructure interfaces. However, the exclusion of substructure capabilities by Mesh objects limits the capability of Mesh links.

4.4 Composites

Composites provide the ability to combine Mesh objects into a single composite object – essentially, a collection of Mesh objects maintained by a specific object. Composite objects express a requires relationship, a statement that a particular set of objects playing a specified role are "required" for the composite to behave in its intended manner. We argue that composites can only be achieved by pushing a notion of composites into the Mesh.

4.4.1 Need

The motivation for a composite structure has been understood by the hypertext community for quite some time. In his Seven Issues paper [13], F. Halasz suggests: "The basic hypertext model lacks a composition mechanism, i.e., a way of representing and dealing with groups of nodes and links as unique entities separate from their components [13]." Further, the notion of a composite component is formalized in the Dexter Model of Hypermedia System[12] which specifies composite components as a directed graph of components. Thus, composite objects can be justified by the need to provide composite objects at the Mesh level.

A further motivation for composite objects is the nature of the Information Mesh itself. As a distributed system, the Information Mesh may be unable to provide complete information about an entity; such capability is infeasible in the vast domain of the Information Mesh. This lack of system-wide knowledge implies that entity knowledge must be maintained by the entity itself. Thus, a composite object would allow one to expose objects at a Mesh level as composites. An explicit specification of composite objects provides system-level capability and awareness when moving, copying or relocating objects. With such an explicit mechanism, the system can ensure (within policy constraints) that if a composite object is moved, all its associated objects can be moved as well. This is particularly useful if an object is being moved to where it can not communicate with other objects – allowing the system to ensure that necessary objects are moved as well.

Thus, composites allow the "wrapping" of objects into a composite – allowing the composite to expose a new interface. This is particularly useful if one needs a new interface to an object, but can not make the object play a new role. In a related manner, composites allow the "bundling" of independent Mesh links with an object. The need and mechanism for "bundling" Mesh links is described in further detail in Section 5.2.3.

4.4.2 Composite Options

There are several possible implementations of Composite objects. Security and availability considerations limit our implementation options. The main issue is whether composites can be implemented using the basic Mesh capabilities or whether composites will require additional Mesh capabilities.

- *Requires Link*

  In theory, all relationships between Mesh entities could be expressed using Mesh links. One could imagine creating a "requires" link to express that a particular Mesh object requires another set of Mesh objects.

  Unfortunately, independent links can not describe intrinsic characteristics of Mesh objects because the independent link and the object could become "separated" in the Mesh. The reason for this is that there is no implementable Mesh mechanism to determine if all possible links to an object have been examined or determined. [2] Thus, links can not be utilized to create composite objects.

- *Composite-Role*

  Under this implementation, composite objects play the *composite-role*. When a Mesh object plays the composite-role, it must answer requires questions for all other playable roles for that object. This approach causes problems because the require-role will have to answer questions about roles it doesn't play. Since there is no internal mechanism to allow different roles to share information (particularly between different implementations), this approach requires significant modifications to the Mesh object architecture.

- *Monolithic object*

  Monolithic objects bundle all required objects into a single object – wrapping oids via some as yet unspecified mechanism and exposing the embedded objects through some interface. The advantage of this approach is that previously multiple objects are now accessible through a single, monolithic object.

  Unfortunately, security and practicality concerns prevent utilizing this mechanism on all objects. First, one may not have access permissions to all objects which need to be bound into composite object. That is, some objects may not allow copying or movement into a new composite object. Further, one might desire a composite object without the requirement of moving all objects into one monolithic object. Finally, this mechanism doesn't work if an object is a component of more than one composite object.

- *Complete Object Awareness*

  Another implementation option is to require that every object maintain a list of all composite objects of which it is a member: contained or containing. This mechanism ensures that every object is completely aware of the composite relationships of which it is a member.

[2] The notion of "embedded" links to describe intrinsic Mesh object characteristics will be explored in Section 5.2.3.

There are several problems with this approach. First, it would be necessary that all objects maintain a store describing all composites of which it is a member. This would require that all objects be mutable and maintain permissions for modifying composite attributes. For public documents, such a need could quickly drive up the cost of maintaining the object as a public entity. Second, it would be necessary to synchronize all copies of an object to ensure linking to one object is exposed by all copies.

- *Special "Requires" Action*

  This approach pushes the notion of composites into the Mesh as a basic Mesh capability similar to 'supports-action?' and 'parts-supported'. Thus, every role must support an action which returns the objects "required" by that role. The main problem with this approach is that it entails adding additional capability to the overall Mesh.

4.4.3 Composite Implementation

Our composite implementation is realized by pushing the notion of "requires" into the basic Mesh capabilities through the optional action, 'get-required-objects'. The absence of 'get-required-objects' from a particular role implies that the object does not require any other objects when playing that role.

(**get-required-objects** *object role*) Optional for all roles

Returns the set of oids necessary for the object to play the specified role. Associated with each oid is the role or roles required from that oid.

Note that 'get-required-objects' does not produce the closure of required objects and roles; 'get-required-objects' returns only the objects and roles directly required by the specified object playing the specified role. The only exception occurs when the same object is playing or supporting multiple roles, there is an interaction between the roles and there are different notions of composition. Under such conditions, the result of invoking 'get-required-objects' contains the required components of all roles.

While a composite object conceptually "contains" other objects, the contained objects are not aware of their inclusion in a composite object. Thus, composites can specify any set of objects as being required without the need to notify the contained nodes. This assures privacy regarding objects contained in one's composite, but it also makes the determination of all composites containing a particular object impossible. Further, composites can provide no guarantees about the "contained" objects; a "contained" object may change in an unexpected manner.

4.5 Node Examples

Mesh objects can provide the node capabilities of the examined hypertext system nodes. As a demonstration, we provide role definitions of various hypertext system nodes.

4.5.1 Dexter Component Role

As described in Section 2.5.3, Dexter components are composed of a base component together with component information providing unique identification, anchoring, presentation specification and attribute-value pairs.

For our Dexter Component Role, we utilize oids to provide unique identification and roles for component characteristics. Anchoring, presentation specification and attribute-value pairs are provided through parts utilizing a 'named-set-of' selector type. Role actions to expose attributes and determine Dexter links to the component are provided. Part content manipulation is provided by the generic Mesh part manipulation capabilities described in Section 4.3.2.

Inherits from: **Object-role**

Actions

(**all-attributes** *object*) Required

Returns the set of all attribute-value pairs.

(**links-to** *object*) Required

Returns the Dexter links to the Dexter component. A mechanism to provide this functionality will be described in Section 5.2.3.

(**links-to-anchor** *object anchor-name*) Required

Returns the Dexter links to the specified Dexter anchor.

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**anchors** : *named-set-of unspecified-type*) Required

Named anchors associated with component.

(**attribute-value** : *named-set-of unspecified-type*) Required

Pairs of attributes which describe the Dexter component.

(**presentation-specifier** : *unary-of value*) Required

The value describes the presentation of the component.

4.5.2 Aquanet Node Role

As described in Section 2.4, Aquanet node slots are a named set of contents restricted to primitive datatypes such as text, images, numbers, strings, etc.

Inherits from **Object-role**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**slot** : *named-of unspecified-type*) Required

Contains slots of an Aquanet node.

4.5.3 Aquanet Statement Role

Aquanet statement nodes are utilized by Aquanet argument relations to describe the grounds, rationale or conclusion of an argument (as described in Section 2.4). Aquanet statement nodes are simple Aquanet nodes with the additional requirement that they contain a statement slot.

For our Aquanet Statement Role, we create a role which contains a single statement part (selector type is unary) and inherits from the general Aquanet node.

Inherits from **Aquanet-node-role**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**statement** : *unary-of text*) Required

Contains text of statement node.

4.5.4 World Wide Web HTML Document Role

As described in Section 2.3, World Wide Web HTML documents provide marked up text with content linking provided by anchors. An anchor HREF specifies the beginning of a link. Anchor names specify the potential targets of a link.

Inherits from **Object-role**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**html-text** : *unary-of text*) Required

HTML text of WWW document.

(**anchor-names** : *named-of text*) Required

Named destinations in WWW document. Part content is text contained in region marked by anchor name.

(**anchor-hrefs** : *ordered-of text*) Required

Ordered list of HREFs in WWW document. Part content is text contained in region marked by anchor reference.

4.6 Summary

Hypertext nodes require naming, typing, substructure interface, and composite objects to support better linking. Naming and typing are provided by the Information Mesh object system which provides naming through the use of oids and typing through the use of roles. Substructure interface is provided by parts, which have been enhanced to provide exposure of part selector characteristics, specialized actions for certain selector types and a mechanism for the manipulation of part instance content. These part enhancements, while not strictly necessary, enhance the overall capability of Mesh links.

Composite objects are provided by pushing the notion of "requires" into basic Mesh capabilities. Composite objects are motivated by the need to express composites at the Mesh level, the ability express fundamental interrelationships between Mesh objects explicitly, and the ability to "wrap" Mesh objects into a single Mesh object. Composite relationships are one-way; composites can specify an objects as "required" without any need to notify the "contained" objects.

Chapter 5

Link Architecture

The Information Mesh project has a vision of *Mesh links* for expressing relationships among objects in a global, *information mesh* of objects: "A link, as the expression of a relationship, is composed of a *kind*, identifying the nature of the relationship, and *descriptors* identifying the objects involved in the relationship, and which parts of the objects are indicated. A descriptor can identify all of an object, some aspect of an object, or some component of any object [7]." Thus, Mesh links need to be exposed to the Mesh in some manner.

As an inherent component of the Information Mesh, Mesh links need to provide the capabilities expected of all Mesh entities – clearly defining minimum requirements in a manner that recognizes unavailability and provides flexibility in both implementation and evolution. For Mesh links, the overall goal is to allow a wide variety of linking capabilities to be built on top of the base Mesh link implementation. Links need to provide and utilize exposed semantics.

In this chapter, we will examine a *Mesh link* architecture. We examine link attributes in the context of the Information Mesh and the hypertext link issues examined in Chapter 2, including: link utilization, link relationships, link independence and endpoint capabilities. From this examination, we describe a minimum Mesh link implementation which either fulfills the examined attributes or provides sufficient flexibility for their adaptation in a more specific Mesh link. Finally, we provide examples of Mesh links built on top of the minimum Mesh link mechanism.

5.1 Link Attributes

Mesh links should be sufficiently flexible to provide the link capabilities described in Chapter 2:

- *Link utilization*

  Mesh links are the primary mechanism for expressing object relationships in the Mesh. "Links are an inherent part of the Information Mesh, expressing relationships among nodes [8]." Mesh links should further be able to describe relationships between other Mesh links. Thus, Mesh links are the fundamental mechanism for expressing relationships in the Information Mesh.

  Mesh links need the capability to express relationships between Mesh objects in a sufficiently flexible manner to provide the navigation, quotation, annotation, knowledge representation, association and all other link capabilities examined in Chapter 2. In short, Mesh links need to be exposed to the Mesh in a manner to allowing a variety of link mechanisms.

- *Link relationships*

  Mesh links must be able to describe the nature of link relationships – including the characteristics described in Chapter 2 such as directionality, multi-ended linking, named endpoints and presentations. This support is made additionally difficult because hypertext systems have different minimum or even contradictory expectations for endpoint characteristics. For example, Xanadu expects a distinguishable FROMSET and TOSET to describe directionality, while Dexter specifies individual link endpoints as TO, FROM, BIDIRECT or NONE.

- *Link independence*

  Mesh links need the capability to be independent Mesh entities. This need can be justified by the example of independent links in Aquanet and Dexter and the desire to provide an equivalent mechanism in the Information Mesh.

Some Mesh links need to be "bundled" with Mesh objects. This capability is described by an Information Mesh proposal [9]: "Links can be either explicit or implicit; an implicit link is one that declares a relationship between objects that is a necessary part of one of the linked objects, while an explicit link represents a relationship that is not inherent to any of the objects it links... An implicit link is likely to reside with the object to which it "belongs," while an explicit link may reside anywhere, and in fact may need to be an object in the sense it can be named with an oid and have further links... [9]." Note that in Section 5.2.3 we shall propose an alternative to designating links as either explicit or implicit.

- *Endpoint capabilities*

  Mesh links may relate an object, some aspect of an object, or some substructure of an object. We use the term *endpoints* to describe the substructure related by a link. Mesh links must be able to support a variety of endpoint characteristics. In Dexter, the mechanism to designate components and substructure was implemented as a link specifier which dynamically resolved to a set of components and an anchor id. Thus, links should be able to designate the endpoints dynamically in a manner similar to Dexter specifiers. Further, endpoints should be transparent across Mesh object mutations.

  As noted in Chapter 2, link endpoints are fundamentally limited by the invariant substructure exposed by the nodes being linked and the system in which the links are implemented. From our examination of Mesh objects as nodes in Chapter 4, it should be apparent that Mesh object substructure is formalized as "parts". Further, it should be apparent that Mesh links can provide no guarantees about referenced objects – a link may be "dangling" because of object changes. Finally, the unavailability of complete entity information (as described in Section 4.4) prevents the implementation of a mechanism to determine all links to a particular object.

Mesh links can usually be viewed as passive data structures that relate but do not act on objects. We do not expect that the use of a particular link will result in many computations outside of the link object itself. However, there are a few special cases where a link should have the capacity to do more than simply reference Mesh parts. For instance, Xanadu provides a mechanism for linking to nodes through the use of a computation involving character matching. Mesh links should be able to perform equivalent computations on Mesh objects.

5.2 Implementation

Mesh links are implemented as Mesh objects that must play the *link-role*. The link-role allows the expression of link relationships through several mechanisms. Link endpoints are determined by the 'extract-endpoints' action. The set of oids related by a link (the object portion of a link endpoint) can be determined using the 'get-oids' action. The overall intent of the link-role is to specify the minimum requirements for Mesh links in a manner allowing maximum flexibility of implementation and specialization. [1]

Link Role:

Inherits from **object-role**

Actions

(**get-oids** *link role*) Required

Returns set of oids related by the link

(**extract-endpoints** *link role*) Required

Returns set of endpoints which describe the object and object substructure related by the link.

[1] Note that the link can play more than one link-role, where the roles may not be sub-role or super-roles of each other. We provide this capability by allowing the designation of the role in the link-role actions.

(get-number-endpoints *link role*) Required

Returns number of endpoints

(set-endpoints! *link role endpoint-list*) Optional

Changes the link to relate the specified endpoints and removes any previous endpoints. Endpoints provided as a set of descriptors.

content extraction/manipulation .

We utilize the default part manipulation mechanisms.

Parts

(endpoint : *unordered-set-of descriptor*) Required

Contains text of statement node.

Makers

(create *oid implementation endpoint-list*) Required

Create a link.

Link endpoints, utilized to reference an object and (optionally) object substructure, are implemented as *descriptors*. Note that we have not associated a type value with descriptors. A descriptor is a structure containing object, role, part and selector information. Descriptors are described in more detail in Section 5.2.4

Link-role endpoints can be listed in any particular order (unordered); there is no naming of endpoints in the base link-role. Endpoints do not contain an associated type value, direction or any other semantic descriptions. In short, capabilities to group or distinguish endpoints are not provided in the minimum link-role. Such a capability can be provided in roles which inherit from the link-role. The link role contains two restrictive requirements. First, the number of link endpoints returned by 'get-number-endpoints' is required to be a determinable value. Second, the link endpoints returned by 'extract-endpoints' must be discrete and returnable. These minimum requirements are unlikely to restrict Mesh link capability significantly.

The remaining Mesh link details are described by individually addressing the link attributes described in Section 5.1.

5.2.1 Link Utilization

Mesh links are exposed to the Mesh as link objects which play the link-role. Thus, Mesh links playing the link-role provide minimum capability. Note further that because links are objects, links can link links! The overall capability of Mesh links is demonstrated through examples in Section 5.3.

Implementing Mesh links as objects results in some limitations. For example, there is nothing to prevent a Mesh link from changing its exposed endpoints whenever desired. Further, the implementation of a Mesh link as an object requires that we invoke the overhead of invoking a Mesh object action every time we desire determination of the endpoints of a link.

5.2.2 Relationship description

Link relationships are provided through roles. Roles provide an extensible link type mechanism. Additional link capability is provided by creating a role which inherits, either directly or indirectly, from the link-role. Thus, new Mesh links can be defined by specifying a role which inherits from the link-role.

- *Directionality*

  The base link-role has no directionality information. Mesh links are inherently bidirectional in describing endpoints. Specific hypertext implementations of directionality can be provided through a link role specific identifier similar to Dexter's model of recording directionality with the added advantage that the domain of directionality, e.g. semantics, transit, etc., can be declared formally through the role mechanism.

- *Multi-ended links*

  Objects playing the Mesh link role can have multiple ends. Indeed, the base link-role allows links to relate a single Mesh entity or even no entities, although this raises a question about what is being "related". Regardless, an object playing the link role can choose to have no endpoints implying that it relates nothing! Note that this could be a temporary situation. An example of a single-ended link might be an "offspring link" assigned to an object that has none. We expect the common case will be a link with two or more descriptors; a specific link role will be introduced which provides these capabilities.

- *Presentations*

  The base link-role has no presentation information. However, more specific link roles can contain presentation information. For example, a Dexter link could easily have its presentation specification as a part.

5.2.3 Link Independence

Mesh link independence is assured because links are implemented as Mesh objects. Further, Mesh links can relate any objects; an object does not have to contain all links to it. One problem with using independent links to relate Mesh objects is that there is no bounded way to determine all possible links to an object. Thus, independent links can not describe "intrinsic" characteristics of Mesh objects because the independent link and the object could become "separated" in the Mesh; there is simply no guarantee the link will always be available to describe the object.

Implicit or "bundled" links are provided through the use of the composite mechanism described in Section 4.4. Composites ensure that Mesh links can be embedded in Mesh objects. Bundled links usually reference some aspect of the object with which they are bundled, but this is not necessarily required.

Note that implicit links are utilized to allow the Mesh-level expression of a link relation. If the link relationship is a "requires" relationship and there is no need to expose the exact parts required, then it makes sense to utilizes the composite object's "requires" operation rather than creating an implicit links which does the equivalent. One example of the need for this capability is WWW links "contained" in a WWW document, but exposed as Mesh object and Mesh links in the Information Mesh. By exposing the links as "required" by the Mesh object (through the use of a composite object), we can ensure that the Mesh links move with the object.

Link independence raises some feasibility issues in implementing a system that expects complete determination of all links to a specified node. As already noted, such complete availability of information is not possible in the Information Mesh. However, one can accommodate such systems in a limited manner by using the "requires" operation to specify all links to a node. For instance, Dexter nodes can answer the 'links-to' and 'links-to-anchor' actions described in Section 5.2.3 by examining the links "required" by the Dexter node. Clearly such a mechanism is insufficient for reporting all Mesh links to a given Mesh object, but the utilization of "requires" allows the determination of links designated as such.

5.2.4 Endpoint capabilities

Endpoints are realized using *descriptors*. A descriptor is a simple data structure containing object, role, part and selector information. A descriptor is more than an oid, to allow the distinguishing of a particular substructure component of a Mesh object (a part instance). Note that we have not associated a type value with descriptors. Further, there are no sets of descriptors in the link-role; all descriptors are presented as a single set. These decisions were made to minimize the requirements of the base link-role. We shall see later that type values and sets can be associated with descriptors in specialized link roles.

Base Mesh links are restricted to linking the substructure exposed by Mesh objects through parts. To simplify the implementation of descriptors, we only allow a single value for each object, role, part and selector information. We do not provide sets or ability to operate on part instances in the base link-role. Further, Mesh links can not specify a subpart or any other piece of a part. The link-role can not operate the on the linked part; the link merely expresses a substructure reference to the part.

There is no mechanism to hold a range or set of parts in the base link-role except by providing individual endpoints for each specifiable part. If an object does not provide parts for the role it is playing, then we can only provide an object and role in the descriptor. The remaining values are ignored.

A link may dynamically change the endpoints produced. Such capability is provided by allowing the link to perform computations whenever it is asked to expose endpoints via the 'expose-endpoints' action. Through this mechanism we may produce different endpoints at different times. For example, we can provide Dexter's specifiers (see Section 2.5.3) by hiding the specifier within the object and revealing the result of its computation in the link endpoints presented. In summary, Mesh links are able to designate dynamically endpoints in a manner similar to Dexter specifiers.

Note that minimum Mesh links are limited by the substructure exposed by the object for linking; we can only link to exposed parts. Linking a subcomponent or piece of a part can not be done with the minimum Mesh link. We need to express some form of endpoint computation which is not provided in the minimum Mesh link mechanism.

While minimum Mesh links do not support computations to get a part, it is possible to have a specialized link role which provides such capability. Unfortunately, this approach may not be recognized by the entity examining the link, limiting utilization of the link to those that understand the specialized link. One solution to this problem is to specify a generalized "computation" Mesh link which provides a general mechanism to perform endpoint computations. Unfortunately, such a link would require a mechanism to describe generalized control and state. Further, such a mechanism would require a mechanism to control the threading of computations across the Mesh.

Fortunately, there are several alternatives to a "computation" Mesh link:

1. Ask for a part to be created

   This approach requires both knowledge of the object being linked and the capability to create the desired part.

2. Make the object play a more suitably linkable role containing the desired part

   This approach requires that a suitable role be available such that one can force the object to play the more suitable role. The problem with this approach is that one may not have the permissions to force an object to play the desired role.

3. Wrap the uncooperative object into a composite object.

   This approach exposes the object through a separate composite object containing the desired parts. The composite object performs computations on the wrapped object to provide the desired parts. An example is creating a role which exposes paragraphs on top of an oid with only chapters by doing computations on the paragraphs.

   The problem with this approach is that the link relates the composite, not the original object. One way to work around this problem is to express a link to both the composite and the original object so that it is clear that the wrapped object is being described via the composite.

5.3 Link Examples

We demonstrate some example link roles. Note that these link roles are able to serve as a strong set of base Mesh link roles.

5.3.1 Named Link

The named link provides a set of endpoints, each endpoint named. Named-links provide a base set of link functionality that many other links can utilize to expose individually named endpoints.

Named-Link Role:

Inherits from **link-role**

Actions

(**extract-named-endpoint** *named-link endpoint-name*) Required

Returns endpoint described by endpoint-name.

(**add-named-endpoint!** *named-link endpoint-name endpoint-value*) Optional

Deletes endpoint with endpoint-name.

(**remove-named-endpoint!** *named-link endpoint-name*) Optional

Adds endpoint with endpoint-name. Endpoint is a descriptor structure.

content extraction/manipulation .

We utilize the default part manipulation mechanisms.

Parts

(**named-endpoint** : *named-of descriptor*) Required

Contains named-endpoints.

Makers

(**create** *oid implementation named-endpoint-list*) Required

Create a named-link. Named-endpoint list is a list of names and descriptor pairs.

5.3.2 Ordered Link

Set of endpoints ordered in some manner.

Named-Link Role:

Inherits from **link-role**

Actions

(**get-ordered-endpoint-range** *named-link start end*) Required

Returns range of ordered endpoints.

(**extract-ordered-endpoint** *named-link position*) Required

Returns the endpoint at numbered position in ordering.

(**set-ordered-endpoint!** *named-link ordered-endpoints*) Optional

Changes the ordered link to relate the specified endpoints. Endpoints provided as a ordered set of descriptors.

content extraction/manipulation .

We utilize the default part manipulation mechanisms.

Parts

(**ordered-endpoint** : *ordered-of descriptor*) Required

Contains ordered-endpoints.

Makers

(**create** *oid implementation endpoint-list*) Required

Create a ordered-link. Endpoint list is an ordered list of descriptor pairs.

5.3.3 Binary link

A binary link is a two-ended Mesh link. Binary links are guaranteed always to contain exactly two ends. Note that the Binary Link Role utilizes the inherited link-role actions and parts, but with the guarantee that the result of ‘extract-endpoints’ and ‘get-oids’ will return exactly two endpoints.

Binary Link Role:

Inherits from **link-role**

Actions

content extraction/manipulation .

We utilize the default part manipulation mechanisms. Note that the manipulation mechanisms must maintain the two endpoint characteristics.

Parts

(**binary-endpoints** : *unordered-of descriptor*) Required

Contains two endpoints of a binary link.

5.3.4 Link Example Summary

Note that in the previous examples, we have made mutability considerations optional. This allows named-links, ordered-links and binary-links to be potentially be implemented as immutable relations. Similar criteria was provided in designing the base link-role where mutability is optional to ensure that one can build an immutable link on top of the minimum link-role.

5.4 Extended Example

The power of Mesh objects and links is best demonstrated on a particular problem, preferably a dynamic environment in which changing objects are related by mesh links. We have chosen to create Mesh objects which represent the people, groups, and rooms at the MIT Laboratory for Computer Science (LCS). Specialized Mesh links describe the relationships between these three entities as the objects evolve through time as people, groups and rooms change.

5.4.1 LCS Entity Objects

Our example Mesh objects utilizes several specialized roles to describe their capabilities and representations. An individual person at LCS is represented by an object playing the LCS-Person-Role. LCS groups and LCS rooms are described by an LCS-Group-Role and LCS-Room-Role respectively. All three roles contain a "name" part. The LCS Person Role and LCS Group Role optionally contain a webpage and email part. The LCS Person Role optionally contains a phone part. All of these specialized roles inherit from the following Entity-Role which provides a mechanism to associate attributes with a named object:

Entity Role:

Inherits from **object-role**

Actions

content extraction/manipulation .

We utilize the default part manipulation mechanisms.

Parts

(**name** : *one-of text*) Required

Entity name.

(**attribute** : *named-of unspecified-type*) Required

Attributes for entity

Makers

(**create** *oid implementation name named-attributes*) Required

Create an entity with name. Named-attributes are attached to the attribute part.

5.4.2 LCS Entity Links

As previously described, the three specialized LCS entities (people, groups and rooms) are related using specialized Mesh links. The specialized links utilized to relate Mesh objects are the link roles: LCS-Group-Member-of and LCS-Occupant-of. LCS-Group-Member-of links relate a LCS Person to a LCS Group. Such relationships are unlimited; there are no limitations on the number of groups a lcs-person can be associated with as a member. LCS-Occupant-of links describe a relationship between LCS persons and LCS rooms. As with group membership, a person can occupy multiple rooms without restriction.

Both LCS-Group-Member-of and LCS-Occupant-of link roles inherit from the Member-of link role (which further implies the indirect role inheritance of named-endpoint and binary link roles). The Member-of link role allows entities to be related such that a member (as specified by an endpoint) is a component of a container (as specified by another endpoint). Note that while a Member-of link specifies a relationship between a “member” and a “container” but this terminology has no relationship to the composite object notion of “requires”.

Member-Of Link Role:

Inherits from **binary-link-role, named-link-role**

Actions

content extraction/manipulation .

We utilize the default part manipulation mechanisms.

Parts

(**member** : *unary-of descriptor*) Required

Member entity endpoint.

(**container** : *unary-of descriptor*) Required

Container entity endpoint.

Makers

(**create** *oid implementation member container*) Required

Create a member-of link. Member and container are descriptors.

5.4.3 Summary

The key insight for this example is that links can provide and expose capability based on their position in the role hierarchy. That is, extremely specialized link roles can utilize some of the more general links described in Section 5.3. As an example, a LCS-Group-Member-of link also plays (indirectly) the Member Link Role, Binary Link Role, Named Endpoint Link Role and the minimum Link-Role. By playing these various roles, the LCS-Group-Member-of link reveals itself as a 2-ended Mesh link utilizing named endpoints to describe some form of membership. Thus, the above objects and links which play more general roles through the utilization of role inheritance can be more widely understood.

The complete role specifications for LCS-Person-Role, LCS-Group-Role, LCS-Room-Role, LCS-Group-Member-of link role and LCS-Occupant-of link role are provided in Appendix 9.

5.5 Summary

Mesh links provide the primary mechanism for expressing object relationships in the Mesh. Mesh links express relationships through the utilization of roles and describe endpoints through the use of *descriptors*. A descriptor is a structure which allows Mesh links to specify an object, some aspect of an object or some substructure of an object. Mesh links are exposed to the Mesh as independent Mesh objects which play the link-role. Implicit links, describing "intrinsic" characteristics of Mesh objects, are provided through the use of composite objects. Thus, Mesh links can be "bundled" with Mesh objects.

All Mesh links must play the link-role described in Section 5.2. The link-role provides the minimum capabilities available for expressing relationships between objects. The link-role requires that endpoints be determined by the 'extract-endpoints' action which returns a set of descriptors describing the endpoints of the Mesh link. No directionality or presentation capabilities are provided with Mesh links. End-point capabilities are largely limited by the substructure exposed by Mesh objects through parts, but links may dynamically change the endpoints produced. Endpoint computations are possible, but are limited to specialized links.

In summary, Mesh requirements are met for a Mesh link mechanism. Minimum agreement is provided by requiring all Mesh links to play the link-role. Minimum coordination is met by ensuring Mesh link requirements account for unavailability. Flexibility is provided through the utilization of roles to create, describe and adopt new link types and mechanisms. Finally, we have demonstrated the flexibility of Mesh links in the form of various Mesh links and an extended example.

Chapter 6

Conclusions

The Information Mesh provides a framework for the implementation of a system of nodes interconnected by links expressing relationships; the Information Mesh kernel and object system provide the necessary system capabilities. The modified Mesh object system enhances Mesh link capabilities. The described Mesh link architecture provides a mechanism to relate Mesh objects.

In this chapter, we review Mesh links and describe how they satisfy the observations of Chapter 2. We conclude with a list of open issues.

6.1 Mesh Links

Mesh links provide the capabilities necessary to serve as the primary mechanism to express object relationships in the Mesh. The goal of Mesh links to provide a minimum mechanism for expressing Mesh relationships has been met. Further, Mesh links have been shown to provide provide a rich, flexible mechanism for relating Mesh objects. Finally, we noted that Mesh links need a mechanism to "embed" a link in an object for expressing fundamental object characteristics.

Overall, we have shown that meeting certain minimum requirements in links and the entities they connect is sufficient to provide a rich flexibility of relationship expressions. Thus, Mesh links provide the benefit of a minimum but flexible mechanism to express Mesh Object relationships.

6.2 Overall Linking Issues Addressed

Our examination of a Mesh link architecture has resulted in a stronger understanding of the object, system and link capabilities necessary for linking. We examine this understanding in terms of the hypertext system observations discussed in Section 2.6:

1. Scalability issues are often ignored.

   The issue of scalability is met by the utilization of the the Information Mesh's Mesh kernel and Mesh Object System for system and object capability. The Mesh link architecture accommodates scalability by utilizing the object system and by not requiring completely available system information.

2. Node and link typing limitations emphasize the need for an extensible typing mechanism for nodes and links.

   The Mesh Object System provides these capabilities to both Mesh objects and Mesh links through the utilization of roles to describe abstract structure and behavior of objects. Role capability as a flexible and extensible typing mechanism was previously described in Section 3.5.2. Further, Chapter 4 showed the ability to apply roles to provide the type capabilities of all examined hypertext text systems, including single value, attribute-value and hierarchical types.

3. Substructure interface limitations emphasize the need for a formal mechanism for exposing substructure.

   The Mesh Object System provides "parts" to reference substructure. As described in Chapter 4, parts are similar to hypertext node anchors but are more systematic and generalizable, as well as hiding representation and other implementation details behind an abstraction barrier.

   Note that the Mesh Object System was enhanced to provide exposure of part selector characteristics, specialized actions for certain selector types and a mechanism for the manipulation of part instance content. These part enhancements, while not strictly necessary, improved the overall capability of Mesh links.

4. Endpoint capabilities for substructure reference and computation are necessary.

   As described in Chapter 5, Mesh link endpoint capabilities are largely limited by the substructure exposed by mesh objects through parts, but links may dynamically change the endpoints produced. Mesh link endpoint computations are possible, but are limited to specialized links.

5. The necessary link capabilities for an effective hypertext system are unclear

   Mesh link minimum requirements are that all Mesh links must play the link-role. Thus, the link-role provides a minimum mechanism for describing and expressing relationships between objects. As demonstrated, these minimum requirements provide sufficient flexibility to allow a rich set of relationship expressions.

6.3 Open Issues

Several issues remain open to future examination.

- **Mechanisms for Object Discovery**

  There are no mechanisms for object discovery implemented in the present Information Mesh. In particular, there is no mechanism to find links based on a description, nor to find links to a particular object. Thus, there is a need for a link hint server (an entity which can provide links based on description or endpoints).

  Note that the implementation of Mesh links as Mesh objects implies that there is nothing to prevent a Mesh link from changing its exposed endpoints whenever desired. This makes the implementation of a Mesh link hint server increasingly difficult because the server must periodically determine if a stored Mesh link has changed its endpoints.

- **Endpoint Architecture Limitations**

  The link-role requires that the endpoints be countable, enumerable and reference a single part instance (no sets). We have not examined whether countable link endpoints is too restrictive. Further, we have not determined whether the inability to express sets of endpoints as a primitive Link-Role capability is too limiting. Finally, computation capabilities have not been sufficiently examined.

- **Mesh Part Capability**

  Mesh Parts have been enhanced through the exposure of part selector characteristics, specialized actions for certain selector types, and a mechanism for the manipulation of part instance content.

  These enhancements, while enhancing the overall capability of Mesh links, require addition examination and modification. For instance, there is no mechanism to describe the nature or value type of a particular part instance. Further, there is no mechanism to provide additional selector types or specialized actions in a generalized manner. These must all be pushed into the Mesh.

- **Presentation Capability**

  There is no generalizable mechanism for presenting Mesh Objects and Mesh links to a user.

Chapter 7

Object-Role

The object-role provides a starting point for all dialogs with Information Mesh Objects. Since all Mesh objects must play the object-role, we are guaranteed that the required object-role actions are answerable by any Mesh object. Thus, the Object Role describes the base set of actions and parts which all Mesh Objects must support.

Actions

(**roles-played** *object*) Required

Returns the list of roles that the object can play at this instant.

(**plays-role?** *object role*) Required

Returns true if the *object* plays *role*

(**play-role!** *object role implementation*) Required

Makes the given object play the given role using the given implementation. Initially, all objects play the *object-role*.

(**is-role?** *object*) Required

Returns true if the given object is a role. Objects which are roles can be used to describe the abstract behavior of other objects. Note that 'is-role?' is syntactic sugar for applying 'plays-role?' to an object and specifying the *role-role* for the role argument.

(implementations-supported *object role*) Required

Returns the list of implementation objects for the given role that the object supports.

(describe-yourself *object*) Required

Returns a description of the object. The nature of this documentation is out of the scope of this specification.

Parts

whole Required

The part containing the entire object.

documentation Required

The documentation associated with a given object.

Chapter 8

Versioning

"Versioning is an important feature in hypermedia systems. A good versioning mechanism will allow users to maintain and manipulate a history of changes to their network [13]."

8.1 Versioning Options

Versioning options include:

- *authoritative server*

  This approach uses a server which is guaranteed to contain the latest version.

  Utilizing an authoritative server requires the availability of the server for any versioning operations. Thus, an authoritative server requires a large degree of coordination and availability – a violation of the Mesh requirement for minimum coordination. Therefore, an authoritative server mechanism is best not utilized as the default behavior for objects in the Mesh.

- *name versioning*

  Name versioning associates each oid with an immutable object and a mechanism to determine the oid for the next "version" of object. This scheme is not only clumsy, but it breaks our intention of not associating semantics with oids. Further, there is no mechanism to determine the latest version.

- *latest time-date stamp*

  Latest time-date stamp versioning utilizes a time stamp to determine the "latest" version. The "latest" version is the object with a time stamp later than any others. The limitation of this approach is that there is no mechanism to ensure one has the latest version.

- *versioning time-out*

  Versioning time-out has a universal time at which point the information is invalid. This mechanism requires that either that the information have a life expectancy or that periodic updates are provided.

- *probabilistic versioning*

  Version is probabilistically valid depending on time since creation; after a specified period, object is only guaranteed to be latest with a specific probability. As an example, a "half-life" probability would specify a time period after which the object would only be half as likely to be valid as before.

8.2 Versioning Implementation

There is no clearly superior versioning implementation option. For our current object implementation, we utilize versioning based on time-date stamps – via the *latest time date stamp* mechanism. As previously noted, the key problem with this mechanism is that there is no means to ensure one has the latest version.

Note that regardless of versioning choice, Mesh objects may utilize additional versioning capabilities. For instance, Mesh object may choose to use an authoritative server in addition to time-stamps.

Chapter 9

LCS Entities and Semantic Links

This appendix describes the role implementation for the people, rooms and groups at the MIT Laboratory for Computer Science. Also described are the mesh link relationships which interrelate the people, rooms and groups. The roles are detailed on the following pages.

LCS Person Role: Objects playing the LCS Person Role represent an individual person at the MIT Laboratory for Computer Science. Note that room and group is not part of a lcs-person's attributes because a lcs-occupant-of and lcs-member-of link (described shortly) describes these attributes. The LCS person role inherits from the Entity role described in Section 5.4.1

Inherits from: **object-role**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**name** : *unary-of text*) Required

Contains name text

(**phone** : *unary-of text*) Optional

Contains phone number.

(**webpage** : *unary-of text*) Optional

Contains webpage URI.

(**email** : *unary-of text*) Optional

Contains email address (text URL format).

Makers

(**create** *oid implementation person phone webpage email*) Required

Create a lcs-person.

LCS RoomRole: Objects playing the LCS RoomRole represent an individual room at the MIT Laboratory for Computer Science.

Inherits from **entity-role**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**name** : *unary-of text*) Required

Contains room name text

Makers

(**create** *oid implementation room name*) Required

Create a lcs-room

LCS Group Role: Objects playing the LCS Group Role represent a group at the MIT Laboratory for Computer Science.

Inherits from **entity-role**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**name** : *unary-of text*) Required

Contains group name text

(**webpage** : *unary-of text*) Optional

Contains webpage URL.

(**email** : *unary-of text*) Optional

Contains email address (text URL format).

Makers

(**create** *oid implementation room name*) Required

Create a lcs-room

LCS Group-Member-of Link Role: An LCS Group-Member-of Link expresses a relationship between an object playing the LCS-Person Role and an object playing the LCS-Group role – namely that the person is a member of the group.

Inherits from **member-of-link**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**lcs-person** : *unary-of descriptor*) Required

LCS Person descriptor

(**lcs-group** : *unary-of descriptor*) Required

LCS Group descriptor

Makers

(**create** *oid implementation lcs-person lcs-group*) Required

Create a lcs-group-member-of link stating that LCS person is a group member of LCS group.

LCS Occupant-of Link Role: An LCS Occupant-of Link expresses a relationship between an object playing the LCS-Person Role and an object playing the LCS-Room role – namely that the person is an occupant of the specified room.

Inherits from **member-of-link**

Actions

content extraction/manipulation .

We utilize the default part content manipulation mechanisms.

Parts

(**lcs-person:** *unary-of descriptor*) Required

LCS Person descriptor

(**lcs-room:** *unary-of descriptor*) Required

LCS Room descriptor

Makers

(**create** *oid implementation lcs-person lcs-room*) Required

Create a lcs-occupant-of link stating that LCS person occupies LCS room.

Bibliography

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielson, and A. Secret. The world wide web. *Communications of the ACM*, 37(8):76–82, August 1994.

[2] T. Berners-Lee and M. McCahill L. Masinter. Uniform resource locators (url). Technical report, CERN, Xerox Corporation, University of Minnesota, December 1994. Network Working Group RFC 1738.

[3] Tim Berners-Lee. Hypertext markup language (html). Technical report, CERN, June 1993. http://www.w3.org/hypertext/WWW/MarkUp/HTML.html.

[4] Tim Berners-Lee. Hypertext markup language (html). Technical report, CERN, May 1994. Draft.

[5] Tim Berners-Lee and Daniel Connolly. Hypertext markup language (html). Technical report, CERN and Atrium, June 1993. Internet Draft. IIIR Working Group.

[6] Vannevar Bush. As we may think. *The Atlantic Monthly*, July 1945.

[7] David D. Clark, Karen R. Sollins, John T. Wroclawski, and Michael L. Dertouzos. Critical technology for universal information access. Research proposal submitted to ARPA, 1994.

[8] Karen R. Sollins David D. Clark and John T. Wroclawski. Paradigms for universality: Networking in the information age. Abridged Version of Proposal Submitted in Support of Work by the Advanced Network Architecture Group, 1991.

[9] Karen R. Sollins David D. Clark and John T. Wroclawski. Advanced network architecture - progress report '93-'94. Progress Report '93-94, 1994.

[10] D. Goodman. *The Complete HyperCard Handbook*. New York, 1987.

[11] Kaj Grønbaek and Randall H. Trigg. For a dexter-based hypermedia system. *Communications of the ACM*, 37(2), 1994.

[12] Frank Halasz and Mayer Schwartz. The dexter hypertext reference. *Communications of the ACM*, 37(2), 1994.

[13] Frank G. Halasz. Reflections on notecards: Seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7), 1988.

[14] Guy L. Steele Jr. *Common Lisp: The Language*. Digital Press, second edition edition, 1990.

[15] John J. Legget and John L. Schnase. Dexter with open eyes. *Communications of the ACM*, 37(2), 1994.

[16] Catherine C. Marshall, Frank G. Halasz, Russell A. Rogers, and William C. Janssen Jr. Aquanet: A hypertext tool to hold your knowledge in place. In *Hypertext '91 Proceedings*, pages 261–276, 1988.

[17] Theodor Holm Nelson. *Literary Machines*. The Distributers, 1987.

[18] J. Postel. Media type registration procedure. Technical report, USC/ISI, March 1994. RFC1590.

[19] Dave Raggett. Hypertext markup language specification version 3.0. Technical report, W3C, March 1995. IETF Draft draft-ietf-html-specv3-00.txt.

[20] Donald L. McCracken Robert M. Akscyn and Elise A. Yonder. Kms: A distributed hypermedia system for managing knowledge in organizations. *Communications of the ACM*, 31(7), 1988.

[21] K. Sollins and L. Masinter. Functional requirements for universal resource names. Technical report, MIT/LCS and Xerox Corporation, December 1994. Network Working Group RFC1737.

[22] Karen Sollins. Overview of the information mesh. DRAFT, 1995.

[23] Benvenido Velez-Rivera. Information mesh objects. Working Document.

[24] Benvenido Velez-Rivera and Alan Bawden. The information mesh kernel. Technical report, MIT Laboratory for Computer Science, 1994.

[25] Fabio Vitali Wilma Penzo, Stefano Sola. Further modifications to the dexter hypertext reference model: a proposal. Technical report, University of Bologna, Laboratory for Computer Science, January 1994.