# Linking in a Global Information Architecture

Karen R. Sollins Jeffrey R. Van Dyke<u>\*</u>

#### Abstract:

As network-based applications and knowledge-based systems become more prevalent and useful, it will become increasingly important to provide a common, long-lived model of the infrastructure on top of which such applications will exist. Not only does the longevity and evolvability of the infrastructure have a direct impact on the value of the information in it, but also the provisioning of a meta-level infrastructure capable of defining relationships among the pieces of information become increasingly important. In this paper, we examine, first, a general, flexible information architecture and then enhance it with an extensible, simple model of linking, recommended as a component of the substrate of network-based applications and knowledge-based systems of the future.

Keywords:

Information Infrastructure, Linking

### **1** Introduction

A system of any sort has behind it a model or architecture that defines the components of the system and how they relate to each other. The architecture will determine the extent and limitations of the system, by providing a set of capabilities. Architectures can be more or less restrictive and more or less complex, making the capabilities of a system implementing the architecture more or less accessible to the client of the system. If the architecture is both restrictive and complex, it is likely that a system implementing it will be less understandable to a user of the system than simpler, less restrictive architectures.

As the Internet grows, we find ourselves in a world of evolving heterogeneity. There is a multiplicity of transport technologies, protocol suites, programming and runtime communications paradigms, and applications support services. In addition, there are the applications on top of all this. Many organizations find that although they want to be able to take advantage of new developments in technology in terms of speed, effective utilization of resources, and increased functionality, they also have at least some applications that are extremely stable; they do not want to revise or rewrite those applications as the technology evolves.

In addition, the information generated by the organizations often has a longer lifetime than we have traditionally expected, presenting us with two closely related issues. First, with time the computing universe is expanding. Second, this longer-lived information will often outlive the original application from or for which it was created. By being both more widely available and more independent of specific applications, the information becomes more valuable to society as a whole.

If we take a layered perspective, we have distributed and network-based applications supported by transport protocols between hosts. The discussion above motivates the provision of an information infrastructure, separating the applications from the supporting infrastructure. This is, in effect, moving the boundary of what is ``in the network" yet further into the host machine and pushing to include not only references and access to information, but also information structure or relationships. The implication is that using a name to build, find, or access information or relationships among information objects will be comparable to accessing other resources provided by the network. We will need a global model with standard access models or protocols for learning about and accessing information resources.

Over a number of years the Internet community has developed various network wide information

architectures. One early such architecture is based on FTP [18], the file transfer protocol. The elements of FTP are file repositories and files themselves. The repositories or hosts are identified by the Domain Name System [13, 14] and communicate with each other by the FTP command set. Files are named by the repository on which they reside. FTP does not distinguish among sorts of files, although it does permit several transfer modes, such as ascii and binary, providing simplistic presentation models. With the growth of systems such as Gopher [1] and the widely used World Wide Web [2], the nature of Internet based information architectures has taken another leap forward. At this point it is valuable to consider carefully the strengths of such an architecture and any future requirements for it, in order that it evolve effectively.

Setting aside any global architecture for information, distributed architectures have become popular for a variety of reasons. First, since much information and its management are by nature distributed, distributed capabilities better reflected the inherent nature of the information. Second, the parallelism of multiple machines provide higher throughput. And, third, distribution and replication provided improved resiliency to failures.

Within the general framework of global information architectures, we will focus most specifically on **linking**. One of the weaknesses of the FTP architecture was the inability to build a superstructure within it. For example, although a file name might be embedded in a file that was retrieved, the architecture and therefore the implementation had no way of recognizing the content of a retrieved object in order to discern any composite or structured relationship expressed within an object. In comparison, the hypertext model, as proposed, for example, in the Dexter model [7], reflecting a consensus among hypertext models, or instantiated in the World Wide Web, has as a central component of the model, the ability to build and recognize in the architecture arbitrary relationships among objects. Linking of this sort allows for a significantly increased value in the information system because is there an ability to store, manage, retrieve and manipulate not only the base information, but also the relationships among them or the infrastructure.

In the context of a general information infrastructure architecture, we will examine the issues surrounding linking, concluding with a a system that supports general, flexible and extensible linking. First, we will explore the requirements for both a general information architecture and more specifically for linking within such an architecture. With this in mind we will consider the work of others, and in particular other linking models. We will then lay out the information architecture of the Information Mesh, and with this general model in mind, address the architecture and design of links and related linking issues. The Information Mesh Project lays out a common, extremely general substrate that sits between network-based systems such as distributed programming or database environments and network transport protocols such as TCP or other bitstream protocols. The intention is to provide sharing and exchange of potentially long-lived information independently from the underlying transports protocols and related infrastructure. For a complete description of this work and its implementation see [23].

# 2 Requirements

We begin the discussion of requirements by considering those of an information architecture such as the Information Mesh in general terms. Within such an architecture, we then consider the specific requirements for linking. The general requirements lead to an object model by means which satisfies our requirements for linking.

### 2.1 Requirements for the General Information Architecture

As discussed above, allowing for heterogeneity while providing a stable architecture within which applications can exist is critically important. In addition, it is clear that, because network-based activities will occur across administrative and programming model boundaries, no single model, policy, or even set of policy mechanisms, nor, as a matter of fact, any single way of doing anything will suffice. As we will see in

the Information Mesh architecture, implementations and realizations of it can be separated cleanly from the architecture, allowing for heterogeneity, multiplicity, and flexibility coincidentally with a common infrastructural model. In terms of layering, the Information Mesh architecture provides a minimal model below the level of any application, that will allow it to survive and permit new applications and applications paradigms to evolve unconstrained by the infrastructure.

The goals of the Information Mesh project are:

- **Ubiquity:** The Information Mesh should provide support for network-based applications accessing information that is distributed both physically throughout the net and administratively across regions of differing management policies.
- **Longevity:** Both information and identifiers for the information should be able to survive indefinitely; this means that, at some level of abstraction, the same object is considered to exist for all practical purposes 100 or more years.
- **Mobility:** Information should be able to move not only from one physical location to another, but also from one administrative region to another. An administrative region may range in scope from organizations claiming their own boundaries to individuals managing their own objects, or handing that control to someone else.
- **Homogeneity:** The Information Mesh should provide a single model for information identification, location, and access, as a substrate for distributed systems and applications. Such an abstraction barrier should allow for taking advantages of increased functionality only when desired. A stable substrate model is a requirement for a world in which applications and information have independent lives.
- **Heterogeneity:** The Information Mesh should be prepared for changes both from above and below. It 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.
- **Resiliency:** In an extremely large network, unreliability is a fact of life. Hence, there may be situations in which it will be impossible to locate or access a particular piece of information. Both the Information Mesh and the applications using it should be resilient to such a lack of success.
- Evolvability: The Information Mesh should be prepared to evolve as application and administrative requirements evolve. This may mean supporting new sorts of information, as well as new sorts of relationships. It is even possible that particular pieces of information may evolve as new aspects of them are created or recognized. As mentioned above, the Mesh should also be prepared to take advantage of the evolution of lower level support.
- **Minimality:** In order to succeed the Information Mesh should be as simple as possible, placing a minimum of requirements and restrictions on its users. We must understand what is required of it to achieve the other goals identified here, and provide no more than that minimum.

In <u>Section 4</u>, we will describe an architecture that meets these requirements.

#### 2.2 Requirements for Linking

In addition to the requirements for the Information Mesh architecture, we can identify six additional requirements for linking and building relationships. As we will see in the discussion in <u>Section 5</u>, by making the features explicit, they can be managed and utilized in a consistent and clean manner.

- Multiplicity: One must be able to express relationships between two or more objects.
- Link typing: It is important to be able to express the nature of a link or relationship extensibly and with unlimited scope, in order to support evolution in the representation of knowledge.
- Linking into the structure of objects: In order to provide generality and flexibility in linking objects, it is necessary to be able to link components, aspects, views, or parts of objects to each other.
- Linking to links: There are situations in which the ideas being linked may be themselves links. We

must be able to relate to links.

- **Composition of objects**: Relationships can be divided into those that are intrinsic or integral to an object and those that are not. If a relationship is intrinsic to an object, that composite object is unable to behave in the intended manner without its components. In contrast, other extrinsic relationships do not have a direct bearing on whether or not the related objects behave as expected. It is important to be able to express this distinction and support composite objects.
- Support existing models of linking: Existing systems define the minimum set of characteristics we must enable or support.

Of these issues, we will see later that composition will be provided by a separate mechanism, as a result of supporting the others with a single mechanism. Composition provides the ability to combine more than one object into a single composite object, expressing a "requires" relationship, a statement that a particular set of objects playing specified roles are required for the composite to behave in its intended manners. Composites will be discussed further in Section 5.2 It is within the context of the goals and requirements identified above that we will address related work.

## **3 Related Work**

The related work that is addressed here falls into two categories. The first is work that relates to the Information Mesh in general. Here, the background is quite broad, so we will simply touch on a sampling, discussing both the naming and addressing issues, as well as typing. The second set of related works again are only a sampling, in this case to demonstrate the breadth of current thinking in linking models. It is important to understand the scope of linking models, because, if our proposal is to be general purpose, it must support existing systems as well as any requirements.

In terms of naming and addressing, the work that is most closely related to the Information Mesh is what is coming from the Internet Engineering Task Force, in particular the separation of location from long-lived naming, as found in URLs [9, 3] and URNs [20]. In this model, a URL specifies not only the location or address of an object, but also, at least traditionally, the access protocol as well. This is also current practice in the World Wide Web [4]. In contrast, URNs are intended to be long-lived, globally unique, and permanently assigned to a single network based resource. There are older global naming schemes, such as the Domain Name System [13] and X.500 [17]. Although names in both of these systems are globally defined, there are no restrictions on reusing a name for different objects at different times. In contrast, a system such as ANSA [21], naming [22] is not global, but rather a federation of more local naming contexts, with gateways among them to translate them. By this technique, within each context there is a single model of naming the whole universe. One can incorporate any legacy naming system at the cost of requiring both negotiation at the time of federation of legacy systems and that names always be recognizable as such at times of transmission, since translation is required. In addition, it is not clear whether cyclic naming can systematically be avoided.

The typing model has three major sources of references, those that provide functional abstraction, such as the CLU programming language [10] and CORBA interfaces [16], those that define a more structural abstraction, such as CLOS [8.], and those that support polymorphism, in particular KRL [6] and Eiffel [12]. CLU provides the ability to define an extensible set of data abstractions. Representation and implementation are not part of the specification, but rather only the signatures of the procedures defined for each *cluster*. An object is created as an instance of a particular cluster and remains exactly that and no more throughout its existence. CORBA has a similar model for its *interfaces*, with the additional feature that an interface can inherit from one or more other interfaces, providing for an inheritance relationship among interfaces. CLOS defines classes which reflect the structure that an object of each class will have. Functionality is factored out in this model, but provided by a combination of *generic procedures* and the methods that implement them. A class does not specify which generic procedures are or should be provided for an object of a particular class. Eiffel and KRL both propose polymorphic typing models. In addition, KRL allows for an object to

evolve in terms of the sort of knowledge it embodies as it becomes better defined. The Information Mesh *role* model takes aspects of all of these, in order to provide the functionality and structure that each object supports.

In terms of linking models, we will summarize the issues by considering various aspects of hypermedia modelling. In particular, the Dexter model [7], Xanadu 15], Aquanet [11], and the World Wide Web [2, 5]. Rather than cataloguing all these systems, we will discuss them based on various features: scalability, typing, availability of substructures for linking, link endpoint capabilities, and characteristics of links themselves.

First there is the question of ubiquity and whether a system supports or even allows for it. Neither Dexter nor Xanadu scales up; both require links and other system information be completely available at all times. The WWW does better, by providing URLs that are globally defined, but are location dependent, limiting one's ability to relocate or replicate documents in different places.

Second, the typing mechanisms for nodes and links in these systems vary widely. Xanadu provides no typing for nodes or links. The WWW provides a single "relation" name for links, allowing for a string, defined by the W3 Consortium. These are single valued strings with no formal definition behind them. Aquanet supports hierarchically extensible typing for nodes and links alike. These provide structural constraints based on the CLOS model, but no definition of functionality. Thus, Aquanet supports polymorphism, while the other models do not. They also do not support polymorphism. Lastly, the WWW and Dexter also provide an attribute-value mechanism for nodes and links. This allows for the association of characteristics with links and nodes, although it is not strictly speaking a typing mechanism as with the WWW single values for relations. It has a serious name conflict problem, because there is no model of global naming or attribute-name resolution in either system.

Each of these typing mechanisms has limitations. 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.

Third, in general, the structure of objects being linked is not part of the link typing facilities described above and therefore the question of exposure of the substructure of objects must be addressed separately. Aquanet provides no substructure exposure, allowing only linking to whole nodes. In contrast, Xanadu exposes complete documents, supporting linking to parts of documents by pattern matching against strings. Dexter and the WWW provide a middle ground in which arbitrary anchors can be defined on ranges within objects, at the instance level. There is no model that objects of one sort might all have the same names for some consistent set of anchors. HTML 3.0 [19] will provide naming and therefore linking to almost all syntactic structures. There will still be no model that there will be sorts, kinds or types of objects that will have consistently named substructures.

Fourth, link endpoint capabilities are generally loosely coupled to substructure exposure. For example, because Aquanet exposes no substructure, link endpoints can do no better than link to whole nodes. The WWW and Dexter base endpoints on anchors, linking from one object to another. It should be noted that in HTML, one can define either end of a link to be a whole object. By using the "link" feature of the header element, one can link from a whole object. One need not specify a remote anchor name, in which case the remote endpoint is the whole object. In HTML 2.0 the only elements of a object that can be the ends of a link are anchors. A third alternative is that found in Xanadu, of computed links. These are not related to exposed structure, but rather to the computations available on an object. A powerful link endpoint mechanism would utilize exposed substructure invariants, yet provide the capability to utilize computations on nodes.

Finally, in addition, there are a number of different sorts of characteristics that each of the systems considers

important for links, each with its validity and utility.

- **Dimensionality of links**: Xanadu, Aquanet and Dexter can relate more than two entities. The WWW restricts links to being two-ended structures.
- **Directionality**: Xanadu expects a distinguishable FROM-SET and TO-SET. In contrast, Dexter marks individual endpoints as either TO, FROM, BIDIRECT, or NONE, although it is not explicit about the meaning of directionality. For example, it might express evolution, transit, or one of a variety of other directional sorts of relationships. The WWW has implicit directionality from the markup in a document to the referent. HTML 3.0 will provide a REV flag to reverse the direction.
- **Presentations**: Dexter links provide a "presentation specifier" with both the link and each endpoint. Aquanet utilizes a graphical appearance specification associate with node and link types to designate the presentation of Aquanet objects. The WWW utilizes HTML as a markup language to describe presentations.
- Link independence: Aquanet and Dexter links are independent hypertext entities. The WWW and Xanadu require that links be embedded in a hypertext node.
- Endpoint naming: All Aquanet endpoints are named. Some WWW and Xanadu endpoints or anchors are named. Dexter does not name its link specifiers.

We will address these aspects of links again in relationship to our own proposal in <u>Section 5</u>.

## 4 An Architecture: An Object Model

Many of the requirements of the Information Mesh are also requirements of systems such as the World Wide Web, in particular ubiquity, support for heterogeneity and the need for homogeneity, as well as minimality. What makes the Information Mesh effort distinctive is the primary focus on longevity and the attendant requirements for mobility and evolution. If the system and the information in it is to survive and continue to be useful, it must be prepared for both mobility, information and clients of the information will move, and evolution, both the information itself and the applications may evolve with time. These are issues that will grow in importance with broader development and deployment of the WWW. As an example, we can consider a text document, which makes reference to a 10 second piece of a video and audio recording of a speech. In order for a client to "read" the text document, there must be available some mechanism for viewing and listening to the section of the speech. At a later date, the audio component is run through a speech recognition system, after which it can be enhanced to have a text component as well, with indications of the relationship between time in the audio and specific words in the text. If the client is prepared for such evolution, the next time the human wishes to read the document, one might be able to print it. Furthermore, at another time, a new, specially tuned video/audio storage service might come into existence. At that time the video/audio components may be moved to the new service for better access, while the whole object also remains at its original location.

In considering this simplistic example, two major issues are highlighted. First, we need to be able to name or identify the object independently of where it is located, or perhaps even of how one accesses it. Second, we need some model of the functionality that objects support, in order both to be able to understand them and also to allow them to evolve. We will consider these two aspects of the Information Mesh separately.

#### 4.1 Names and Access

There are three functions that are often tightly coupled in naming. In this work we have separated them, in order to better support longevity, mobility, and evolution. Names are often used for identification, in order to distinguish named objects from each other without direct access to the objects in question. Thus, for example, in many cases it is useful to be able to compare two names to determine whether or not they refer to the same object. Depending on uniqueness, one can answer several different questions about the

Linking the Mesh

distinction among objects. If no name can be assigned to more than one object, then if two names are equal they refer to the same object. In contrast, only if each object can have no more than one name, can one be certain that if the names are not equal then the objects are not the same object. If a name can be reassigned to different objects, there is no way to use the names to test for equality or distinction among objects.

The second function often provided by names is access. This may take the form of an address with or without an access method, such as a transport protocol. For example, although this may not have been a requirement, URLs, as they are used in the WWW, generally define an address in an address space of a particular transport protocol. Furthermore, it is assumed that that protocol is the one that will be used for accessing the object. Thus, *http://www.w3.org* defines a location of a file; it is also common practice to assume that the machine providing that file expects HTTP to be used to access the file with that name.

The third function often ascribed to names is something descriptive in the name. This may be something that makes it easier for humans to remember the name, such as something about the nature or content of the object, or may be something of use to a program, such as identifying the programming language in which some piece of code is written. These are important functions to provide, although there is no single "best" way to capture them. For names that are intended to be human friendly, one probably does not want them to come from a global namespace. Furthermore, what is mnemonic for one person may not be for another, if for no other reason that they have different interests in the named object. If we engineer a system in which applications expect to find information in a name, then that information had better remain correct for as long as the name will be used in that way. Many such characteristics cannot be guaranteed not to change. For humans, we are better off providing small, human friendly namespaces, with translations to globally unique names or identifiers, while for applications, we are better off providing some other way of learning about meta-information than embedding it in names.

The proposal for naming in the Information Mesh separates the provision of the three functions from each other. This is similar to the proposal in the IETF standards process (See the work of the Uniform Resource Identifiers Working Group [9, 20, 3]. ) Each object or resource in the system has an *oid* (object identifier) or *URN* (uniform resource name). These are globally unique, long-lived (in other words the intention is that they will never be reused), and are not required or expected to carry semantics. Thus an oid or URN will only ever be assigned to one object or resource, and that is its sole required semantics. It may have other semantics that the creator may choose to expose, but no one and nothing can depend on or expect there to be more. Beyond that, in the URI Working Group terminology, there will be Uniform Resource Locators (URL), and Uniform Resource Characteristics (URC). The URL indicates an access protocol and location. The URC is the source of meta-information about a resource. This may contain ownership, access constraints, URLs, and other information about an object. It is also a potential container for information about programming or natural languages, and any other information about an object or resource deemed useful. In the Information Mesh these two sorts of information are separated.

The Information Mesh has a need for a particular sort of meta-information, called *hints*. It is assumed that there will be a number of services that are able to resolve URNs into location and protocol information. A collection of hints related to a particular URN will consist of a set of potential routes to accessing the object. The most direct may be a previously known address. Slightly less direct would be the address of a previously successful resolution service, or a URN for such a service. There may be a variety of such services with varying access policies. Thus, for example, some may be limited to certain communities, or may require a fee, or provide fairly ubiquitous, but not very current information, while others may make an effort to be current, but may not be as readily available. The set of hints at one location may be different from those at another for a variety of reasons, such as varying access policies or previous successes and failures. When a resolution service receives a request, in addition to or in lieu of returning a resolution for a URN, it may return alternative hints, thus further increasing for methods of resolving the URN, and increasing the divergence of hint information for a URN at different locations.

The Information Mesh and the work of the URI group to some extent separate the provision of naming as described here from that needed for humans on an everyday basis. The URN requirements document [20] proposes that URNs should be human transcribable, as distinct from human friendly, implying that humans might easily remember and use them.

Thus we reach a position where objects have globally unique, human unfriendly, long-lived names or identifiers that are translated by some service into addresses, or the "names" used by the transport and access services. We have a model for a hint mechanism that supports that translation.

### 4.2 Typing: Roles

The Information Mesh provides a rather distinctive typing model for objects, allowing for more flexibility and evolution than is traditional. It also must operate in a universe in which enforcement of typing cannot be guaranteed because of its federated nature. It is composed of a set of cooperating components that can agree to behave correctly, but cannot enforce that.

Object behavior in the Information Mesh is built around the concept of the *role*. A role has three aspects, *actions*, *parts*, and *makers*. In each case, some may be required and others optional. Only those that are required will necessarily be provided by all implementations. The actions of a role define the abstract functionality of that role. They are a specification of the actions, not implementations of them. Similarly, parts define the abstract structure of an object playing a particular role, but how the structure is represented in any particular situation is not part of the role specification. Finally, makers define the abstract functions used in creating objects playing a particular role. Again, realizations of makers are distinct from the specification.

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. Inheritance is singly rooted in the *object-role* (see <u>Appendix A.1</u>), but beyond that multiple inheritance and, in fact full polymorphism, is provided. Not only can a role inherit from more than one super role, but also objects can play more than one role at any given time. Furthermore, the set of roles an object plays can evolve over time.

Finally, roles themselves 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*. Because roles are first class objects and the object-role requires the "roles-played" action, one can always determine the identities (oids) for those roles, and find the definitions of those roles, barring access limitations.

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 objects may utilize multiple implementations. It is the job of implementations to actually figure out how to implement new nature on old objects. Implementations are first class objects related to but not part of the roles they implement. There may be more than one implementation of any role.

Thus, the Information Mesh provides a single, extremely general object model. Everything is an object and an object is defined by having one or more oids and playing one or more roles.

#### 4.3 Implications for Linking

With the object model supported by the Information Mesh, we provide a simple model in which to support linking that meets the requirement of longevity. There are two aspects to this. First, by separating naming from location, objects can move. It is only if and when one needs to access an object identified in a link that resolution need occur. Otherwise, if there is a guarantee that oids will not be reused, linking using oids can never cause unpredictable or surprising behavior by linking to a different object by using the original oid.

Second, the defined abstract parts of an object allow for implementation and representation independent linking into the structure of the object. Thus, one can link to a view, component or aspect of an object, with full knowledge that the existence of the part is independent of a particular implementation. It is not based simply on the syntax of a particular instance. Hence when an object has evolved to a new representation either through time or because of a new location, by linking to parts as defined by a role, the link should remain valid, assuming the object has not mutated at the abstract level in the intervening time.

### **5** A Linking Architecture

The proposition of this paper is that the linking problem should be handled by two complementary architectural features. First, the link itself will be a first class Mesh object. Second, intrinsic relationships will be provided a "composite object" mechanism as an enhancement of the basic object-role. We will discuss these two separately.

#### 5.1 Links as First Class Objects

In order to support links as objects, we will define a generic link-role. (See <u>Appendix B.</u>) Several other aspects of the object model will comprise the full link model. Each link will be a first class object with one or more oids assigned to it and playing one or more roles. At least one of these roles will be a link role, either the generic link-role or a subrole of the generic link-role. In that capacity the link will be able to provide the answer to the 'get-oids' action, enumerating the oids for all the objects linked by the link in question.

The parts of a generic link are simply an unordered, unnamed set of *endpoints*. 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 oid, role, part and selector information. There is no provision in the generic link-role for the selector to determine a set or range of parts. Each descriptor identifies exactly one endpoint. Subroles of the generic link-role can provide such capabilities, leaving the generic link-role as simple and general purpose as possible.

Capabilities to group or distinguish endpoints are not provided in the minimum link-role. Link role endpoints can be listed in any order; there is no naming of endpoints in the base link-role. Endpoints do not contain an associated type or value, direction or any other semantic description. 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 capabilities significantly.

In terms of the sorts of issues addressed in considering other linking models, we can consider each separately. First, the linking model is general, flexible, and extensible enough to allow for whatever sort of link utilization might be required. None is dictated or proscribed by the model. Second, again because of the extensibility of the typing model, directionality can be expressed in any of the ways required by the pre-existing models. Mesh links are implicitly bidirectional, although this can be enhanced or restricted as needed. Third, links provide for multiplicity, although again any limit on the number of endpoints of a link can be provided by more restrictive link subroles. Fourth, because links are first class objects, they can support models such as Aquanet and Dexter. By being first class objects, links no longer can easily provide intrinsic relationships, but this topic will be addressed further below. Last, because links are relating abstractly structured objects with potentially named components, all the problems addressed in discussing endpoint capabilities and therefore linking into substructures are non-existent in a model such as this. It is also worth noting that the generic link-role does not provide for presentation information, but more refined

subroles can do this. Mesh links can be defined to provide the more limited capabilities of each of the other systems.

It is important to remember that Mesh links can provide no guarantees about referenced objects; a link may be "dangling" because of object changes. In addition, the unavailability of complete entity information prevents the implementation of a mechanism to determine all links to a particular object. Thus, this feature cannot be provided for such systems as Dexter or Xanadu without further mechanism.

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 refer to 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.

Given the generic Mesh link role, more interesting link subroles can be defined. For example, one might support endpoints named within the scope of a link, as in Aquanet. As detailed in <u>Appendix C.1</u>, this will require additions to both the parts and actions of the link role. As part of support the World Wide Web or Dexter links, one might need the binary-link subrole, requiring only a change or restriction in parts from the generic link role, as demonstrated in <u>Appendix C.2</u>. In order to support models such as Dexter, the Web, Aquanet, or Xanadu, roles for their nodes need to be defined as well. For a full description of such roles, see [23]. A third interesting link subrole is the ordered-link role. Here the ordering of the endpoints of a link are important, as defined in <u>Appendix C.3</u>. Additional new subroles might be defined, inheriting from several of these such as a named binary link. Extensibility and inheritance are important features here.

#### **5.2** Composites

There are several possible alternatives for expressing composition in Mesh objects. Security and availability considerations limit our realization options. The main issue is whether composites can be implemented using the basic Mesh capabilities or whether the model will need to be extended. We can identify five options:

- **Requires link**: In theory, all relationships among 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 cannot describe intrinsic characteristics of Mesh objects because the independent link object could become "separated" in the Mesh. The reason for this is that there is no implementable Mesh mechanism to determine if all possible link objects have been examined or determined. Thus, links cannot 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 of the object. This means that the underlying representation of an object, whether it currently happens to be composed of several other objects, will determine which roles it does or does not play. This tight coupling between abstraction and implementation, and in particular this reverse dependency between them seems like a bad idea.
- **Monolithic object**: Monolithic objects bundle all required objects into a single object, wrapping objects via some as yet unspecified mechanism exposing the embedded objects through some interface. The advantage of this approach is that previously distinct objects are now accessed through a single, monolithic object. Unfortunately, security and practicality prevent use of such a mechanism on all objects. First, one may not have access permissions to all objects to be bound into the composite. Further, one might desire a composite object without the requirement of moving all objects into one monolithic object. Finally, this mechanism does not work if an object is a component of more than one composite object.
- Complete object awareness: Another option is to require that every object maintain a list of all

composite objects of which it is a member, contained or containing. This will ensure that every object is completely aware of the composite relationships of which it is a member. There are several problems with this approach. First, it would necessitate that all objects maintain a store describing all composites of which they are members. This would require that all objects be mutable and provide permission for modifying composite attributes. For public documents, this is untenable. Second, it would be necessary to synchronize all copies of an object to ensure linking to one object is exposed by all copies. Again, this is untenable.

• **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. This option is part of complete object awareness, in that an object is aware of its components, but not those object of which it is a component. The main problem with this approach is that it entails additional capability to the overall Mesh.

Our choice is the last of the options (See <u>Appendix A.2</u>), that of pushing the notion of "requires" into the core capabilities of the Mesh, by adding a new optional action to the object-role, 'get-required-objects'. Since 'get-required-objects' is an optional action, it may either not be inherited in subroles, or not be implemented if inherited, since it is optional. In either case, an object playing such a role could not be a composite without an implementation of the action.

'Get-required-objects' does not produce the closure of required objects and roles, but only those objects and roles directly required by the specified object playing the specified role. The only exception occurs when three conditions are true simultaneously. First, the object must be playing multiple roles. Second, there must be an interaction among the roles. Finally, the object must have different notions of composition for the different roles. Under such conditions, the result of invoking 'get-require-objects' contains the required components of all the roles played by the object.

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 assumes privacy regarding objects contained in one's composite, but is also makes the determination of all composites containing a particular object impossible. Furthermore, composites can provide no guarantees about the "contained" objects; a "contained" object may change unexpectedly.

Thus, we have proposed incorporating the solutions to the linking problem fully into an object model such as that of the Information Mesh, in order to meet the requirements we set out initially. For the complete report on this work, including an implementation see [23].

## **6** Conclusion and Summary

Not only does the Information Mesh architecture meet the requirements originally set out for it, but also by making links first class Mesh objects and enhancing the object-role with the ability to handle required components, we provide a simple self-consistent architecture that meets the requirements set out originally for linking. The general model of globally unique, long-lived oids with the attendant hint mechanism, and the role model, together meet the requirements of supporting ubiquity, longevity, mobility, homogeneity, heterogeneity, resiliency, evolvability, and minimality as they are defined in Section 2.1. In addition, for links we have provided support for multiplicity, link typing, linking into the structure of objects, linking to links, composite objects, and support of existing models as defined in Section 2.2. There remain a number of open issues in this work. For further discussion see [23]. For example:

- resource discovery with its implications for how to determine what should go into a link,
- generalized computations in endpoints with their implications for portable code and the role model,

• part naming in more detail) with the implications for the nature of selectors inside descriptors.

There are a number of directions to pursue in making these idea available in the World Wide Web. As several communities are considering as well, moving to an identifier rather than locator (URL) based labelling scheme is important. In order to do this either a single universal name resolution scheme will be needed or an architecture feature such as *hints* will be needed to allow for discovering and using a variety and evolving set of resolution services. *Roles* or a more generic and extensible typing scheme than is currently available is important. We are investigating a simple first step that requires no changes to the current infrastructure. It will provide simple structural templates that would define commonly understood anchoring schemes for specific templates or "types". Another area where preliminary work is being done is in providing "link" web servers. These and other efforts are needed to move the World Wide Web to a more extensible, evolvable and long-lived infrastructure.

### References

1 Anklesaria, F. et al., *The Internet Gopher Protocol (a distributed document search and retrieval protocol)*, Network Working Group RFC 1436, March, 1993. See also <u>ftp://ds.internic.net/rfc/rfc1436.txt</u>.

2. Berners-Lee, T., et al., *The world wide web*, Communications of the ACM, 37 (8):76-82, August, 1994.

3. Berners-Lee, T., Masinter, L., McCahill, M., *Uniform Resource Locators (URL)*, Network Working Group RFC 1738, December, 1994. See also <u>ftp://ds.internic.net/rfc/rfc1738.txt</u>.

4. Berners-Lee, T., *Universal Resource Identifiers in WWW*, Network Working Group RFC 1630, June 1994. See also <u>ftp://ds.internic.net/rfc1630.txt</u>.

5. Berners-Lee, T. and Connelly, D., *Hypertext Markup Language* - 2.0, MIT/W3C. Sept. 1995. See also <u>http://www.w3.org/pub/WWW/MarkUp/html-spec/html-spec\_toc.html</u>.

6. Bobrow, D. and Winograd, T., *An overview of KRL, a knowledge representation language*, **Cognitive Science**, 1(1):3-46, January, 1977.

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

8. Keene, S., **Object-oriented programming in Common Lisp: a programmer's guide to CLOS**, Addison Wesley, Reading, MA, 1988.

9. Kunze, J, *Functional Recommendations for Internet Resource Locators*, Network Working Group RFC 1736, February, 1995. See also <u>ftp://ds.internic.net/rfc/rfc1736.txt</u>.

10. Liskov, B., et al., CLU Reference Manual, Springer-Verlag, New York, 1981.

11. Marshall, C. C., et al., *Aquanet: A hypertext tool to hold your knowledge in place*, **Proceedings Hypertext '91**, ACM New York, December, 1991, 261-275.

12. Meyer, B., Eiffel: the Language, Prentice Hall, New York, 1992.

13. Mockapetris, P., *Domain Names - Concepts and Facilities*, Network Working Group RFC 1034, November, 1987. See also <u>ftp://ds.internic.net/rfc/rfc1034.txt</u>.

14. Mockapetris, P., *Domain Names - Implementation and Specification*, Network Working Group RFC 1035, November, 1987. See also <u>ftp://ds.internic.net/rfc/rfc1034.txt</u>.

15. Nelson, T. H., Literary Machines, The Distributors, South Bend, IN, 1988.

16. Digital Equipment Corp, et al., **The Common Object Request Broker Architecture and Specification**, OMG Document Number 91.12.1, Rev. 1.1, Object Management Group, John Wiley & Sons, New York, 1991.

17. OSI, ISO9594 and CCITT X.500 Directory Services.

18. Postel, J. and Reynolds, J., *File Transfer Protocol (FTP)*, Network Working Group RFC 959, October, 1985.

19. Raggett, D., *HyperText Markup Language Specification Version 3.0*, Internet Draft, draft-ietf-html-specv3-00.txt, March, 1995. Note: This is a draft and expires in September, 1995. See also <a href="http://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-html-specv3-00.txt">http://ietf.cnri.reston.va.us/internet-drafts/draft-ietf-html-specv3-00.txt</a>.

20. Sollins, K. and Masinter, L., *Functional Requirements for Uniform Resource Names*, Network Working Group RFC 1737, December 1994. See also <u>ftp://ds.internic.net/rfc/rfc1737.txt</u>.

21. van der Linden, R., *An Overview of ANSA*, AR.000.00, Architecture Projects Management Ltd., May 1993. See also <u>ftp://ftp.ansa.co.uk/phase3-doc-root/ar/APM.1000.01.ps.gz</u>.

22. van der Linden, R., *The ANSA Naming Model*, AR.003.01, Architecture Project Management Ltd., February, 1993. See also <u>ftp://ftp.ansa.co.uk/phase3-doc-root/ar/APM.1003.01.ps.gz</u>.

23. Van Dyke, J. R., *Link Architecture for a Global Information Infrastructure*, MIT/LCS/TR-659, June, 1995. See also <u>http://ana-www.lcs.mit.edu/anaweb/pdf-papers/tr-659.pdf</u>.

\* Acknowledgment: This work was supported by the Department of Defense Advanced Research Projects Agency, under contract number DABT63-92-C-0002. We would like to acknowledge significant contributions to the Information Mesh Project as described above by: Bienvenido Velez-Rivera, for his work on roles, Alan Bawden, Timothy Chien, and Matthew Condell.

Note: Authors are listed alphabetically.

## **Appendix A: The Object-Role**

### A.1 The Basic 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
```

### A.2 The "Composite" Additions

The documentation associated with a given object.

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. Note that the actions for adding components to an object are specific to particular roles and do not appear in the general object-role. As with all actions, these will be invoked by any client of the object that is allowed to modify it, such as most likely its original owner.

Additional action:

```
(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.
```

### **Appendix B: The Link-Role**

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.
```

```
(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 a descriptor pointing at or into the exposed abstract
structure of an object.
```

Makers:

```
(create oid implementation endpoint-list) Required Create a link.
```

### **Appendix C: Sample Link Subroles**

#### C.1 The Named-Link Role

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.

#### C.2 The Binary Link Role

```
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.
```

Makers:

```
(create oid implementation endpoint1 endpoint2) Required Create a binary-link.
```

#### C.3 The Ordered-Link Role

Ordered-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.
```

### **About the Authors**

Karen R. Sollins <u>http://ana-www.lcs.mit.edu/people/sollins</u> M.I.T. Laboratory for Computer Science 545 Technology Square Cambridge, MA 02139 sollins@lcs.mit.edu

Jeffrey R. Van Dyke <u>http://ana-www.lcs.mit.edu/people/jvandyke</u> Trilogy Development Group 6034 W. Courtyard Dr. Austin, TX 78730 jvandyke@trilogy.com