Extending the Network Abstraction to an Information Infrastructure: The Information Mesh

Karen R. Sollins
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139
sollins@lcs.mit.edu

February 2, 1996

Abstract

This paper is focussed on two central ideas. The first is that it is valuable and important to extend our model of *the network* to a higher level of abstraction and generality than has been provided heretofore. The second idea is that the level of abstraction to which the network should be extended should provide an extremely simple yet powerful network-based object model. The goals of such an infrastructure should be not only those commonly used such as ubiquity and support of some model of application heterogeneity, but longevity, mobility, evolvability, and resiliency to failures and unpredictable behavior. We describe the Information Mesh, such an information infrastructure architecture and address the issues in realizing it. This work is put into the context of ongoing research and development in the field.

Keywords: Information infrastructure architecture, network abstraction.

1 Introduction

In their simplest form, networks enable the communication among applications or programs running on disparate computers. The applications may themselves be executing on behalf of humans, thus enabling human communication, or perhaps communication between humans and remote computers. In this paper, we will explore the model of the network available to applications, and a particular enhancement of that model.

Consider the provision of communications or transport to the applications. The TCP/IP suite of protocols is the most widely used, probably throughout the world. In order to support it, each site must provide IP over which sits TCP and UDP. These provide bit-streams with various characteristics. Above that there are a variety of more application specific transport and communication protocols such as FTP[35], Telnet[34], SMTP[33], and HTTP[6]. Each of these protocols can be described as providing enhanced abstractions for communication and exchange. FTP supports files, allowing for both learning about them and more importantly exchanging or copying them. The communicants are processes operating either on behalf of humans directly or by means of another intermediary, such as a web browser. Telnet provides character streams again operating on behalf of a human in communication with a remote operating system and set of applications. SMTP provides for the various activities related to delivering mail messages, on behalf of a local mail services. SMTP delivers mail to particular mailboxes, so it will verify that the destination mailbox is known or forwarding can be handled by the recipient. HTTP exchanges HTML[5, 36] documents. In this case, it will be invoked by a "web client" and transmits its messages encoded in MIME[9, 26] types.

In each of these cases, it is important that the protocols be considered, as they are, part of the protocol suite. The reason for this is that there are cross-cutting issues that should be addressed through mechanisms that span more than one layer of protocol. The most well-known of these cross-cutting issues is security, but there are problems in naming, functionality, and performance that also can only be addressed effectively in the context of a whole protocol stack.

With these issues in mind, it is important to consider the requirements and needs of applications and whether the supporting protocols meet them. Each of the application level protocols supports fairly specific sorts of *currency* exchange. The currency of a protocol is the sort of units or objects that are being exchanged. In FTP the currency is files whose content is limited only by the imagination of the applications developers. In SMTP the currency is mail messages. In HTTP the currency is HTML documents limited by what can be expressed in HTML, and in Telnet the currency is character streams.

At the same time we see several apparently contradictory trends in applications domains. The first of these is the survival of legacy software. This is software that may have been expensive or difficult to develop and is not given up easily. It often solves very specific problems and to date generally has been built on very specific models of data communications, and sharing or exchange. In contrast, we also see dramatic new directions for applications. New sorts of media, such as animation and video, 3-D rendering, etc. are allowing applications to take off in completely new directions. In addition, new models of communication in support of various group interaction models also are having a major impact on the breadth of scope of applications.

This paper makes the two-fold argument that first a more general *currency* model should be supported as part of our vision of the network and protocol stacks embodying it, and second that we can define a particular such model and identify the set of core services and protocols needed to support it. The paper proceeds by first identifying a core set of requirements derived from application domains, in Section 2. Section 3 presents an overview of the Information Mesh itself, our proposed model. This section lays out the model itself, and Section 4 identifies the core services and protocols needed to realize the Information Mesh model, with a detailed example presented in Section 5. To put this work into context, it is important to review not only the derivation of our ideas, but also similar work that is being done elsewhere, such as the World Wide Web, CORBA and OLE and how our proposal differs from those efforts. This is done in Section 6. This is followed by a brief summary of the current work on this project, in Section 7. Finally, the paper concludes with a discussion about the coupling of this model to the supporting protocol layers.

2 Needs of application domains

The need for an information infrastructure is driven by the needs of applications and application-building environments. It is in this context that we see three significant threads of evolution: national and international scaling of the applications themselves with the development of widely distributed applications; heterogeneity and the complementary need for stability in a heterogeneous and evolving environment; longevity with the need for the ability to provide long-lived identification and complementary location facilities, as well as an evolvable typing model and the ability to build and maintain long-lived relationships among objects.

Networks and distributed systems have been growing by orders of magnitude in the sheer size of the network, as well as scope, reaching to previously inaccessible spots of the earth. As each new Eastern European nation has become independent, it wants to join the networking community, providing email and Web access. Penetration into previously inaccessible regions is being provided by mechanisms such as an electronic funds transfer product for developing countries, originally designed for South Africans out in the bush.[1] There is deep penetration in Asia ranging from a widespread computer industry to the more specific governmentally sponsored project on an information industry such as in Singapore.

We continue to develop in a world of evolving heterogeneity. There is a multiplicity of transport technologies, protocol suites, programming and runtime communications paradigms, and applications support services. Finally, 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, the computing universe is expanding. Second, longer-lived information will often outlive the original application from or for which it was created. Both by being more widely available and independent of specific applications, the information becomes more valuable to society as whole. Therefore, building a set of relationships among such objects cannot be constrained by the local naming and addressing generally used at present.

These factors lead us to a set of specific goals for an *information infrastructure architecture* that sits between the applications and our traditional models of network transport protocols. This infrastructure falls into two parts, a core that is domain independent, and domain specific facilities that embody general knowledge and models specific to particular domains. For example, the naming and searching needed to manage a combination of private and corporate phone books may be very different from what is needed by a library providing access to books, videos, and music. but supporting both may be location translation services that map from globally unique identifiers to locations.

It is in this context that we define a set of goals for the Information Mesh Project:

- **Longevity**: Both information and identifiers for the information should be able to survive indefinitely; this means that, at some level of abstraction, an object may exist for a period measured in human lifetimes rather than computer or technology lifetimes. We like to think in 100 year lifespans for objects.

- **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.

- **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 via a particular path or access method. 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.

- **Support of relationships**: The Information Mesh must support the ability to express relationships among objects within the Mesh. There must be some mechanism for expressing the nature of a relationship or link. One must be able to relate more than two endpoints if desired in a single link, and there must be the ability for an object in its capacity as an endpoint to expose its structure for potential linking. For generality and expressiveness one must be able to use a link as an endpoint of another link. It is mandatory that one be able to express that an object is a composite and identify its elements. Finally, all the other requirements for the infrastructure must also apply to links. Thus, for example they must be able to span any geographic or administrative scope and must be able to survive both for an extremely long time and in the face of mobility of their endpoints.

- **Ubiquity**: The Information Mesh should provide support for network-based applications accessing information that is distributed both physically around the world and administratively across regions of differing management policies.

- **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 advantage 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.

- **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.

It is the first five of these, longevity, mobility, evolvability, resiliency, and our particular emphasis on linking that distinguish this work from the many other apparently similar efforts. Of these, longevity is probably the most important. Support of mobility and evolvability are important even in the near term, but without specific engineering attention become insurmountable problems in the long term. Although most systems pay lip-service to resiliency, it is only when one truly accepts the plethora of potential failure modes in a global network environment that the need for resiliency becomes clear. Our requirements for linking are that links must not only meet all the other Information Mesh requirements, but also provide rich abstraction and functionality.

The other goals of ubiquity, support of heterogeneity, homogeneity, and minimality, although common to many other similar projects are equally important in this effort. It is in these goals that we embody the issues of broad scope and utility. It is also here that we express the need for any such effort to be simple yet expressive enough for application and tool builders to use as an effective substrate.

3 The Information Mesh model

The model in the Information Mesh is extremely simple, powerful, and expressive. It achieves this by specifying only those aspects that are needed for commonality and communication, leaving out anything that might limit the options of designers and programmers, especially where differences of opinion have proven to be strong. The model itself is an object model, and as such addresses the issues of naming/location and typing. In a general infrastructure model there are three other significant areas that are commonly addressed, but that are missing from our model: a communications model, a security model, and a transport model.

The question of how programs communicate with each other is traditionally a significant part of any distributed system. SunRPC[43] and DCE[15] support various forms of Remote Procedure Call (RPC). ISIS[7] provides causal invocation and group communications (a form of multicast) as does PSynch[32]. Message passing is more common in strict object oriented systems, such as ANSA[44, 45, 37] or CLOS[18]. Each one of these provides a distinct and valuable communications model; it is exactly for this reason that it does not make sense for the Information Mesh, if it is to be generally useful to a broad spectrum of clients, to dictate a single communications paradigm, but rather must support only those aspects of network-based and distributed systems that are useful in common among them.

On the other hand, in spite of not supporting a particular communications model, there are mechanisms that might valuably be provided by the infrastructure. One of these is the ability to express the scope of a communication. Whether the activity is a search for a document or some form of group communication, it may be desirable or necessary to limit the logical (or perhaps physical) scope. This is an area where we have done very little work, but expect more in the future. It will have interesting couplings to interactions with underlying transport and policy mechanisms.

There are a number of policy mechanisms, such as authentication and certification services, access control, and financial transaction mechanisms that must be a local administrative decision. In each of these cases, there may be support the infrastructure can provide, such as the place holders in protocols for certificates, but the Information Mesh should not be making a single decision that should suffice for all, since administration requirements will vary so widely, and presumably evolve with time as well. Decisions about both policy and mechanisms for security must be local and should not be dictated by a global infrastructure.

That being said there are some aspects of these problems that can and should be addressed at the level of the supporting infrastructure. For example, given that we are taking an object oriented approach in our core model, it follows from that, in truly object oriented environments, that each object will be its own determinant of its own access policies. We have been involved in work addressing this problem[39, 10] from the object oriented perspective, but recognize that the problem is more complex than that. Objects may reside within administrative domains, perhaps nested domains, each of which claims to have at least some right to controlling access to the object. Given that more realistic model, we can and are currently addressing the problem of the nature of the protocols that might support such a model. Such a protocol should not and will not define choices such as which mechanisms or specific policies should be utilized in any specific case, but rather a protocol to achieve collaborative and nested security policies provided with authentication, encryption, and one-way functions as needed and desired.

The third concern here is what to use for transport. We see several examples of infrastructure models such as the World Wide Web[3] and Prospero[28, 29] taking advantage of a suite of transport protocols. In both of these cases, the protocols supported by the models are higher level protocols, such as HTTP[6], FTP[35], and Gopher[2], as well as several other file access protocols (AFS and NFS). A set such as this will only grow, and therefore selecting only one or configuring into a system even a particular set would severely hamper the utility of an information infrastructure.

There is a closely related problem in transport, that of what is known in the OSI layering model as the *presentation* protocol. Here the question is what is the form into which objects, information, or data are marshalled for transmission. We see several examples of such protocols available and in use, such as ASN.1[14], XDR[42], and MIME[9, 26].

As with the other issues in transport, neither is there one provably correct choice nor is this a stable situation in which no new protocols will be developed to solve yet unknown problems. Evolution must be supported. Of course, there are likely to be preferences for particular protocols, and many locations will only support a limited set of both transport and presentation protocols. With this in mind, a simple negotiation protocol is needed to agree on which protocols will be used. Such a protocol itself will need to be based on a protocol supported by the endpoints of a communication, simply as a starting point, although they may agree during a negotiation that they will use a more efficient transport/presentation protocol. By providing a negotiation protocol rather than a single or limited set of transport protocols, we have allowed for evolvability and mobility, so that objects can move to new environments, supported by different transport protocols, without losing the ability to access such objects.

The remainder of this section falls into three parts, a discussion about naming and location, a description of the Information Mesh type model, concluding with a description of the design of linking in the Mesh.

3.1 Naming and Location

In general, there are three functions for which names are used: identity, knowledge gathering, and location. The identity function is that of distinguishing one object from another by means of the name. Thus two sets of information that are intended to be equal (by some definition of equality) will have the same name, whereas those intended to be distinct will have different names. This will allow one to recognize when two higher level "nicknames" refer to the same object and to distinguish objects in the infrastructure when necessary or useful.

The knowledge gathering function is one of learning facts about an object based on its name. One must ascertain the correctness of such information as well as the utility of it. For example, simply because an object name has a terminating name component such as ".c" does not necessarily imply that the object is C source code, although it may imply that someone thought that was the case. Furthermore, inclusion of such terms as "new" may only be relative. In many cases the inclusion of such semantics or knowledge in a name is achieved by organizing the names hierarchically and assigning some meaning to the elements of the hierarchy. Such hierarchies have the same problem of being unverifiable; for example, simply because a document resides in a piece of the hierarchy named "Information Mesh" does not guarantee that its contents are relevant to, or part of, the Information Mesh, but only that someone or some program stuck the object there. Hierarchical names that include semantics have the added problem that there is no particular reason to believe that all such semantics in component names naturally fall into a single, self-evident hierarchy. Thus, the ordering of the components may not be obvious, nor even relevant for all uses of the name, especially over a long time. The final, and perhaps most important issue with embedding knowledge in names is that such information may change. It may well be that originally a program was written in C, but as it evolved it was rewritten in C++ or something else. If the name is to last as long as the object, almost any knowledge one might cast into the name may change, and thus should not be included in a name used for identity that is to survive as long as the object itself, especially if those names will be widely dispersed throughout the network. Thus, in the Information Mesh, the functions of identity and knowledge gathering have been separated.

Embedding location or access information in a name has the same problem as embedding other semantics. If the name includes location information, that is very likely to change for any object that may survive 100 or more years. Even for archived books in a library, the vast majority of them will be moved because the maintenance of the building, shelving, organization of the books, and books themselves will cause them to be moved to another location for at least some of such a period of time. In the case of electronic information, technology is changing much more rapidly than in the library world and it is extremely unlikely that anything that is currently part of the support structure (protocols, addressing schemes, host naming schemes, file systems, etc.) will survive for that period of time. Thus, including valid location or access protocol information in a name will mandate that the name must change and for identification purposes, names cannot. Given this contradiction, in the Information Mesh, the functions of identity and location have been separated. It is valuable to understand that what is considered a location at one level of abstraction will be considered a name at a lower level of abstraction. The abstraction barrier allows us to distinguish where it is a location and where a name.

Given these arguments, the Information Mesh posits that the name or names (*OIDs*) for an object are used solely for identification and can have a lifetime as long as or longer than the object itself. This also implies that these names will not be reused because there is no way of knowing whether such a name has been hidden somewhere to be found at a later time. It is important to be able to recognize that two references are to the same object based on the identifier. It is not difficult to imagine a number of name assignment schemes that will support such uniqueness. For example, we know that we can distinctly identify any host on the internet by a combination of protocol suite and address or name within that scheme. Combining that with a time stamp and reliably increasing counter, one can generate unique identifiers, assuming that duplicate host identifiers are not assigned. If such a naming protocol and hostname within it are not reused or if they are then some guarantee is provided that locally generated ids will not be reused, uniqueness can be provided. Within the IETF such names are called Uniform Resource Names (URNs)[40]. We use this term interchangeably with object identifier or OID throughout this paper.

Given that names will be free of semantics, they cannot directly reflect location. Location information may well change with time for technical or administrative reasons. For example, as storage technology changes, it may become either necessary or more desirable to move an object to another location. If a server technology is no longer supported, all information stored on servers of that technology will need to be moved. If new technologies are developed that store certain sorts of information more efficiently, from either the client or server perspective, the information may be moved. If information changes ownership or management or the management decides to organize information differently on storage services for any number of policy reasons, the information may move. Thus, in order to find an object, the best the Information Mesh can do is provide *hints* to help in the location process. Hints may range from the specific to more and more general. It is probably the case that the more specific a hint the shorter its lifetime or half-life. For example, the simplest hint is an address at which the object might be found. This is probably a location at which the object was recently found. It is this kind of hint that will change most frequently.

We assume that if and when an address hint fails, the client will resort to some sort of location translation service (aka location service) that helps in finding locations of objects given their OIDs. Thus, another sort of hint is the address of a location service. If the object moves, it may well report back to one or more location services about its new address. Hence, an address for a location service should be reasonably helpful, that is, until the location service moves. A more useful and longer lived hint would be an OID or URN for the location service. At this point, in order to find an address for an object, a request would go out to translate a location service URN into an address, followed by a request to the location service for translation of the URN for the object into a location for the object. After all this, the object itself can be accessed.

It is always possible when querying a location service, that that service does not have the answer to the query. In cases of this sort, or even if it does have an answer, a location service may also provide suggestions for alternative location services to try, or hints for selecting them. For example, a location service might provide an answer, knowing that there is a plan for it to go out of service in the near future, so it will suggest that any further requests should be submitted to an alternative location service. Hints may also help locate an instance of an object that is in a desirable form, for example supports the desired functionality and on a host that supports the most desirable transport protocols. It is the hints for locating an object that allow for finding an object as it moves through the net and as storage, support and transport technology evolve.

From this description, it is clear that OIDs or URNs and the supporting hint mechanism address several of the original goals laid out for the Mesh. In order to support global identification of objects, OIDs are intentionally globally unique and because they are free of any semantics will outlive any changes in semantics that frequently invalidate names in our current naming schemes. The combination of OIDs and hints allows objects mobility as well as supporting multiple instances reflecting among other things a heterogeneous set of transport protocols. By separating this distinction from naming, a homogeneous model for naming is provided to the applications environment. At the same time, the choice here is for a simple separation of function and mechanism supporting it, while allowing for full flexibility of choice in mobility, transport, and evolution as reflected or not in OIDs themselves.

3.2 Roles: a typing model

It is important to be able to recognize the behavior of an object. An object may exhibit more than one behavior. It may be a book to one client and a location service to another, as for example a phone book is. As a book it has pages, a title, a publisher and perhaps a table of contents. As a location service it has entries that provide a translation from a name to a location for certain sorts of objects. Thus we say that the object plays two *roles*. An object can play more than one role at a time and the set of roles that it plays may change with time.

There are three aspects to the definition of a role: *actions*, *parts*, and *makers*. For each of these there may be both optional and required components. In addition, a role definition may inherit from one or more other roles. The object role definition in Appendix A provides an example, as do the additional roles in Appendix B. An implementation of a role must support all the required components, but need not support the optional ones.

The actions define the abstract functionality of the role, and as such are defined in terms of the signatures of the functions. These definitions are abstract in that the choice of a role supporting particular actions does not imply the choice of an implementation of those actions. This is important not only to allow for implementations in different languages, but even within the same language, as an example there may be situations in which different algorithms will be needed for varying demands on performance. Thus, the definition of which actions can be invoked on an object is separated from the realization of those actions in executable or compilable code. The tags of required and optional on actions carry more meaning than which ones must or may be implemented. In addition, when a subrole inherits from a superrole, it must inherit the required actions, and may also inherit the optional actions.

The parts define the abstract structure of an object playing the role being defined. By defining parts abstractly, they are not constrained by any particular representation. Thus, for example, the table of contents of a book might be represented in one case by an array of the entries, while in another case it might be represented by a linked list. In a third case, the local programming language may have its own type abstraction model and the representation may remain hidden from the programmer using the role. In addition, in specifying the parts of a role definition, one must also specify constraints on the nature of the selectors when a part definition reflects the potential for more than one part of that sort. Thus, if a part of a book is page the selector will be numerical, while the selector for appendix may be ordered, alphabetic. The selectors may also come from a completely arbitrary, unordered set. One of the issues with respect to selectors is whether they can be known and predicted by potential users of them, or whether they can only be generated with the appearance of randomness by the object itself. There are subtle security issues with exposing the algorithms for generating selectors or making them guessable.

Lastly, we separate out the activity of creating new objects playing a particular role. These functions fall into the group of makers. They do not operate on objects that play the role, but rather may take other sorts of arguments (or none) to create a new object playing the specified role.

By abstracting away from a particular implementation, Mesh roles allow for multiple implementations, hiding from the network-based client of an object details and specifics of any particular implementation. It is exactly those abstract features defined by a role that are accessible via the network.

Role definitions form a multiple inheritance hierarchy. At the root is the object role, defined in full in Appendix A. Beyond that there are a small set of kernel roles, which must be builtin in order to support the Information Mesh. Each role is an object in the Mesh and as such has a URN and plays the role role. We will discuss roles further when addressing linking.

There are several important features of the role model. First, it is important that it be flexible enough that with refinement it support existing type and class abstraction models. Thus, it needs to allow for both functionality and structure. In addition, multiple inheritance and polymorphism are important. Second, evolution is important. Thus the ability for an object to be able to report the roles that it supports is necessary; this implies that a client must be able to ask an object which roles it plays at any given time. Third, in order to support the heterogeneity and anarchy in policy decision making that is currently the case in the Internet, and will become even more prevalent, both multiple implementations and optional parts and actions are necessary. Of course, this will often make it difficult to *guarantee* anything about an object, but not much will be guaranteed in this widely distributed, anarchic environment. The application must be prepared to be defensive, but we must do the best we can to provide a reasonably stable comfortable environment for the applications builder and applications themselves on all fronts.

3.3 Linking and composition

With the simple object model as a basis, building relationships among objects becomes a matter of applying the object model appropriately. It also provides the reader with a further example of the use of the model, through the effective use of inheritance in the definition of roles and through the use of OIDs.

The solution for links is that they are first class objects in the Mesh universe. As such they play roles, and in particular they all play the superrole of link. Appendix B defines the basic link role and several subroles that inherit from the link role. It is worth noting here that based on the subroles of Named Link, Ordered Link, and Binary Link defined in Appendix B, we could then define link roles to support many of the popular linking models. References [41] and [46] present the details of this work.

In these definitions, it is the parts that are critically important. The reason for that as we will see below is that when linking into the structure of an endpoint, it is the abstract structure of the endpoint that determines the granularity and nature of the exposed structure. If we look at an infrastructure model such as the World Wide Web defined by HTML documents, we find no such abstraction. If one wants to link into the structure of another object, one does this by linking to a region defined with an arbitrary string name that is determined by declaring an anchor around some region in the internal representation of the document. If that object were ever to be transformed into some other representation, there is no guarantee that the meaning of the defined region would or could survive. Although we have exemplified this with HTML documents, it is not uncommon in hypertext models in general, such as the Dexter model.[12]

A link consists of one or more endpoints specified as the parts defined by a role the link is playing. The realization of an endpoint is a descriptor. A descriptor includes a URN and a qualifier for that URN. The qualifier may specify the whole object or some view or component exposed through a part of one of the roles the object plays. A tag specifies which part of which role is being specified and the selector identifies the particular part if more than one is available. There are a number of issues embedded in this link structure. Again, these have been addressed and published in further detail in [41] and [46].

By using URNs the problems of identity and mobility are addressed. Since links are first class objects, they have identities and therefore can be used as the targets of links. In addition, since URNs are independent of location, unlike the URLs used in WWW links, the endpoints are free to move as needed without voiding the meaning of the endpoints and therefore the link.

Now consider the problem of composition of objects, as exemplified in Figure 1. We would like to be able to specify that spreadsheet A is a figure in document B. Furthermore, without the inclusion of A, B is incomplete; it does not tell the whole story that its author intended. This composition is expressed in actions, as can be found in the object role. There we have included the action get-required-objects. This is an optional action, whose complementary actions for declaring required components are specific to particular roles. Only if this optional action is inherited and then supported in an implementation will a component organization be possible for an object in a particular role. But, it readily falls out of the role inheritance model. Again, this aspect of building relationships is more complex than can be described here, but is given more detail in [41] and [46].

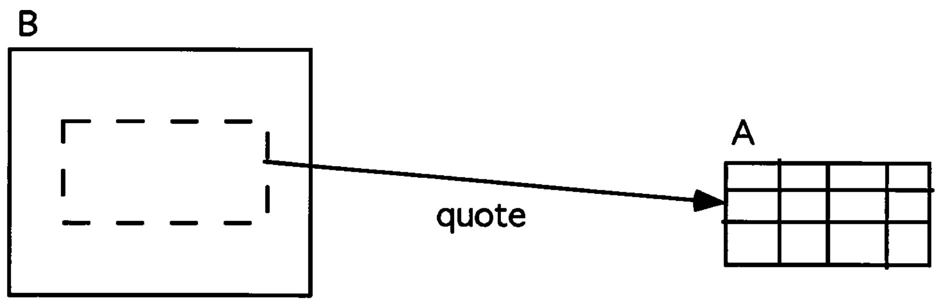

Figure 1: Building a composite relationship.

4 Core services

In addition to the components of the central model of the Information Mesh, there are a set of core services that are needed in order for the Mesh to function. We will only identify these here without significant detail, for lack of space. Each of these descriptions only identifies the basic role for a service of its sort. There may be many servers playing each role. In addition, there may be subroles of each of these roles, that provide more functionality than is described here, and hence, an object or service may play a richer role that provides for more than simply the base role. For a server to provide one of these services it must support the basic role of that sort. The role definitions are too lengthy for inclusion here.

- **Naming**: A naming service actually performs two functions, which may be separated. First, it is necessary to determine whether a new or previously assigned name is to be issued; in other words it must be determined whether or not the object being named has been previously assigned a name by this naming service or not. Second, a URN will be created and assigned when needed.

- **Location translation**: A location translation service maps URNs to transport level addresses. It may also, either in addition or instead, return hints, if it cannot translate a URN, as a means of further aiding in the search to find an object given its URN.

- **Object management**: An object management service handles the business of creating, storing, and providing access to an object.

- **Role library**: A role library provides storage and therefore access to role definitions. It may also provide implementations of roles.

- **Boot service**: In order for a new participant host to join the Information Mesh, it must be able to find such information as the location of instances of the above services. This can be done either by assigning them well known addresses in protocol suites, e.g. port numbers in TCP/IP or by providing at the boot time of each node joining the Mesh the address(es) of one or more boot service, not necessarily on well-known ports. By postulating boot servers, we can limit the amount of information that must be well known to only that needed to find a boot service.

There is a close relationship between this set of core services and those enumerated in the IAB Workshop on an Internet Information Infrastructure Architecture[23] as we were active participants in that workshop. The set identified there is broader than this set because it includes services and functionality that we would judge not to be in the core, although quite important. In particular, we have not included here services that need to be application domain specific, such as resource discovery and billing.

As discussed earlier, specific security services and transport are not part of the Information Mesh project, although aspects of these are addressed in supporting protocols.

5 An example

Let us consider an example briefly. In this case, as in Figure 2, we will assume that Client A wishes to replicate the object with URN B locally. It should be noted that because everything is an object in the Information Mesh, the protocols used for communication are determined by the defined and implemented roles played by each of the participants in this activity. The participants are the client A (whose roles are not identified here because they are extraneous to the example), the object B whose role supports the ability to transmit a copy of the object via the transmit-object action, and location translation services C and D, which because they are playing the role of location translation service support the action map-urn. In addition, initially A does not have the definition or implementation of the role R that B plays, so it will need to retrieve it from the library L. The implementation of R it receives supports the receive-object action.

Finally, there are several other facts that we must set in place. First, B resides at ADDR2. A has several hints for finding B. They are:

1. ADDR1, an address at which object B was once known to reside;

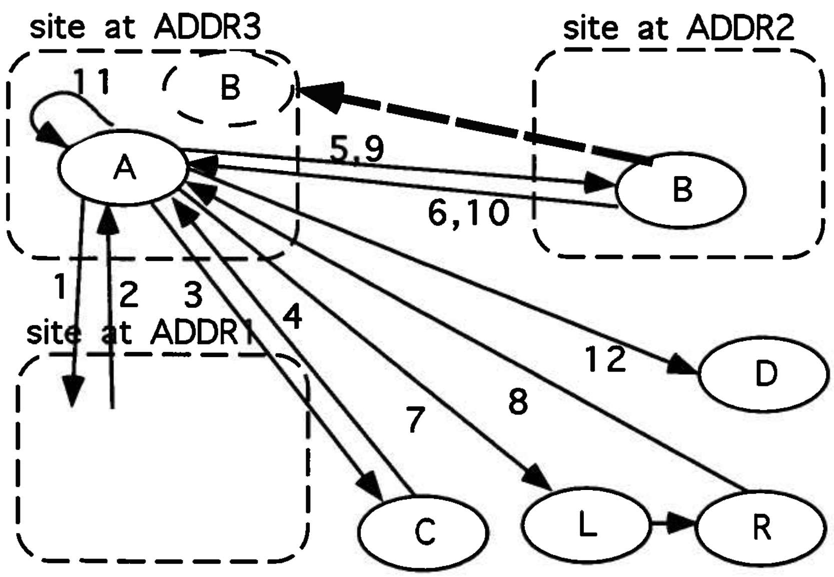

Figure 2: An example of the steps involved in making an authoritative copy of object B at the site of object A.

2. C, a location translation service believed to know about B's location;
3. D, known to be an authoritative location translation service for B.

Figure 2 lays out the steps to be followed.

1. A sends a request to site ADDR1 for B asking which roles it plays.
2. A receives back a response that B is not known at that location.
3. A sends a request to the next hint on its list, the location service C. It may even have made a decision about the ordering of requests to C and D, knowing that C generally provides faster service. In this case it requests resolution of B.
4. C sends ADDR2 to A.
5. A sends a request for any roles played by B to ADDR2.
6. A receives a message saying that B plays role R with hints that R is known by library L including L's location.
7. A sends a request to L requesting a copy of R, both definition and implementation.
8. A receives back the definition and implementation, inserting or otherwise handling them appropriately.
9. A now requests from B at location ADDR2 a copy of itself.
10. A receives and installs the copy of B locally.

11. A modifies its set of hints about B reflecting both a local copy and the fact that there is a copy at ADDR2 but not at ADDR1.

12. A reports to D the facts that B is at ADDR2 and ADDR3, but not at ADDR1. D may decide to verify ADDR1 and ADDR2 independently.

In this example we see several important features of the Information Mesh. First, objects can move and the environment has the resiliency to track them. By binding to location only at the times that that is needed, we can put fewer constraints on mobility. The mutable and tunable set of hints allows for a resilient and configurable backoff mechanism in the face of potentially increased mobility. Second, it is possible to support environments in which dynamic loading of implementations allows an application to learn about the roles objects play, find them, and take advantage of them. Third, the roles an object plays may change with time. Because roles are objects as well, we can take advantage of the existing mechanisms for managing and finding objects to do the same for roles, thus allowing applications to learn about the evolution of the sorts of objects with which they interoperate. Fourth, it is trivial to allow for third party solutions to any of the components. If the minimal roles are defined, anyone can get into the business of providing realizations of them with particular additional features. There is no single, authoritative, or proprietary solution to any component of the infrastructure, yet at the same time there is opportunity for anyone to provide specialized and perhaps proprietary services as they wish.

6 Related work

There have been several major categories of influence by other work on the Information Mesh. The first of these is the author's previous work in both distributed systems and naming problems. The work on the type model was influenced by a number of activities. The Mercury Project [21] allowed for a variety of communications paradigms ranging from message passing to simple RPC to multiple, sequenced outstanding remote invocations, as well as a global typing scheme. Work closely related to this was the MLP project at the University of Arizona [13] and Matchmaker [16] at CMU. These projects all dealt with the problem of exchanging information when different programming paradigms existed at the two ends of a communication. In addition, the Object Management Group's recent work on CORBA[30, 31] providing both multiple inheritance and polymorphism, as well as the polymorphism provided by a language such as Eiffel[24], in addition to the type systems of Clu[20] and CLOS[18] led to our generalization of typing into the role model described above. Microsoft's OLE[25, 31] in the Component Object Model (COM) takes a different approach. It supports single inheritance between interfaces in a very limited way, and discourages its use. It supports and encourages the use of aggregation of interfaces, each perhaps more limited and simplistic than those of CORBA or the Information Mesh. Such an aggregated interface may directly export the interface of a component object rather than incorporating it into the interface more abstractly. Microsoft believes that this provides a more manageable solution to the evolution of base interfaces.

There also is both significant previous work by the author and others on the problems of separating naming and location. The thinking in the Information Mesh was influenced by the author's previous work on distributed naming [38] as well as the work of others on object oriented distributed systems, such as Emerald [17] and ANSA[44]. These had an early effect on the object model and the separation of identification from location of objects. In addition, the thinking in this project has been strongly influenced by both individual large information systems, as well as participation in the Internet Engineering Task Force (IETF)[40, 19, 4, 23] working groups on information infrastructure, as well as the large hypertext systems of the World Wide Web[3] and Project Xanadu[27]. There are clear distinctions between these works and the Information Mesh, for example, in terms of separating location from identification. In addition, the WWW is focussed very much on the present in terms of other mechanisms and does not provide anything akin to roles. WWW identifiers include not only location information, but also protocol or access method identification. Xanadu is based on the assumption that one must buy into not only their model but their specific technology, not promising for an open heterogeneous and evolving environment. The IETF is in the process of standardizing an information identification architecture. This is a collaboration among participants from the networking, information infrastructure, and library communities, as well as other interested participants. Our service model influenced the IAB Workshop on an Internet Information Infrastructure Architecture.[23] model. Both ANSA[45] and CORBA[30, 31] provide federated naming. In ANSA names are defined within domains; each domain has a name for all other domains as well. Hence to name an object in a remote domain, one concatenates the local name for the remote domain with the name of the object within that remote domain. This has the feature that domains can evolve completely independently, but it has two significant problems, based on the fact that location information has now been embedded in the names. First, a name must be translated on transmission across domain boundaries. If it is encapsulated in email, or worse yet written down on a piece of paper and handed across such a boundary, it may be impossible to discover that translation should happen. Second, if an object moves across a domain boundary, the location information embedded in the identity of its domain becomes invalid, but it is extremely difficult to guarantee that all copies of that name are corrected. In CORBA the domains are defined by Object Request Brokers or ORBs. Each ORB has a globally unique, not relative, name, avoiding the first of these two problems. But the second is also applicable to CORBA. Extreme longevity and mobility are not well supported in these systems. Although object reference is mediated through a COM in OLE[25, 31]; there is no cross COM reference supported at present.

The linking model was influenced by a number of works. In particular, part of that effort was to accommodate the features as many different models of linking as possible. Of particular concern were the models of the Web[5, 36], Xanadu[27], the Dexter model[12], and Aquanet[22], as representative of various models of linking. None of these provides the full flexibility and generality provided by the Information Mesh, but it in turn does support all of them. See [41, 46] for the full detail of these relationships.

Finally, the negative influence of recognizing the need for a variety of communications paradigms such as that of Mercury[21], ISIS[7] and Psynch[32], the ANSA communications paradigm[37] and traditional RPC[8] caused us to reject the idea of a single, pervasive communications paradigm.

7 Current status and future work

To date much of the implementation of the ideas described here has been in a prototype and demonstration environment written in Scheme and running on Unix on both DEC/MIPS processors and Alphas. The demonstration core provides a naming service, hint and supporting hint servers, location translation, and a role library. Demonstrations have included importing all of the Gnu Info Tree[11] into our environment, and making the Gopher[2] universe accessible through a gateway. In addition, the role library as a set of browsable objects is also accessible. A web server has made these accessible through unmodified web browsers.

At this point, we realize that the pervasiveness of the World Wide Web makes that the right environment in which to present our ideas. This is being done in two ways. We are in the initial stages of building Web servers to provide some of the services. The first will be one to allow for URLs to be used as though they were URNs, as much as possible. This service will translate and track URLs, learning about new locations of objects and allowing for both mobility and replication. It will also be a provider of real URNs to any customers desiring them. URLs will always have some problems as URNs, so we would like to encourage the use of true URNs. A second service will provide HTML templates to be used in conjunction with a template editor. Any editor with a scripting language would be suitable for use with templates. The templates will provide a first step toward an extensible typing model, that allows for multiple representations of a particular type. Templates will at least allow for some portability and continued utility of links that identify fragments of documents (in Web terminology). Other services will follow.

In order to further study the problems of global scaling, our group has also begun work simulating very large numbers of network components. The model will allow for the addition of new sorts of network services and protocols used for communication, in order to understand better the performance of new algorithms for resolving some of the problems facing us. This work is also in its earliest stages and not yet ready for reporting.

In addition, we continue to participate in standards activities both in the IETF (active participant in the URI Working Group as author of one of the requirements document, participant in others, chair of the Internet Information Infrastructure Architecture Research Group of the IRTF) and the Web Consortium.

8 Conclusion

The relevance of this paper to a networking community is that we must recognize the abstractions and mode of communication that are required by applications and environments for building network-based applications. This is necessary because it is only with such an understanding that we can define both the functional APIs needed between such a layer of abstraction and those protocols on which it sits, such as transport protocols. In addition, without an understanding and some degree of control over how that next layer above transport will behave, we cannot provide the performance support needed by it. It is only when we understand and can manage a layer of communication abstraction that we can allow for it to exchange information with its supporting layers, to allow them to cooperate in providing effective performance.

What we have done in this paper is lay out an extremely simple, powerful object model to be used to provide ubiquitous and long-lived access to a heterogeneous, evolving and mobile set of resources and objects in the face of both the needs of legacy systems and information as well as evolutionary and perhaps revolutionary changes in the domains of applications.

The model we have laid out is object based. Every object has a globally unique, long-lived identifier. We provide location resolution through a set of location translation services, to the addresses needed by the supporting transport protocols. These addresses and/or location translation services are found through a mutable set of hints, allowing for resiliency to failures or performance problems experienced in the net. In addition each object supports one or more roles, defining the abstractions provided by the object. An object may support more than one role and the set of roles it supports may very over time. A role defines a set of the abstract functionality, the abstract structure and the abstract makers for an object playing that role. Through the abstraction of roles we gain separation between definition and implementation, allowing for multiple and perhaps evolving implementations. Since roles themselves are also objects, such a system provides and extremely minimalist model. Linking and composition follow directly from the object model.

In Section 4, we also identified a small, core set of services required to realize such a model. It should be understood that each of these is in fact a role definition for a set of core service instances, and as such can be the superroles of more enhanced services of these sorts.

Thus we find that although all resources in the Information Mesh are objects, we have gained something significant by being able to support a richer infrastructure, as needed by network-based applications, on top of the transport protocols needed for actual communication and exchange of bits.

Acknowledgements

Acknowledgements for this work go to Alan Bawden, David Clark, Timothy Chien, Matthew Condell, Lewis Girod, Jeffrey Van Dyke, and Bienvenido Velez-Rivera. In addition, we wish to acknowledge the support of the Department of Defense Advanced Research Projects Agency under contract number DABT63-92-C-0002.

References

[1] R.J. Anderson, *UEPS - A Second Generation Electronic Wallet*, **Proc. Second European Symposium on Research in Computer Security**, Toulouse, France, 1992, 411-418.

[2] F. Anklesaria, et al., *The Internet Gopher Protocol (a distributed document search and retrieval protocol)*, Network Working Group RFC 1436, Informational, March, 1993.

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

[4] T. Berners-Lee, *Uniform Resource Locators: A Syntax for the Expression of Access Information of Objects on the Network*, Network Working Group RFC 1738, December, 1994.

[5] T. Berners-Lee and D. Connelly, *Hypertext Markup Language - 2.0*, MIT/W3C, Sept. 1995. Also, Network Working Group RFC 1866, November, 1995.

[6] T. Berners-Lee, R. Fielding, H. Frystyk, *Hypertext Transport Protocol – HTTP/1.0.* MITW3C, Oct. 1995.

[7] K.P. Birman, A. Schiper, and P. Stephenson *Light-weight causal and atomic group multicast*, **ACM Transactions on Computer Systems, 9**, 3 (August, 1991), pp 272-314.

[8] A.D. Birrell and B. Nelson, *Implementing Remote Procedure Calls*, **ACM Transactions on Computer Systems, 2**, 1, (February 1984), pp. 39-59.

[9] N. Borenstein and N. Freed, *MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies*, Network Working Group RFC 1521, Sept., 1993.

[10] J.A. Bull, L. Gong, K.R. Sollins, *Towards Security in an Open Systems Federation*, **Second European Symposium on Research in Computer Security**, Toulouse, France, Nov. 1992, 1-20. Also published as Y. Deswarte, G. Eizenberg, J.-J. Quisquater, eds., **Lecture Notes in Computer Science: 648**, Springer-Verlag, Berlin, 1991.

[11] Free Software Foundation, Gnu Info Tree is self-documenting and has the Free Software Foundation copyleft.

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

[13] Hayes et al., *MLP: A system for constructing distributed mixed language programs*, **IEEE Transactions on Software Engineering**, 1988.

[14] International Organization for Standards and International Electrotechnical Committee, *Information Processing - Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*, International Standards 8824, 1987.

[15] B.C. Johnson, *A Distributed Computing Environment Framework: An OSF Perspective*, Open Software Foundation, DEV-DEC-TP6-1, June, 1991.

[16] M.B. Jones, R.F. Rashid, and M.R. Thompson, *Matchmaker: An Interface Specification Language for Distributed Processing*, **Proceedings of the 12th Annual ACM Symposium on Principles of Programming Languages**, pp. 225-235, January, 1984.

[17] E. Jul, H. Levy, N,. Hutchinson, and A. Black, *Fine-Grained Mobility in the Emerald System*, **ACM Transactions on Computer Systems, 6**, 1, (February, ,1988) pp. 109-133.

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

[19] J. Kunze, *Functional Requirements for Internet Resource Locators*,Network Working Group RFC 1736, February, 1995.

[20] B. Liskov et al., **CLU reference manual**, Springer-Verlag, New York, 1981.

[21] B. Liskov et. al, *Communications in the Mercury System*, **Hawaii International Conference on System Sciences**, Kailua-Koni, Hawaii, Jan 5-8, 1988.

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

[23] M. McCahill, et al., *Report of the IAB Workshop on Internet Information Infrastructure, October 12-14, 1994*, Network Working Group RFC 1862, Nov. 1995.

[24] B. Meyer, **Eiffel: the Language**, Prentice Hall, New York, 1992.

[25] Microsoft OLE 2.0 Design Team, *Microsoft OLE 2.0 Design Specification*, **Microsoft Development Library**, April, 1993.

[26] K. Moore, *MIME (Multipurpose Internet Mail Extensions) Part Two: Message Header Extensions for Non-ASCII Text*, Network Working Group RFC 1522, Sept. 1993

[27] T.H. Nelson, **Literary machines**, The Distributors, South Bend, IN, 1988.

[28] B.C. Neuman, *Prospero: A Tool for Organizing Internet Resources*, **Electronic Networking: Research, Applications and Policy**, 2(1):30-37, Spring 1992.

[29] B.C, Neuman, *The Prospero File System: A global file system based on the Virtual System Model*, **Computing Systems**, 5(4):407-432, Fall 1992.

[30] Object Management Group, **CORBA: Architecture and Specification**, The Object Management Group, August, 1995.

[31] R. Orfali, D. Harkey, J., Edwards, **The Essential Distributed Objects Survival Guide**, John Wiley & Sons, New York, 1996.

[32] L.L. Peterson, N.C. Bucholz, and R.D. Schlicting, *Preserving and Using Context Information in Interprocess Communication*, **ACM Transactions on Computer Systems, 7**, 3 (August 1989), pp. 217-246.

[33] J. Postel, *Simple Mail Transfer Protocol*, Network Working Group RFC 821, August, 1982

[34] J. Postel and J. Reynolds, *Telnet Protocol Specification*, Network Working Group RFC 854, May, 1983.

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

[36] D. Raggett, *Hypertext Markup Language Specification Version 3.0*, Internet Draft March, 1995. If this reference is has not been finalized by publication time, it will be removed.

[37] R.T.O. Rees, **The ANSA Computational Model**, Architecture Projects Management, Ltd, AR.001.01, February, 1993.

[38] K.R. Sollins and D.D. Clark, *Distributed Name Management*, **IFIP 6.5 International Working Conference on Message Handling Systems**, Munich, Germany, April, 1887.

[39] , K.R. Sollins, *Cascaded Authentication*, **Proc. IEEE Symposium on Security and Privacy**, Oakland, CA, April, 1988, 156-163.

[40] K.R. Sollins and L. Masinter, *Requirements for Uniform Resource Names*, Network Working Group RFC 1737, February, 1995.

[41] K.R. Sollins and J.R. Van Dyke, *Linking in a Global Information Architecture*, **Fourth International World Wide Web Conference**, Boston, MA, Dec. 1995, 493-508.

[42] Sun Microsystems, *XDR: External Data Representation Standard*, Network Working Group RFC 1014, June, 1987.

[43] Sun Microsystems, *RPC: Remote Procedure Call Protocol Specification Version 2*, Network Working Group RFC 1057, June, 1988.

[44] R.J. van der Linden, **An Overview of ANSA**, Architecture Projects Management, Ltd, AR.000.00, May 1993.

[45] R.J. van der Linden, **The ANSA Naming Model**, Architecture Projects Management, Ltd, AR.003.001, May 1993.

[46] J.R. Van Dyke, **Link Architecture for a Global Information Infrastructure**, MIT LCS-TR 659, June, 1995.

Appendix

A. The Object Role

Inherits from: Null

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

(is-role? *object***)** Optional
Returns true if the given object is a role. 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***)** Optional
Returns a description of the object. The nature of this documentation is out of the scope of this specification.

(get-required-objects *object role***)** Optional
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.

Parts:

whole Required
The part containing the entire object.

documentation Optional
The documentation associated with a given object.

Makers:

(create) Optional
Returns a new empty object. This is optional because a subrole may not inherit this particular maker.

B. Link Roles

Link Role:

Inherits from: object role

Actions:

; Note that when two arguments are *link* and *role*, the action is applied to the link playing that *role*.

(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 text of statement node.

Makers

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

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 named descriptors.

Ordered Link Role:

Inherits from: link role

Actions

(get-ordered-endpoint-range *ordered-link start end*) Required
Returns range of ordered endpoints.

(extract-ordered-endpoint *ordered-link position*) Required
Returns the endpoint at numbered position in ordering.

(set-ordered-endpoint! *ordered-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
Creates a ordered-link. Endpoint list is an ordered list of descriptors.

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 descriptor1 descriptor2*) Required
Creates a binary link.