# Fixing the “Broken-link” Problem: The W3Objects Approach

D. B. Ingham, S. J. Caughey and M. C. Little
Department of Computing Science, University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, United Kingdom
{dave.ingham, s.j.caughey, m.c.little}@ncl.ac.uk

## Abstract

*One of most serious problems plaguing the World Wide Web today is that of broken hypertext links, which are a major annoyance to browsing users and also a cause of tarnished reputation and possible loss of opportunity for information providers. The root of the problem lies in the current Web architecture’s lack of support for referential integrity. This paper presents a model for the provision of referential integrity for Web resources which supports resource migration and tolerates site and communication failures. The approach is object-oriented, highly flexible, completely distributed, and does not require any global administration. An attractive feature of our design is the provision of a lightweight mechanism which provides referential integrity, and which may be customised on a per resource basis to provide increased fault-tolerance and performance. Our system follows an evolutionary approach, supporting parallel operation with the existing Web, allowing users to gain the additional benefits of referential integrity while allowing continued access through trusted software components.*

**Keywords:** distributed systems, World Wide Web, object-oriented, referential integrity, migration transparency

# 1. Introduction

The World Wide Web is a good example of a system where *the whole is greater than the sum of the parts*, i.e., the true power of the Web is achieved through the linking of related resources. The Web is also a true distributed system in the sense that in addition to distributed resources the administrative control over these resources is also distributed.

In a completely decentralised system, like the Web, architectural support is required to maintain consistency between the individual publishing domains. The current Web is lacking in this respect in a number of areas, one of which is that no support is provided for referential integrity. The observable consequences of this deficiency are *broken-links* or *dangling-references*, arguably one of the most serious problems plaguing the Web today. Broken links are a major annoyance to browsing users and also a cause of tarnished reputation and possible loss of opportunity for information providers.

One of the causes of broken links is the deletion of resources that continue to be referenced. Not all resources can be maintained forever; this does not scale since there are various overheads associated with publishing resources, such as system overheads in terms of occupied memory or disk space. Ideally, an information provider would like to maintain resources that are still being used while discarding those that are not. However, this knowledge is not available in the current Web since providers are unaware as to which of their resources are referenced, and so are unable to make sensible decisions as to which resources should continue to be maintained. Without this information, referential integrity cannot be supported and therefore broken links cannot be prevented.

Another cause of broken links is resource migration. Experience has shown that Web resources migrate for many reasons. Intra-server migration typically occurs due to the re-organisation of resources, for example, as a result of an increase in the volume of information published by a given site. Resources move between servers for reasons which include increased popularity necessitating movement to a more powerful machine, or all of a server’s resources may be moved due to decommissioning of their current host machine. Since linking is currently defined in terms of the location-based URL naming scheme, any resource movement causes the URL to change thereby breaking any existing links to the old location.

The HTTP redirect facility only provides a partial solution to this problem since resources containing references to the old location are not automatically updated, and so future requests will continue to access the old location first. More importantly, even with an automatic update mechanism, the lack of referential integrity means that the redirector can never be safely removed since it is impossible to determine when all of the links have been updated.

This paper presents a model for the provision of referential integrity for Web resources in an environment where those resources may migrate, either within a single site or between sites. The approach is object-oriented, completely distributed and does not require any global administration. The proposed scheme, described in detail in the "Referencing Model" section, aims to provide a high degree of flexibility; a default lightweight mechanism provides referential integrity, which may be customised on a per resource basis with additional facilities to provide increased fault-tolerance and performance. Additionally, although the proposed scheme supports the concepts of referential integrity and migration transparency, it acknowledges the requirement for maintaining overall administrator control, illustrated in the "Administration Issues" section.

The referencing techniques are described in terms of the W3Objects model, an overview of which is presented in the next section, "W3Objects Background". We have implemented the model as a system which supports interoperability with current Web resources and software, as described in the "Interworking" section, which allows users to benefit from the advantages of referential integrity and migration transparency without compromising legacy investment. This evolutionary approach is considered essential and therefore our system supports a parallel mode of operation, allowing resources to be referentially managed in the W3Objects world while they continued to be served using the traditional HTTP daemons. Native W3Object resources can be introduced over time as confidence in the system increases.

The "Related Work" section compares our approach to those of other groups and finally we present some concluding remarks and ideas for future work.

# 2. W3Objects Background

The primary objective of the W3Objects research is to develop an extensible web infrastructure which is capable of supporting a wide range of resources and services, not simply the document-based entities of the current web. The model makes extensive use of the concepts of object-orientation to achieve the necessary extensibility characteristics in a flexible manner [Ingham95].

In the model, Web resources are represented as W3Objects, which are *encapsulated* resources possessing internal state and a well defined behavior, rather than the traditional file-based entities. The model supports abstraction since clients interact with W3Objects only through well-defined published interfaces. The objects themselves are responsible for managing their own state transitions and properties, in response to method invocations.

W3Objects may support a number of distinct interfaces, obtained via interface inheritance. Common interfaces may be shared thereby enabling polymorphic access, for example, many W3Objects conform to the HTTP interface which includes methods such as GET. The specific implementation of the methods may differ between the different classes of object. For example, an HTML W3Object may simply return some static HTML state in response to a GET request, whereas a bank account object may return some dynamically generated HTML reflecting the current status of the account. Behavioral inheritance is used to provide desirable properties such as persistence and concurrency control.

Contexts, which are themselves W3Objects, are used as naming domains, within which W3Objects are uniquely named relative to that context. Contexts may be nested and therefore W3Objects may be identified using compound names. Servers can represent contexts providing network access to W3Objects. Addressing is achieved using first-class referencing objects, *W3References*, which contain the location of a particular object in terms of the Internet address of the server and the compound name, correct at the time of the last access.

The W3Object referencing model, described in detail in subsequent sections, aims to provide referential integrity for W3Objects; that is objects continue to persist for as long as W3References to them exist. This problem is complicated by the fact that failures may occur and that W3Objects tend to move around for reasons described in the "Introduction". The following discussion highlights the available techniques for addressing these problems showing how they are incorporated within the proposed model.

# 3. Referencing Model

We would like the holders of references to be guaranteed referential integrity and for unreferenced objects to be garbage collected, both of which require knowledge of the bindings between references and objects. This could be obtained dynamically through the use of distributed tracing techniques but these are notoriously expensive in large-scale systems. Instead we opt for referencing coherence, i.e., we maintain the distributed referencing graph, updating it inexpensively when references are created or deleted and objects migrated. We shall describe our mechanisms for supporting referential integrity here and those for garbage collection in the section “Garbage Collection”. Our arguments can be applied to any referencing mechanism e.g., URLs and Web resources, therefore throughout this section we shall talk of references and objects, rather than specifically W3References and W3Objects.

Various mechanisms may be used to track migrating objects within a distributed system. Imagine a system which is partitioned into S spaces; a space being any domain containing a set of references and objects where inter-space communication is through the use of message passing. An object, O, lives within one of the spaces ($S_{source}$), R spaces have references to O, and N spaces have name servers which refer to O, for simplicity we assume that R and N are disjoint sets. A holder of a reference on space $S_{migrator}$ sends a message to $S_{source}$ instructing it to migrate O to space $S_{destination}$, and as a result O is migrated as shown in the diagram in Figure 1. For the sake of simplicity we shall assume that all communication within the system is unicast Remote Procedure Call (RPC), with at-most-once semantics, and that $S_{migrator}$ is not equal to $S_{source}$ which is not equal to $S_{destination}$. The results of the migration are returned to migrator and so the subsequent invocations from that reference can go directly to the object. However, all other references are now stale.

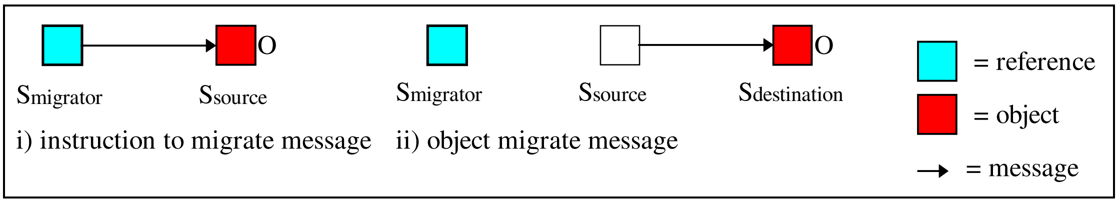


**Figure 1: Object migration**

There are four mechanisms for tracking an object following migration :

1. do nothing at migration time - an object fault occurs at the next invocation and the object must be located through the use of a search.
2. replace the migrated object with a forward reference at migration time - invocation can then be redirected (transparently) through the forward reference.
3. update some known name service (or services) on migration - object faults occur at invocation time, followed by a name service access to obtain the new location of the object.
4. callback to all the object’s references at migration time to inform them of the new location of the object.

Ignoring the 2 RPCs necessary to complete the actual migration as described above, let us now compare the costs for each of these solutions both at migration and invocation time, in terms of inter-space RPCs. We begin our discussion by assuming that we are operating in a fault-free environment, i.e., there are no communication failures or space crashes, and go on to examine the impact of failures later. These messaging costs are summarised in Table 1.

| | **Migration costs (message overhead)** | | **Invocation costs (message overhead)** |
|---|---|---|---|
| | Fault-free | With faults | |
| **Forward Referencing** | 0 | | R - 1 |
| **Search** | 0 | | (1 ... (S - 2)) * (R - 1) |
| **Callback** | R - 1 | [R] | 0 |
| **Name service** | N | [N] | (R - 1) * 2 |

**Table 1: Summary of messaging overheads**

- Forward referencing requires no additional RPCs at migration time but, for the first invocation via each stale reference, an extra RPC is required to follow the indirection to the new destination.
- The search solution also requires no additional RPCs at migration time but for the first invocation via each reference anywhere between 1 and S-2 spaces must be interrogated to discover the new location. (Worse, the search imposes processing and communication costs on spaces which may be completely unconcerned with the object and therefore, while we accept that there will always be a place for search engines within the Web for the initial location of objects, it seems unlikely that searching will be a practical means to relocate large numbers of migrating objects).
- In the case of callback, in order that the holder of every reference be informed of the new location, an additional R-1 RPCs are required at migration time, but no additional RPCs are required when invocations occur.
- For the name service solution, N RPCs are required at migration time, and on invocation a fault occurs, requiring name server access to discover the new location, followed by the actual access of the object.

Examining this table it can be seen that, in the fault-free environment, the search solution is the most expensive in almost all circumstances and that the name server solution is more than twice as expensive as both the forward referencing and callback solutions. These remaining two are equally efficient and differ only in the time at which the costs are incurred.

Now let us examine the situation in which faults may occur. In this situation, in order that referencing coherence be maintained, the callbacks to all references must be carried out atomically, indicated by the square brackets in the table, as must the updates to all name servers. Now the situation changes radically as atomic updates can be expensive, typically requiring a transactional mechanism, and the migration becomes more liable to fail the more callbacks or updates that are required. Callback now becomes much more expensive relative to forward referencing. Name services are also more expensive but, now that faults are possible, they may improve the accessibility of the object as access via each name server provides an alternative path to the object.

Given our assumptions, we can see that no solution out-performs forward referencing (as regards RPC overhead) and, being a totally distributed solution, forward referencing offers distinct advantages in a situation in which faults may occur. For these reasons we have chosen forward referencing as our basic mechanism for tracking object migration. However, we have also shown that both the callback and name server solutions have merits in certain circumstances, i.e., where invocation time costs are to be minimised and where alternative paths to the objects are required for fault-tolerance reasons.

In the next section we will describe our forward referencing mechanism in detail and in a subsequent section describe an extension to our basic mechanism which supports the use of both callback and name server update to provide accessibility and performance improvements.

## 3.1 Forward Referencing

As we have already described, migration introduces an indirection reference into the access path from reference holder to object and further migrations may create a chain of forward references. Similarly, a chain may be extended in the backward direction by the passing of a reference between parties. The diagram shown in Figure 2 illustrates both of these situations.

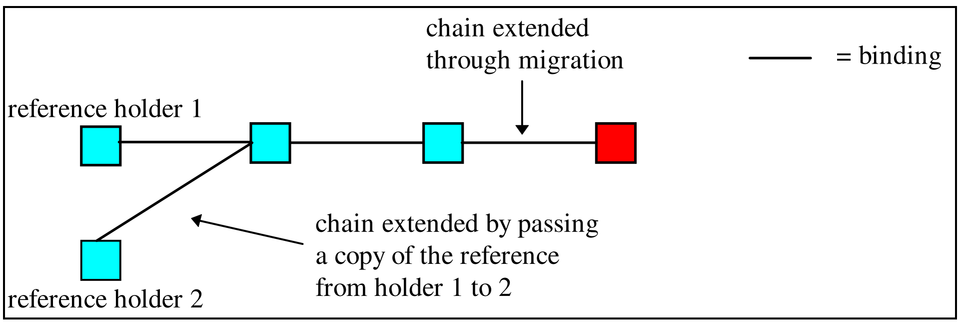


**Figure 2: Extending referencing chains**

Our basic mechanism, which is similar to that proposed in [Shapiro92] and described in greater detail in [Caughey95], requires no additional messages when objects are migrated, and on the first invocation along a chain which contains indirections, every possible point along the chain is short-cut (with certain exceptions to be defined later) and any unnecessary forward references are garbage collected. For example, in the diagram above, if reference holder 2 invokes an operation on the object, then, following the invocation, the situation becomes that shown in Figure 3. Short-cutting is carried out using information piggy-backed onto the normal invocation so no additional messages are required.

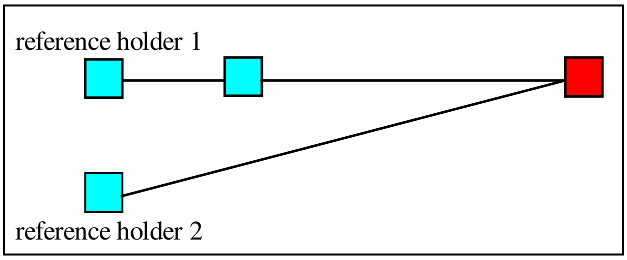


**Figure 3: Situtation after short-cutting**

References are obtained through message exchange as is illustrated in the diagram in Figure 4. Whenever a reference is to be passed to some party, e.g., a user on Space A, the holder of the reference concerned on Space B first creates a new reference which he binds to his existing reference (step 1). As binding occurs within the holder's address space no external messages are required. The new reference is then passed in a message and the chain extended in the backward direction (step 2). However, in order to avoid invocations through the new reference having to follow the indirection, a hint is included within the reference which allows the old reference to be bypassed. Note that it is the chain through the original reference which guarantees referential integrity whilst the hint is a simple optimisation for improved performance. The path from the hint may itself contain indirections (if the target object has migrated) and so *all* invocations, whether along the chain or via the hint, cause short-cutting.

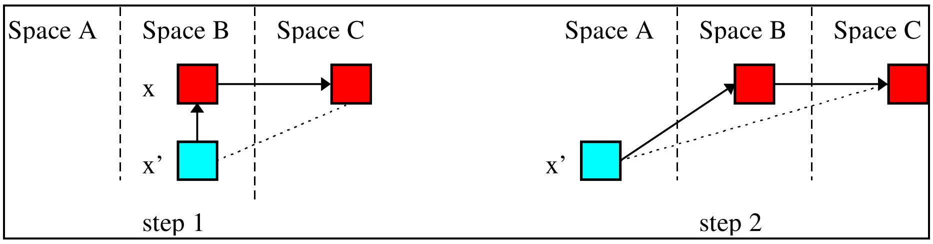


**Figure 4: Obtaining references**

We commonly use the forward referencing mechanism to create context-dependent binding services, that is, forwarding references and remotely-accessible objects (collectively termed *entities*) may be held within context objects so that a user with access to the context may bind to any of the context's entities. This is illustrated in the diagram shown in Figure 5 where a reference holder has bound to a reference held in the context (where it is named 'x'), and therefore indirectly to some object, 'y'. These binding services have many purposes, for example, the entities held within a context may represent the set of objects exported or imported by some domain. *Import contexts* help to minimise network traffic as users may bind to local references which represent remote objects, whilst *export contexts* may provide firewall control over object access.

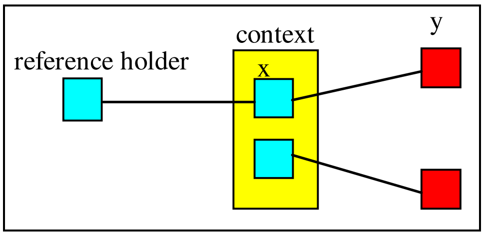


**Figure 5: Illustration of the use of context objects**

Messages generally pass transparently through an indirection reference but in certain circumstances they may be intercepted by either the reference itself, or its containing context. An intercepted message might be transformed into a new format and then forwarded e.g., at a network gateway; partially interpreted e.g., for authentication or encryption purposes; or applied to a local cache of the object. Short-cutting occurs as a (transparent) default in our system but in circumstances where we require reference holders to *always* utilise an indirection (in order to intercept messages, e.g., for security purposes) short-cutting may be inhibited. An example requiring the suppression of short-cutting is described in the section "Support for Browser Hotlists".

However, there are two potential problems. Firstly, if a chain contains indirections then i) additional points of failure are introduced, and ii) performance at invocation time is affected. Although our short-cutting mechanism efficiently removes indirections at invocation time, there may be reference holders with such strict requirements for accessibility or performance, that any, even short-lived, indirection is a problem. Secondly, forward referencing relies on an unbroken chain from reference to object. Our implementation is fault-tolerant in that it copes with space crash and network partition but it does rely on eventual recovery. If an object is migrated from some space and the space is then removed from service, the chain may be permanently broken and the object inaccessible (without recourse to search). Fortunately there are solutions to these problems in the use of name services and callback, which we shall describe in the next section.

## 3.2 Name Services and Callback

Name services have two functions. Firstly, they may be used to maintain the binding between some name and the object, i.e., by tracking object location, and secondly they may be used to provide alternative paths to the object. We support referential integrity through the use of our forward referencing mechanism, which means that whilst an object is referenced a path to the object is guaranteed to exist. Reference holders who require greater availability for an object may register it in multiple name servers in order to obtain multiple paths to the object. However, as illustrated in the diagram shown in Figure 6, if all these name servers are up to date, i.e., there are no indirections, and the object is migrated (by a party not shown in the diagram), then all the name servers now share a common path. Each name server will be updated on the first access of the object via their reference but until this happens they continue to share the common path (and, if some of the name servers are being used *only* to provide alternative paths, the update may not occur *until* the primary path fails - precisely the occasion at which a common path is most undesirable).

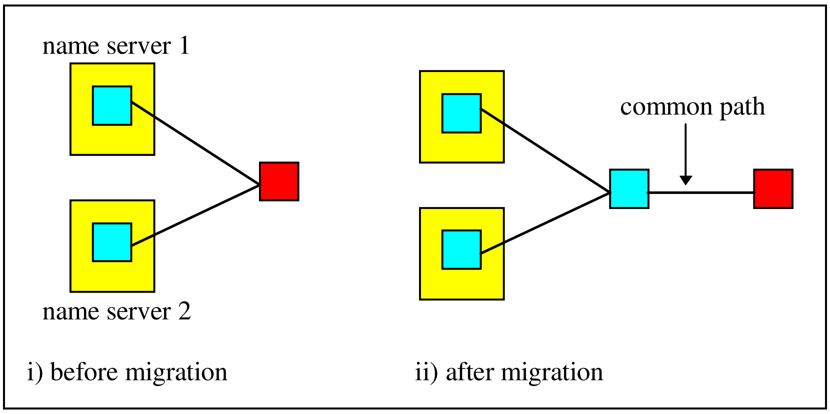


**Figure 6: Migrating name-server referenced objects**

For some reference holders the existence of a common path may be undesirable, or even unacceptable, and they may require the name servers upon which they depend to be automatically updated following object migration. Our mechanism allows individual reference holders to register their object with some name server, and inform the object of this action. Then, if the object migrates, it informs all the appropriate name servers of the new location. Should the primary path to an object fail a reference automatically tries its alternative paths (if any exist), with any redirection being transparent to the reference holder. This technique may be used to maintain bindings to objects even where an intermediate space on the chain has been removed from service permanently. If the object performs the necessary updates during the migration then a burden is imposed on the migrator (in terms of delay) and by all of its reference holders (in that they cannot access the object whilst it is migrating). However, if the updates are delayed until some time after the migrate then a common path will exist until the updates occur. Clearly there is a trade-off here and so our mechanism's default is for updates which occur after the migration (at some time convenient to the object) but allows reference holders the option of explicitly requesting that an update be performed during migration and, if so, whether it is to occur atomically.

There may be reference holders for whom no indirection is acceptable at invocation time, and who therefore require callback in order to be informed of the object's new location. We model the callback process as being identical to the name server process described above in that registering a callback may be seen as the holder of the reference registering *themselves* as a name server for the object.

The diagram shown in Figure 7 illustrates our mechanism for both name server update and callback. In i), the holder of reference 1 has registered an alternative path using the name server shown, and the holder of reference 3 has registered a callback and explicitly requested that it occur as a part of the migration. (We term any reference through which a holder has registered an alternative path or a callback, *a compound reference*). So, during the migration (initiated by a party not shown in the diagram) the object informs reference 3 through a callback that it has migrated, and at some time following the migration, the object informs the name server. After the updates both the name server and the reference which registered the callback hold the new location of the object, as shown in ii). Note that each reference holder may register any number of name servers, and/or a callback on that reference independent of all other reference holders for the object.

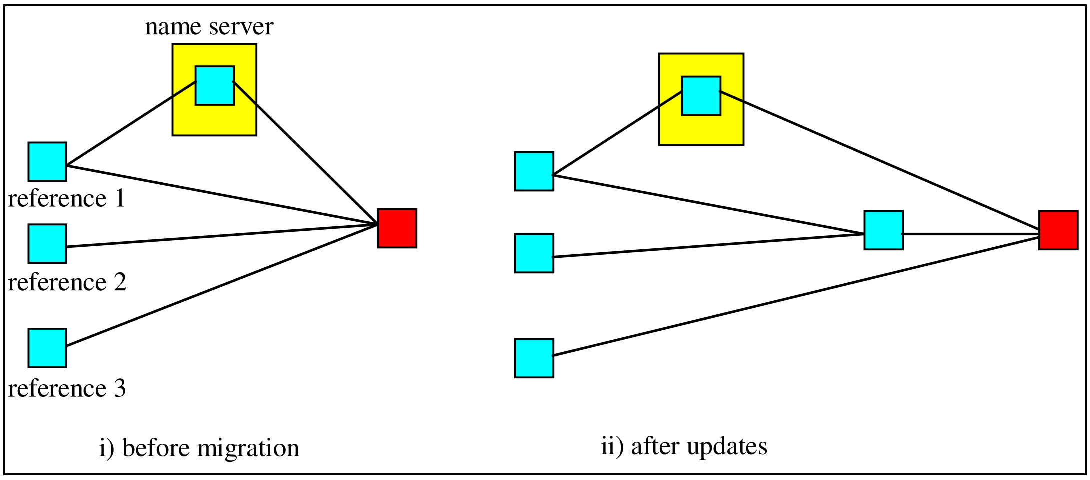


**Figure 7: Name-server update and callback mechanisms**

## 3.3 Conclusion

We use the forward referencing mechanism described above to ensure referential integrity within the W3Object domain. Holding a reference guarantees that the target object will persist, despite space and communication failures and will remain accessible (at least, eventually) via the reference. As we show in [Caughey95], our mechanism is totally distributed and completely scaleable. The basic mechanism caters for object movement, requiring no additional messages at migration time, and the minimum message overhead on invocation. The mechanism is flexible is that a simple extension allows individual reference holders to obtain a variety of additional guarantees concerning the availability and performance of specific objects through the use of name servers and callbacks.

# 4. Garbage Collection

Garbage collection mechanisms identify and delete objects which are no longer required. In a distributed environment these garbage collection mechanisms can be driven both by actions taken by holders of references, i.e., whenever an unreference request is made, and by actions taken by the objects themselves, e.g., periodically determining the liveness of reference holders. In a failure free environment garbage collection is relatively straightforward. However, in the presence of machine crashes and network partitions, fault-tolerant garbage collection is required to ensure that unwanted objects are eventually removed. We shall briefly describe the garbage collection mechanisms within the W3Object model, and the interested reader is referred to [Caughey95] for a more complete description.

The default garbage collection mechanism for W3Objects operates as a result of unreference requests from users and is fault-tolerant to node crashes and network partitions: the system maintains referencing information on behalf of users on stable storage and guarantees that unreference requests are eventually delivered to the object. However, it makes various assumptions about the types of failures which can occur, e.g., there are no catastrophic failures, i.e., loss of stable storage is not supported and it is assumed that crashed nodes eventually recover. Inevitably the assumptions will break down occasionally, causing *lost references*, thereby creating garbage which cannot be detected by our basic mechanism. Worse still, administrators being aware that undetectable garbage *can* occur, will be tempted to manually remove objects they suspect are garbage, and in doing so will inevitably remove referenced objects.

To reduce the probability of certain objects becoming undetectable garbage, our default mechanism can be extended by the addition of garbage collection driven by the objects themselves. In other distributed systems [Parrington95][Rosenburg92][ANSA91][OMG93] this is typically referred to as *orphan detection* [Panzieri88], although we term it *object-driven garbage detection*. If lost references are detected (e.g., through the use of periodic keep-alive messages) then the objects automatically reduce their reference count for the appropriate references.

Our system provides reference-driven garbage collection for all objects and allows object implementors to specify an appropriate object-driven detection mechanism if required.

# 5. Administration Issues

In the following description we shall talk about a site administrator removing objects, but this could be any user with sufficient permission. It is beyond the scope of this paper to consider how such capabilities are assigned to individual users; we simply assume that a suitable mechanism is available.

The previous sections presented an approach to object referencing which offers automatic garbage collection. Objects which are reference counted continue to exist for as long as references to them exist. Garbage collection removes any objects as soon as they are no longer required. However, for a number of reasons administrators will require the ability to directly control the lifetime of objects regardless of the number of references to them:

- *overhead*: all objects must reside in some physical media, such as main memory or disk. Despite the continued reduction in prices of these media, an object can impose an overhead on space and performance which may eventually require it to be removed.
- *administration*: the original publisher of an object may be unable to continue to maintain it, e.g., because of financial constraints, or the publishing site may change their policy on what types of objects they wish to export, or be forced to remove certain objects for legal reasons.
- *fault-tolerance*: catastrophic failures in the distributed environment may prevent unreference requests from being delivered to the object, or may result in references being permanently lost and therefore no unreference can be performed.

For these reasons and others, the site administrator may wish to override the default mechanisms presented earlier and remove referenced objects. However, rather than simply allow these objects to be deleted, which would result in the broken-link problem described earlier, we provide support for a staged removal strategy in which deletion of an object is the final option.

Rather than immediately delete a reference counted object, the administrator has the option of replacing it with another (lightweight) object which indicates to the caller what has happened to the original object. We refer to this replacement object as a *gravestone*. Because the gravestone replaces the object, it obviously assumes the object's identity. Therefore no modifications to other objects and users need be made to reflect that a removal/replacement has occurred. This is important because it allows the site administrator to make unilateral decisions about the objects he controls without affecting those objects he cannot control.

A gravestone is a W3Object and as such has many of the properties of the object it replaces, including its current reference count. A gravestone represents a significant reduction in the overhead imposed on the system since it takes up less space and requires less system processing. In addition, gravestones can share state, further reducing the system overhead. When an unreference request is received for the original object it is applied to the gravestone, and if its reference count drops to zero it is garbage collected.

Gravestones also benefit from being W3Objects in that they can perform arbitrary work whenever they are accessed. For example, an administrator with a need to reduce the overhead on a site may initially be satisfied with archiving objects which are infrequently accessed rather than deleting them. After a suitable period in which no requests for the archived objects are made, further archiving may result, e.g., moving from tape to CD-ROM. Throughout this period the objects may be replaced by suitable gravestones. The gravestone represents the archived object and may unarchive it, replacing the gravestone, either automatically on access, or by offering the user an option to do so. Objects may eventually be deleted and replaced by another gravestone.

# 6. Interworking With Current Web Software

The Web, like most software systems, acknowledges the need to provide support for legacy applications and resources. The native HTML resources effectively provide the gluing agent for the integration of other foreign resource types managed by a variety of information systems, e.g., FTP, Gopher, Usenet etc. W3Object research aims to further this interoperability by allowing more complex objects and services to be accessed using Web technology, therefore providing a common interface to the various resources and services connected to the Internet [Ingham95].

The previous discussion of the proposed referencing model has focused on the provision of referential integrity and object migration transparency in terms of W3Objects. The remainder of this section addresses a number of interoperability issues that require special attention to facilitate the evolutionary introduction of this referencing model into the Web.

## 6.1 URL-based Referencing of W3Objects

In the “Referencing Model” section, it was shown how object references are used to define the hypertext linking between the resources managed by W3Object servers. The use of these *smart* references enables the provision of the desirable properties of referential integrity and migration transparency. However, in order to allow HTML resources managed using traditional server technology to reference W3Objects and to allow access using existing browsers, it is also necessary to provide support for URL-based addressing.

As previously mentioned, W3Objects are named uniquely within contexts that are managed using servers. It is therefore possible to describe the location of a W3Object using a URL, consisting of the Internet address of the server and the compound name of the object within that particular server. However, it should be noted that such URL references to W3Objects only provide the same guarantees as URL references to existing Web resources, since they do not contribute to the referencing information of the object to which they refer. Therefore holding a URL to a W3Object provides no guarantee as to the continued existence of that object. Similarly, if an object migrates, then access via the old URL will only succeed, passed transparently to the real object via the forward reference, until such time as all W3References to the old location have been updated by short-cutting.

One way of improving this situation with regard to migration transparency is through the use of the compound reference technique described earlier. If an object is permanently referenced via a name server, then a URL based upon this name server reference will provide continued access to the object even after migration. This approach is similar in nature to and provides the same level of migration transparency support as the proposed Persistent URL (PURL) scheme [PURL].

For other objects, a variation of this technique can be used whereby an object reference is temporarily registered with a name server for the period of time that it continues to be referenced by at least one W3Object. The name server entry is automatically removed when the object ceases to be referenced. A URL on this temporary name server reference will provide no referential integrity guarantees but will allow continued access to the object, even in the event of migration, for its lifetime.

## 6.2 Maintaining HTTP Access

Current browser software tends to be monolithic in nature, that is, support is provided for a fixed set of application-layer protocols, such as HTTP, FTP etc. Although this situation is likely to change with the wide-spread acceptance of Java technology, which would allow browsers to download code to access W3Object servers directly, currently in order to interoperate with today’s popular browsers then any new server software must provide support for one of these common protocols. In our prototype implementation [Ingham95], HTTP to W3Object gateways are used to translate HTTP requests into appropriate W3Object operations. A client request is received by the gateway which attempts to bind to the W3Object identified by the URL contained in the request. After a successful bind the HTTP operation, e.g., GET, is mapped to the appropriate W3Object operation which is then invoked on the object, the results being passed to the client.

## 6.3 HTML Representation of W3References

Interworking with existing Web software requires support for HTML resources. Dedicated HTML W3Objects consist primarily of static HTML code and may be viewed as encapsulated versions of current HTML files. Additionally, other classes of W3Object are able to dynamically render some aspect of their state in HTML format. The hypertext linking between W3Objects is managed using embedded W3References, which may contain multiple paths to the object through the use of compound references. A specially designed W3Object-aware browser could make use of these alternative paths, either by resorting to a secondary path if the primary path failed or by choosing between the different paths based on some heuristic analysis.

Providing support for existing browser software requires a translation from the internal W3Object HTML representation, with links based on W3References, to the plain HTML form using URL-encoded links. In response to a client request the object generates a plain HTML representation on-the-fly. W3References are translated to URLs at this time, however, in the case of compound W3References this is not a one-to-one translation. There are a range of possible approaches for encoding the multiple paths, such as linking the anchor to an intermediate page which shows the multiple paths, or providing multiple anchors.

Note that this list of alternative URLs provides multiple paths to the object, as opposed to a URN-style approach [Hoffman95] which requires an external resolution service.

The object-oriented nature of the system allows these presentation options to be specified on a per object basis. It should be noted that the URL-based links within these dynamically generated may become invalid during the period of time that the HTML page is cached within the browser.

## 6.4 Support for Browser Hotlists

Hotlists or *favorites* are an essential component of today's Web browsers, allowing users to easily maintain lists of resources that they wish to return to at a later date. Since browsers maintain hotlist entries as URLs, typically in an HTML file, a periodic traversal of a user's hotlist will commonly reveal a number of broken links. There is nothing to prevent URL references to W3Objects from being held within conventional hotlists, bearing in mind that no additional integrity guarantees will be provided.

In order to improve this situation we have developed the concept of a *hotlist server*, which manages hotlists on behalf of an individual or a group of users. A hotlist server runs as a daemon process managing W3Object name server contexts which hold references relating to user's hotlist entries. There are several advantages to this server-based approach:

- as a daemon process, it runs continuously in the background allowing it to take part in reference update activities, such as responding to call-back, when there are no browsers running.
- when coupled with programs known as *update agents*, this technique allows any browser to benefit from the advantages of W3Object referencing.
- a server may manage hotlists on behalf of many users, bringing new benefits for group working as described later.

Hotlist servers manage W3Objects and W3References like any other W3Object server. The server accepts user requests to register objects within the user's name server, and to provide a list of such entries when required. W3Object hotlist entries behave like any other W3Reference, providing the benefits of referential integrity and object location transparency.

A question arises as to how browsers interact with the hotlist server. While a dedicated W3Objects browser may provide direct support for the hotlist server, alternative mechanisms are required to allow current Web browser software to interoperate with the hotlist servers. One technique is the use of update agents programs which are configured to understand the hotlist file format of a particular browser and perform periodic background scans to determine whether any entries have been added or deleted.

Workers in a group-based environment frequently wish to share information about newly discovered resources. Since hotlist servers are designed to manage hotlists on behalf of several users, they provide an opportunity for improving this sharing of information. Assuming certain security mechanisms are available, the discussion of which is beyond the scope of this paper, the hotlist server could provide support for group-based hotlists or user hotlist sharing. The concept of import contexts, described in the “Referencing Model” section, can be used here to minimise the message traffic.

## 6.5 Evolution Support

We consider an evolutionary approach essential. Therefore, a completely parallel mode of operation is supported whereby existing HTML resources continue to be served through the HTTP server of choice while W3Object representations of these resources are used to achieve the additional referencing guarantees. This is an advantage of our approach over related systems, for example Hyper-G, which only provides referential integrity support for native resources [Kappe95b].

This is achieved by providing an alternate implementation of the HTML W3Object, which instead of maintaining the HTML state internally, contains a reference to the HTML file in the file-system. A management tool is provided to support creation of these proxy objects and import them into the W3Objects world. The tool also allows registration of links between these objects in the form of W3References. The tool provides support for link creation by analysing the HTML resources and extracting the existing link information.

Using this technique, information providers can take advantage of the benefits of referential integrity without chance of disrupting service to clients. The next stage of evolution would be the creation of *true* W3Objects which may be accessed through gateway technology, as described above. And finally, W3Object servers can replace HTTPD for all objects.

# 7. Related Work

There are a number of research groups investigating issues related to the work presented in this paper. The difficulties in arriving at a consensus of opinion within the IETF as regards the introduction of URNs (see later) has lead several groups to consider intermediate solutions. The Persistent URL (PURL) scheme from the OCLC is one such approach which introduces a level of indirection into URL addressing [PURL]. A PURL identifies a logical location of a resource which is then mapped to the real location using the HTTP redirect directive. WebLinker [Aimar95] is a similar scheme which uses Local Resource Names (LRNs) as intermediaries to URLs. All resources at a particular site are described in terms of their LRN which is mapped to the URL upon access.

The remainder of this section describes two more ambitious schemes, illustrating their relationship to the W3Objects referencing model.

## 7.1 Hyper-G

The Hyper-G system from University of Technology, Graz, Austria is a fully functional rival to the Web. Hyper-G provides a number of attractive features not present in the Web, including user-authenticated access, linking from arbitrary resource types, and efficient searching [Kappe95b]. Resources within a Hyper-G server are categorised into *core* resources, which are only referenced locally and *surface* resources, which either link to, or are linked from remote resources. Database technology is used to hold resource meta-information and the relationships between resources including linking information, which is bi-directional. The database maintains referential integrity between local resources by removing all links to a resource when it is removed. This is an alternative viewpoint to that adopted in this paper which aims to maintain resources while outstanding references exist.

Maintaining referential integrity for surface resources requires a server update protocol to propagate update information describing the deletion or migration of resources and creation or deletion of links. This information is added to an *update list*. A flooding algorithm (p-flood) is used to disseminate this update information to all other servers. Therefore, a server’s update list contains both locally generated messages and those received from other servers. Exact details of the operation of the flooding algorithm may be found in

[Kappe95a] but essentially the servers are organised in a circle and periodically each server sends its update list to a small number of other servers resulting in the eventual transmission of all updates to all servers.

Analysis of this update scheme reveals a number of drawbacks:

**Scaleability:** The remote server update mechanism claims to scale well. In fact, since it is a broadcast mechanism, in the sense that all servers are informed of all updates, the total additional message traffic required scales proportionally to the number of servers. The forward referencing approach scales proportionally to the number of outward links held by an object, a much smaller figure than the number of servers.

**Cost Distribution**: The cost associated with the maintenance of referential integrity is distributed among all of the servers. The vast majority of update requests that a particular server will process will be for resources that are of no local interest. The W3Objects forward referencing approach divides the cost between reference holders only. Therefore a cost is only incurred by those parties who are explicitly interested in the object.

**Consistency**: The proposed update mechanisms will not eliminate broken links entirely since update information is disseminated lazily resulting in a period of time when links will be invalid. Conversely, forward referencing provides strong consistency; a path to the object will always exist.

In conclusion, while the Hyper-G flood-based approach for maintaining distributed referential integrity may be appropriate for enterprise-wide systems, it is questionable whether the approach is appropriate for an Internet-wide system where the number of servers is likely to be measured in millions.

## 7.2 IETF Integrated Internet Information Architecture (IIIA)

The Internet Engineering Task Force (IETF) has been concerned with the issue of location-transparency for several years now and has developed a strategy known as the Integrated Internet Information Service [Weider94a], (known elsewhere as “architecture” rather than “service” hence IIIA). The proposed architecture incorporates URLs, as a means of identifying a single copy of a resource, but also introduces an abstract, globally-unique, location-independent naming scheme, Uniform Resource Names (URNs) and Uniform Resource Characteristics (URCs) which are used to encode meta-data information about a particular resource. Although the implementation of this scheme is still under negotiation, the basic operation relies on each resource being allocated a URN. This is mapped, through appropriate resolution services, to a URC, which provides information about the resource such as author, keywords etc., as well as URLs identifying the location of various copies of the resource. Using this scheme, resources will describe links in terms of URNs, and appropriate resolution services will provide the mapping to a URL for the resource allowing it to be accessed.

Although the IIIA approach shares some common goals with our work, in the sense that both are concerned with provision of resource migration transparency support, in other areas the aims are orthogonal. URNs are concerned with resource identity and among their identified functional requirements are global scope, global uniqueness, and persistence [Sollins94]. The W3Objects referencing mechanisms could be used in conjunction with the IIIA scheme with W3References acting like *smart URLs*. An object identified via a URN could be mapped to a URC which identifies the W3Reference(s) for the object.

Within the IIIA, when an object moves some mechanism is required to update the URN resolution service with the new location. The concept of a “Resource Transponder” [Weider94b] is introduced. Again, while the exact functionality is still under discussion, the basic operation involves associating a transponder with each resource that is responsible for updating the resolution service when the resource moves. Note that while not specifically mentioned within the IETF documentation, in order to ensure that URLs remain valid this update must occur atomically with the resource migration.

This approach is similar to the W3Objects technique for name server update. In fact, the W3Objects model could support update of URN resolution services simply by providing an alternative implementation of the name server update methods. Using W3References would provide the additional benefit that the URN service update may be performed lazily since the forward referencing mechanism would provide continued access to the resource.

In summary, the W3Objects referencing model could integrate well with the proposed IIIA with W3References providing additional benefits over URLs as the resource location mechanism.

# 8. Conclusions and Future Work

The W3Objects referencing model is an object-oriented, flexible and fault-tolerant approach for maintaining referential integrity for Web resources, thereby providing a solution to the problem of "broken-links". Forward referencing provides a lightweight mechanism for the provision of referential integrity, which can be supplemented by call-back and name server techniques to achieve higher performance or degrees of fault-tolerance; the appropriate combination being selected on a per resource basis. A key feature of our design is the support for an evolutionary approach, allowing our services to be used in parallel with existing trusted Web software components.

Prototype implementations of most of the techniques described within this paper have been constructed, including the referencing mechanisms, maintenance tools and update agents described in the "Interworking" section. We now plan to refine our implementation with a view to employing it to achieve referential integrity among the resources held within the collection of Web servers within the Department.

# 9. Acknowledgments

The work reported here has been partially funded by grants from the Engineering and Physical Sciences Research Council (EPSRC) and the UK Ministry of Defense (Grant Numbers GR/H81078 and GR/K34863) and GEC-Plessey Telecommunications.

# 10. References

[Aimar95] A. Aimar et al., "WebLinker, A Tool for Managing WWW cross-references," Computer Networks and ISDN Systems, Vol. 28 No. 1&2; Selected Papers from the Second World Wide Web Conference, December 1995

[ANSA91] "The ANSA Computational Model," ANSA Architecture Report AR.001.00, August 1991.

[Caughey95] S. J. Caughey and S. K. Shrivastava, "Architectural Support for Mobile Objects in Large Scale Distributed Systems," In The Proceedings of the 4th IEEE International Workshop on Object Orientation in Operating Systems (IWOOOS), Lund, Sweden, August 1995.
URL: http://arjuna.ncl.ac.uk:80/arjuna/papers/shadows-iwooos95.ps

[Hoffman95] P. E. Hoffman and R. Daniel, Jr., "*URN Resolution Overview*," Internet Draft, April 1995. Work in progress.
URL: http://www.ics.uci.edu/pub/ietf/uri/draft-ietf-uri-urn-res-descript-00.txt

[Ingham95] D. B. Ingham, M. C. Little, S. J. Caughey, S. K. Shrivastava, "W3Objects: Bringing Object-Oriented Technology To The Web," The Web Journal, 1(1), pp. 89-105, Proceedings of the 4th International World Wide Web Conference, Boston, USA, December 1995.
URL: http://www.w3.org/pub/Conferences/WWW4/Papers2/141/

[Kappe95a] F. Kappe, "A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems," J.UCS Vol. 1, No. 2, pp. 84-104, Springer, February. 1995.

[Kappe95b] F. Kappe, K. Andrews, and H. Maurer, "The Hyper-G Network Information System," J.UCS Vol. 1, No. 4 (Special issue: Proc. of the Workshop on Distributed Multimedia Systems, held in Graz, Austria, Nov. 1994), pp. 206-220, Springer, April 1995.

[OMG93] Object Management Group Inc, "The Common Object Request Broker: Architecture and Specification, Revision 1.2", OMG Document Number 93.12.43, December 1993.
URL: ftp://ftp.omg.org/pub/docs/93-12-43.ps

[Panzieri88] F. Panzieri and S. K. Shrivastava, "Rajdoot: A Remote Procedure Call Mechanism Supporting Orphan Detection and Killing," IEEE Transactions on Software Engineering, Vol. 14, No. 1, pp. 30-37, January 1988.

[Parrington95] G. D. Parrington et al, "The Design and Implementation of Arjuna," USENIX Computing Systems Journal, Vol. 8, No. 3, 1995.
URL: http://arjuna.ncl.ac.uk/arjuna/arjunadesign-usenix.ps

[PURL] The PURL Homepage.
URL: http://purl.oclc.org/

[Rosenburg92] W. Rosenberg, D. Kenny, and G. Fisher, "Understanding DCE," O'Reilly and Associates, Inc., 1992.

[Shapiro92] Shapiro, M., Dickman, P. and Plainfosse, D., "Robust, Distributed References and Acyclic Garbage Collection," Symposium on Principles of Distributed Computing, Vancouver, August 1992.
URL: ftp://ftp.inria.fr/INRIA/Projects/SOR/RDRAGC:podc92.ps.gz

[Sollins94] K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names (RFC1737)," December 1994.
URL: ftp://ds.internic.net/rfc/rfc1737.txt

[Weider94a] C. Weider and P. Deutsch, "A Vision of an Integrated Information Service (RFC1727)," December 1994.
URL: ftp://ds.internic.net/rfc/rfc1727.txt

[Weider94b] C. Weider, "Resource Transponders (RFC1728)," December 1994.
URL: ftp://ds.internic.net/rfc/rfc1728.txt

```
ERROR: undefined
OFFENDING COMMAND: A

STACK:

(i)
0
324
```