A Community of Agents Maintaining Link Integrity in the World-Wide Web (Preliminary Report)

Luc Moreau and Nicholas Gray
(L.Moreau,ncg97r)@ecs.soton.ac.uk
Multimedia Research Group
Department of Electronics and Computer Science
University of Southampton
SO17 1BJ Southampton UK

Abstract

In this paper we present an agent architecture to maintain link integrity in the World-Wide Web. It consists of a community of agents collaborating to provide services to users, authors and administrators. In summary, agents maintain link integrity when documents are updated, moved, or when WWW sites are reorganised. They also provide authors and administrators with usage information such as documents that are currently bookmarked or documents that are no longer accessed. The architecture is also able to advertise new versions of documents. At a low-level, we use a distributed garbage collection algorithm to trace links and maintain their integrity.

1 Introduction

The World-Wide Web is ubiquitous and is now an essential component of the electronic desktop as it facilitates the access and publication of documents. Unfortunately, WWW users will soon become disillusioned should they continually follow links to documents that no longer exist. A hypertext system is said to have the property of *link integrity* [10] if links always point to documents, i.e. the system does not have any dangling links.

The hypertext community has defined other models of hypertexts, such as Hyper-G [20] or Xanadu [29], which maintain link integrity. Even though these approaches may be very attractive, they would require us to replace the WWW, which seems quite difficult due to its widespread use.

In this paper, we present an agent-based technique to maintain link integrity in the World-Wide Web. The proposed architecture is distributed and uses the collaboration of user agents, author agents, and administrator agents to offer a spectrum of services that are so far unavailable in the WWW. The functionality of our agents may be summarised as follows. *(i)* Our agency provides users with link integrity, i.e. the guarantee to be able to follow links (in particular when bookmarked). *(ii)* It allows authors to publish and update documents, or even to reorganise the structure of documents, while

still preserving link integrity. Furthermore, our notion of agency is able to inform authors about the current usage of documents, such as accessibility or non-accessibility, which in turn may be used to decide to advertise new document versions. *(iii)* Administrators are also given the opportunity of reorganising WWW servers without breaking link integrity; they have access to usage information, which may help them to reorganise their servers.

A notion of *publication contract* [28] is central to our architecture. An author and his/her administrator agree to publish documents and to maintain their availability for a given contractual duration[1]. Similarly, users that adopt the approach are entitled to the aforementioned services within the contractual conditions. Users, authors and administrators are free to adopt or not adopt the architecture, which preserves compatibility with the existing WWW.

From a technical viewpoint, at a low level, the architecture uses garbage collection techniques [22, 26, 31, 32, 36] to maintain link integrity. Garbage collection is a technique used in some programming languages to *automatically* trace objects that are accessible and to detect those that are garbage. The garbage collection approach is very useful as it is automatic, and therefore offers autonomy, and it provides information about current document usage that can be used by a higher-level agent layer.

In this paper, we describe the agent architecture for link integrity (Section 2). Then, in Section 3, we present three agents for users, authors and administrators, respectively. In Section 4, we review our current prototype implementation, which can be used in a corporate intranet. Finally, we discuss the architecture and compare it with related work in Section 5.

2 The Architecture

The World-Wide Web (WWW) architecture [1] is based on a client-server model, as displayed in Figure 1. Entities communicate using the Hypertext Transfer Protocol (HTTP) [15]. Users have access to browsers, i.e. clients, which send http requests to http servers via a communication medium that we generically call the Internet. This communication medium contains proxies, caches, routers, gateways, etc. A request reaches an http server, which typically retrieves a document and sends it back to the browser on the same connection. Documents are referred to by Uniform Resource Locators (URLs) [2]

In addition, a hypertext markup language (HTML) is understood by browsers [2]. HTML documents may contain *links* to other documents; a link consists of a URL to a document. Browsers display links as buttons that users may follow by simple clicks. The process of following links is also called *navigation*. Users may also bookmark documents, i.e. save their URLs for a later use.

In the initial specification of the WWW, servers are defined as *stateless*[2]. As a result, they do not maintain any information about the documents that are currently accessed by users, and they do not keep track of documents that are bookmarked by users. In such a context, it is impossible to maintain any form of link integrity, or even offer any form of link-related service.

[1] Nelson, in the Xanalogical Structure [28] introduces the notion of commitment so that people can rely on continued availability.

[2] The HTTP protocol is evolving and can include a state [21], but that mechanism alone does not maintain link integrity.

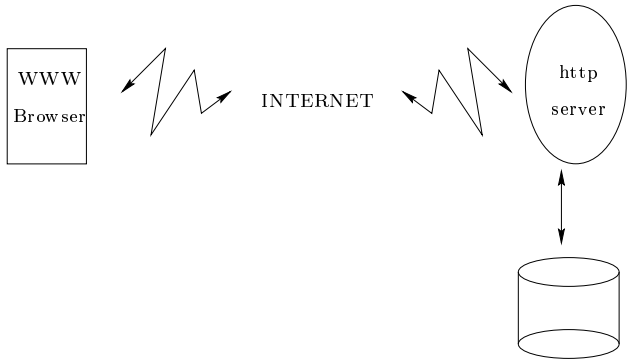

Figure 1: The WWW Architecture

This phenomenon is further amplified by the nature of publishing itself. Every user is allowed to publish documents simply by storing them in a special directory. The very presence of a document in this directory makes it accessible via a URL specifying its access path; symmetrically, removing the document makes the URL a dangling link.

Our new architecture, displayed in Figure 2, extends the current WWW architecture. There are two new essential components called the *client agent* and the *server agent*. They play an active role in the WWW architecture: the client agent is configured as a WWW proxy of the user's browser so that it can observe, intercept, transform, or redirect any user's request. The server agent acts as a new http server so that it can also observe, intercept, or transform requests, and then can pass them to a regular http server. Both the server agent and the http server have access to a database of documents. Furthermore, the client agent and server agent provide agent functionality, which we describe in Section 3, for users, authors, and server administrators. The user interface of those agents takes the form of WWW browsers.

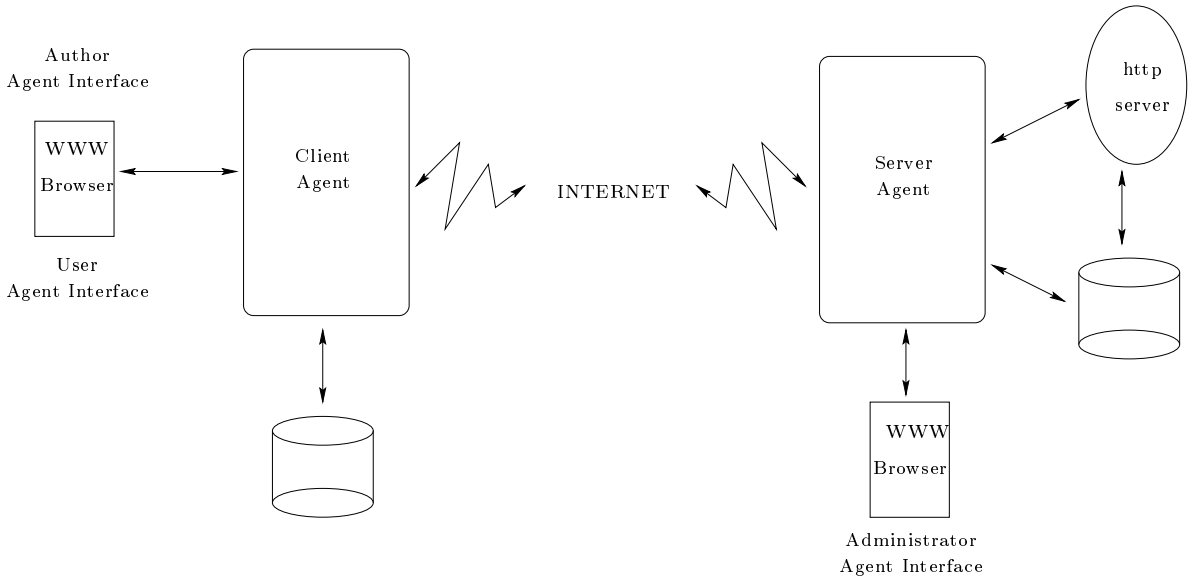

Figure 2: The Architecture

As in the traditional WWW architecture, http requests and retrieved documents are transported by the Internet. In addition, control messages are exchanged between client agents and server agents. They can be issued by client agents when users bookmark, delete or mail documents, or by server agents when a document is published, updated, or moved. Client agents have to be persistent across sessions, and therefore they also access a local store. In the following section, we cover the agent functionality.

3 Agents for Users, Authors and Administrators

The architecture described in Section 2 is the foundation of three agents respectively aimed at *users* navigating the WWW, *authors* publishing documents, and *administrators* managing servers. The functionality of these agents is described in the following sections. Let us note that the services that are made available to users, authors, or administrators are the result of a collaboration between client agents and server agents over the network, as we shall describe in Section 4. At the centre of our agent architecture is a notion of contract that we first define.

3.1 The Publication Contract

In principle, in a system that maintains link integrity, once a document is bookmarked by a user it should remain accessible as long as the bookmark is alive. Not only would such a publishing mechanism be *impractical* but also it would *not* be *flexible* enough. It would be impractical because it would require keeping an amazingly vast amount of documents consuming a lot of resources. It would not be flexible because policies change, institutions disappear, and individuals move. Furthermore, some documents have an intrinsic limited lifetime, e.g. adverts, weather forecast; authors may not be interested in publishing some documents longer than a given period of time.

Therefore, a publishing system should be based on a notion of *contract*, which specifies the conditions under which a document is made available. We have identified the following non-exhaustive list of contract types [28, 29].

1. Publish without contract.
2. Publish forever.
3. Publish for an unknown period, but notifies removal or update.
4. Publish for a specific period of time.
5. Publish until some conditions on usage is satisfied.

The current WWW does not use any form of contract, and consequently offers a minimal service. At the other end of the spectrum, a "forever contract" would guarantee that a document remains available forever; such a kind of contract would be useful for book archives, museum sites, etc. In between these two types of contract, we can imagine many variations. The third option provides more flexibility as it allows long publication duration, but also permits authors to update or reorganise documents; users are guaranteed to receive an update or removal notification. The fourth type of contract is used by authors who wish to make a document available for a fixed period of time. Finally,

our architecture provides some very dynamic information about document usage. This information can be used to determine the end of the publication period; for instance, it is possible to determine when a document is no longer accessed or even no longer reachable by a given set of users.

3.2 User Agent

The user interacts with his/her agent via the WWW browser. The user agent has three goals:

1. Maintaining documents accessibility, i.e. link integrity;
2. Managing bookmarks;
3. Advertising new document versions.

The initial motivation of this paper, that is, maintaining *link integrity*, explicitly appears as the first goal of the user agent. Once a user has downloaded a document d, the user agent makes sure that every document directly or indirectly accessible by following links from d will be available if the user decides to do so.

Current WWW browsers have no real management of bookmarks: they allow users to store or delete bookmarks, and simply maintain a file of URLs that persist across sessions. However, there is no point in maintaining bookmarks if they become dangling (for instance because referenced documents were removed). Therefore, real bookmark management also implies link maintenance.

Documents are usually not static entities; they also have a life that takes the form of a series of revisions. When a user has bookmarked a document, it is the agent's role to keep track of new versions of the document. Information about document revisions can be communicated to the user by different ways. *(i)* On user's demand, the agent must be able to show the different revisions that were published since the user bookmarked the document. *(ii)* In some cases, the user might like to be explicitly informed of the availability of a new document version: the agent has then to contact the user (via email or popping up a window on the screen). *(iii)* In other cases, users are only interested in the latest version of a document. When a user asks for such a document, the agent automatically retrieves the latest version on behalf of the user.

As explained in Section 3.1, some documents can be published for a fixed period of time. It is then the agent's role to ensure that the user is kept informed of the document expiry date. The agent may even pro-actively download a document that is about to be expired, for instance, in the user's absence.

Users frequently exchange documents or URLs by electronic mail. Their agents can again play an active role by ensuring that a URL sent to a recipient is maintained consistent, at least for a given period of time.

3.3 Author Agent

Publishing documents in the current WWW is very easy as authors simply have to store them in a dedicated directory. We must ensure that our new architecture does not add extra burden on authors. Our solution is to keep this approach but the author has to

inform his/her agent that a new revision is available and must be exported to the WWW[3]. However, publishing documents involves more than making them available in a directory. The goals of the author's agent are summarised as follows:

1. Publishing documents and maintaining the existence of previously published documents according to the adopted contract.

2. Informing authors of the current usage of published documents.

Authors have to choose a publication contract and then delegate the task of publishing documents to their agent. In general, publishing a new version of a document does not mean that previous versions should be destroyed, unless pre-specified by a contract. So author agents have to keep previous document versions.

The author agent informs the author of the current usage of documents. For instance, it is able to inform authors about users[4] that are currently accessing a document or that have bookmarked documents.

In some cases, authors want to be told when a document is no longer accessed or bookmarked. More generally, authors can do some processing when such a situation arises: for instance, to re-advertise the existence of a document, to delete the document, or to make it available from another access point. As soon as the author agent detects such an event it can initiate a computation, as specified by the author.

3.4 Administrator Agent

The administrator agent provides web masters with two new services:

- Informing administrators about server usage.

- Facilitating reorganisation while maintaining link integrity.

The administrator agent is a generalisation of the log file created by current http servers. Not only does it maintain the access history of the server, but it also provides more dynamic information, such as users that are currently accessing a document, or users that have bookmarked documents retrieved from this server.

The agent provides the administrator with the possibility of reorganising WWW sites, and the architectures ensures that link integrity is preserved. In order to maintain consistency our distributed garbage collection algorithm uses forwarding pointers, which indicate the position of new documents. However, in the long term, URLs bookmarked by users should be updated. Changes may be actively propagated to users, or they can be lazily propagated as users (or their agent) access documents.

[3]The reader might think that it should be the agent's role to decide to export the new revision. In principle, this could be done by regularly polling the file system or by periodically exporting new revisions. However, in practice, we felt that that the author was the only one to know when a document was ready to be published, and it was then natural to inform the agent. Similarly, it is only the author who can approve the use of the new version.

[4]The reader might wonder about user's privacy in our architecture. The user's identity is not revealed, but their agent and its location may be communicated. We consider that this issue is part of the contract between publishers and users: a service can only be provided if the user agrees to reveal his/her agent identity. In an agent architecture, we could also consider a third-party agent acting on behalf of a user agent and preserving its anonymity.

The information about the usage of sites can help administrators to change the geographic organisation of their sites. This is particularly useful for companies which have mirror sites on different continents: mirrors can now be designed on document usage.

3.5 Agent Collaboration

The collaboration of agents can even offer more services. The current architecture is able to determine which users have currently bookmarked a document. Such information can be used by agents to help users to meet other users that access the same documents. This facility is similar to the Mbone architecture, which advertises the users which are currently using the multicast channel [4]. This facility also bears some resemblance with the project Memoir [30], which helps to foster user's collaboration, by analysing their access trails.

4 Implementation

In this Section, we describe the prototype that we have implemented using Java [16] and NeXeme [27], a distributed implementation of Scheme. It is based on a distributed garbage collection algorithm described in another report [26]. In the current implementation we consider one http server only, but the algorithm is designed to support multiple servers. We now concentrate on the issue of maintaining link integrity of bookmarked and currently accessed documents.

Figure 3 summarises the information flow between a browser and a http server. The client agent is set as a http proxy for the WWW browser; all requests issued by the WWW browser are intercepted and analysed by the client agent (Arrow 1). In the simplest case, the request is sent to the server specified by the URL (Arrow 2). The server agent acts as a http server, and receives requests. It passes requests to a regular http server (Arrow 3), after transforming their URLs. The http server interprets the transformed URL as a call to a CGI script (Arrow 4), which retrieves a document from the document repository, in our case managed by the revision control system CVS [5]. The document is returned to the server agent (Arrow 6), which sends it back on the connection to the client agent (Arrow 7), which in turn passes it to the WWW browser (Arrow 8).

When a server agent delivers a document to a client agent, it also provides the client agent with information about the document; the message `notify-hold` gives the client a brief description of the received document, including a unique document identifier and a version number amongst others (Arrow 9)[5].

Both the client and server agents maintain a table of visited documents. Documents sent by the server are entered in the server agent table; similarly, documents passed to the user are entered in the client agent table. A document remains in the server table, as long as there is a client agent that contains it in its table.

When a document d is downloaded by a WWW browser, d becomes the current focus, and users are allowed to follow links and download other documents. However, the document d stays in memory for a while, and it remains accessible via the history mechanism,

[5] In terms of implementation, this document description could be encoded in the document itself as meta information; however, it is more convenient to pass it as a separate request, for the prototype development.

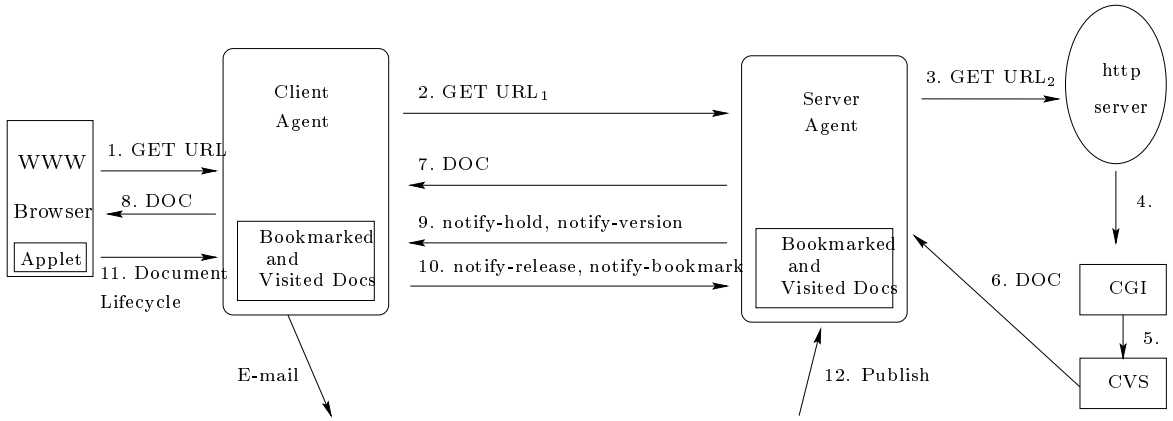

Figure 3: Implementation

until the browser decides to reclaim its space. The user agent keeps the document d (in fact its document identifier) in its table of visited documents for the same length of time, at least. Once the document is released by the WWW browser, it can be removed from the client agent table. Then, using the message `notify-release` (Arrow 10), the client agent informs the server agent who served the document d, that d was released by the WWW browser. It is the client agent's role to detect when browsers release documents. A correct implementation, though not very efficient, is to wait until the browser terminates its session; then, all visited documents are released at once. Another solution is to associate a Java applet [16] with each document sent to the browser; the applet lifecycle is related to the lifecycle of documents, and the applet can inform the client agent when documents are released (Arrow 11), using the `destroy` and `finalize` methods.

As far as bookmarks are concerned, we use a similar mechanism as for visited documents: a bookmarked document is stored in a table by the client agent, which informs the server agent using a message `bookmark-notify` (Arrow 10). Having no access to the internals of browsers, we have defined our own bookmark mechanism: the applet attached to a document also provides a button, which sends a message to the client agent when actioned.

The server agent maintains tables[6] of documents being visited or bookmarked. In addition, it maintains a reference to each user agent that is visiting or has bookmarked one of these documents. When the server agent is informed that a new version of a document is published (Arrow 12), it propagates this information to the agents that had bookmarked the document, using the message `notify-version` (Arrow 9). In addition to bookmark tables, client agents maintain information about new versions of bookmarked documents. In Figure 4, we can see the information shown by the agent when the user displays his/her bookmarks. Here, two documents are bookmarked. For the first one, the user agent shows that two other new versions are currently available. For each document, the user can register his/her interest in the latest version of a document (as it is for the first bookmark). When the user retrieves a URL (Arrow 1), the agent can automatically transform it into URL_1, which is the latest version (Arrow 2). When the user agent receives information about new document versions, it also has the ability to advertise

[6] Caches that maintain similar tables can reduce the server load.

Figure 4: Bookmarks

them to the user.

The author and administrator agent functionality relies on tables of bookmarked and visited documents. At every moment, they indicate what the accessed and bookmarked documents are. It is then very easy to define a function that has to be activated when a document is no longer referenced or bookmarked.

When a user emails a document to another user, his/her user agent should make sure that the sent document is considered as "accessed" by the server agent. Figure 5 displays the exchange of messages when an email is sent. First, each client agent maintains a table of sent documents; when a document is sent by an agent, it is also stored in the table of sent document. When the receiver's agent receives the document, it then registers to the server, which adds the document and the agent to its table of visited documents. In turn, the server informs the emitter's agent that the document was successfully sent; the latter can then remove it from its table of sent documents. This mechanism is described in detail in [26].

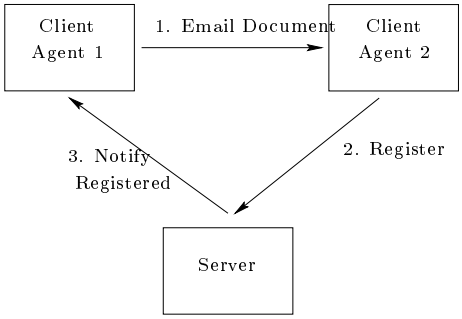

Figure 5: Emailing Documents

An author can publish documents by activating the author agent that will take care of making their public WWW directory available. The current content of the public WWW directory is tagged by a unique version id and stored in the current document repository. Let us consider two documents d_1 and d_2 that were published under version v; let l be a link between d_1 and d_2. As the link l implicitly refers to version v of these documents, it must be instantiated to the version v. We have the choice to instantiate links when documents are stored or when they are retrieved from the repository. We chose the latter options as it allows us to share documents across versions.

5 Related Work

Wooldridge and Jennings [37] distinguish *weak agents* from *strong agents*. The agent architecture described in this paper is of the former category. Weak agents enjoy the following properties: *autonomy*, *social ability*, *reactivity*, and *pro-activeness*. Our agents are autonomous as they can operate without the direct intervention of humans; they observe user's behaviour, and collect or exchange related information. They have a social ability as they interact with users (via browsers), but also with other agents: first, the services provided by the architecture result from agent collaboration; second, we plan to make those services available to other agents. Agents enjoy the property of reactivity

because they constantly react to their environment (http requests, document publication, or server reorganisation). Finally, they exhibit a form of pro-activity as user agents may take the initiative to act on user's behalf (for instance to retrieve documents that become expired).

The agent architecture presented in this paper is part of a more general research activity on Distributed Information Management. In the Multimedia Research Group, at the University of Southampton, we are using similar notions of weak agency in various projects. In the project Memoir [30], agents are used to perform data-mining and resource-discovery tasks. These agents collect trails recording the documents that are being viewed, the time spent to view documents, and whether documents are printed or bookmarked. Using these trails agent use matching algorithms to find other users accessing the documents or other relevant documents. DeRoure *et al.* [14] describe how agents can be used to help navigation, resource discovery, or integration with legacy systems. Dale [7] present a mobile agent architecture to solve a similar problem.

We can also regard our architecture as belonging to the class of "personal agents" such as Maes' interface agents [25], Davies' Jasper [8], Voorhees' InfoScout [35], Lieberman's Letizia [24], and so on. The services that we offer are however different because they are based on the information used to maintain link integrity. The so-called "push technology" [34] pushes requested information to users desktop computer. In our architecture, we do not propagate new document versions, but we inform user agents that new versions are available. This service also comes from the availability of tables of sent documents, which themselves originate from a publishing contract.

The technique that we have described to maintain links is based on distributed garbage collection algorithms. Tables of visited or bookmarked documents is a variant of distributed reference counting called *directory listing* [3, 31]. The rerooting technique used when mailing documents or when moving documents is based on a recently developed algorithm [26]. Reference counting garbage collection is not able to collect distributed cycles. Such cycles occur frequently in the World-Wide Web, as it is quite common to have a set of documents mutually referring to each other. However, in our case, documents only reside on WWW servers or temporarily in WWW browsers. We can easily use timestamping [23] or marking [22] to detect unused distributed cycles. In the latter case, groups of servers can decide to cooperate to detect unused cycles between them; this approach is promising as it involves a negotiation phase, which perfectly fits into the agent framework. Let us mention that the ability to start some processing when a document is unused or unreferenced is called *finalization* [18] in the garbage collection community. Search engines are interesting components to be modelled in our approach; they maintain tables of documents collected by robots, launched periodically from a set of roots. If not designed properly, such engines would keep references to documents, which would appear to be used all the time; such a phenomenon also appears in programming languages, and the solution is to use a notion of *weak pointer* [36].

The initial Dexter hypertext model [17] enforces link integrity, but does not define any way of implementing it. Davis [10, 12] surveys the issue of link integrity in hypertext systems. The techniques discussed by Davis are not automatic and usually rely on post-mortem repairs. Our approach relies on garbage collection technique to ensure that links are never dangling. He also studies *the editing problem* where the update of a document might make a link inconsistent because the document no longer contains the information that it contained when the link was created. Such a phenomenon could occur in our

architecture when the user agent decides to retrieve the latest version of a document. Techniques that analyse the contents of documents should be used to inform authors that some links may be semantically incorrect, even though they are not dangling; our architecture provides the mechanism to inform authors.

An interesting extension of our approach is to deal with separate link bases as in Microcosm [11] or the distributed link service [13]. Both systems introduce "generic links" which computes the source anchor of a link at navigation time. Our agent architecture computes the destination anchor of a document at retrieval time: both the user agent and the server agent use the navigation context to determine the version of documents pointed by links.

Ingham, Caughey and Little [19] present a solution to solve the "broken link" problem. They mention distributed garbage collection technique but no implementation detail is provided. They use an algorithm [32] to short-cut forwarding pointers, but they do not describe the garbage collection algorithm. They call bind and unbind methods to update reference counters, but without precaution, this implementation suffers from causality problems. Our approach is based on a simple algorithm [26] and is able to deal with email and document moves. The WWW community also provides mechanism to publish URLs permanently, e.g. Uniform Resource Names [33], similar to ISBN for books. These approaches are complementary to our architecture as they guarantee that the root of a document will always be available via a URL, but they do not provide any solution for maintaining inner-link integrity.

Creech [6] discusses author-oriented link management. In particular, he provides a mechanism by which links are automatically updated, whenever possible; if automatic update is not possible, he delivers the appropriate information to make reasonable decision on how to update these links. We believe that his robot-based mechanism to collect information is not as sound as our garbage collection technique: new updates may invalidate the information even before robots return it. However, his techniques that analyse the contents of documents and provides authors with information on how to update links can be integrated in our approach.

Hyper-G [20] maintains link consistency by propagating document changes using a probabilistic broadcast algorithm. Our approach does not rely on broadcast to propagate changes. Document updates are propagated in two different cases: *(i)* If an author publishes a new version of a document, it is advertised to users who have bookmarked it. *(ii)* When a document is moved. In the former case, advertising may be delayed as much as we want, as it is not a mandatory computation; in particular, advertising can be lazily propagated as users read documents. Changes resulting from document moves may also be propagated lazily as described in [26].

6 Conclusion

We have presented an agent architecture to maintain links in the World-Wide Web. At a low level, it uses garbage collection techniques to trace links and to maintain their integrity. At a higher-level, it provides three agents aimed at users, authors and administrators, respectively.

We envisage to pursue our work in two directions. The current architecture is suitable for a corporate intranet. However, if we wish to support a world wide network such as the Internet, techniques to reduce server load have to be investigated. In particular, we

need to introduce levels of caches, with tables of accessed or bookmarked documents, which can reduce the traffic and the size of tables to maintain. Even though the agent functionality that we provide is new and unavailable in the current WWW, we believe that more content-based processing would be useful to users and authors: both need to be informed of content changes in documents.

7 Acknowledgements

This research was supported in part by the Engineering and Physical Sciences Research Council, grant GR/K73060 and by the European Union's ESPRIT programme, MEMOIR project, grant 22153. The authors wish to thank John Dale, Hugh Davis, David DeRoure, and Sigi Reich for their useful comments.

References

[1] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World-Wide Web. *Communications of the ACM*, 37(8):76–82, 1994.

[2] Tim Berners-Lee and D. Connolly. Hypertext Markup Language Specification 2.0. Technical Report RFC 1866, MIT/LCS, November 1995.

[3] Andrew Birrell, David Evers, Greg Nelson, Susan Owicki, and Edward Wobber. Distributed Garbage Collection for Network Objects. Technical Report 116, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301, December 1993.

[4] Syngen Brown and Mick Kahn. Janet mbone service. Technical report, The JNT association, 1996. Available from `http://www.ja.net/documents/mbone.html`.

[5] Per Cederqvist. *Version Management with CVS*. Signum Support, November 1993.

[6] Michael L. Creech. Author-oriented link management. In *Fifth International World Wide Web Conference*, pages 6–10, Paris, France, May 1996.

[7] Jonathan Dale. *A Mobile Agent Architecture for Distributed Information Management*. PhD thesis, University of Southampton, January 1998.

[8] John Davies, Richard Weeks, and Mike Revett. JASPER: Communicating Information Agents for WWW. In *Proceedings of the Fourth International World-Wide Web*, pages 473–482, Boston, USA, 1995. Also available at http://www.w3.org/pub/Conferences/WWW4/Papers/180/.

[9] N. J. Davies and R. Weeks. Jasper: Communicating inforamtion agents. In *Proceedings of the Fourth Word Wide Web Conference*, pages 473–482, Boston, USA, December 1995.

[10] Hugh Davis. *Data Integrity Problems in an Open Hypermedia Link Service*. PhD thesis, University of Southampton, 1995.

[11] Hugh Davis, Wendy Hall, Ian Heath, Gary Hill, and Rob Wilkins. Towards an Integrated Environment with Open Hypermedia System. In *Proceeding of the Second European Conference of Hypertext (ECHT'92)*, pages 181–190, 1992.

[12] Hugh C. Davis. Referential integrity in open hypermedia link service systems. Technical report, University of Southampton, 1997.

[13] David DeRoure, Les Carr, Wendy Hall, and Gary Hill. A Distributed Hypermedia Link Service. In *Third International Workshop on Services in Distributed and Networked Environments (SDNE'96)*, pages 156–161, Macao, June 1996.

[14] David DeRoure, Wendy Hall, Hugh Davis, and Jonathan Dale. Agents for distribyted multimedia information management. In *Pratical Application of Intelligent Agents and Multi-Agent Systems*, pages 91–102, London, UK, April 1996.

[15] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. Rfc2068, World Wide Web Consortium, January 1997. Available from `http://www.w3.org/Protocols/Specs.html`.

[16] J. Gosling, Guy Lewis Steele, Jr, and B. Joy. *The Java Language Specification*. Addison-Wesley, 1996.

[17] Frank Halasz and Mayer Schwartz. The Dexter Hypertext. *Communications of the ACM*, 37(2):31–39, February 1994.

[18] Barry Hayes. Finalization in the Collector Interface. In *Proc. 1992 International Workshop on Memory Management*, pages 277–298, Saint-Malo (France), September 1992. Springer-Verlag.

[19] D.B. Ingham, S.J. Caughey, and M.C. Little. Fixing the Broken-Link Problem: The W3Objects Approach. In *Proceedings of the Fifth International World Wide Web Conference*, volume 28 of *Computing Networks & ISDN System*, pages 1255–1268, Paris, France, May 1996.

[20] Frank Kappe. A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems. *J. of Universal Computer Science*, 1(2):84–104, 1995.

[21] D. Kristol and L. Montulli. Http state management mechanism. Technical Report rfc2109, World Wide Web Consortium, February 1997. Available from `http://www.w3.org/Protocols/Specs.html`.

[22] Bernard Lang, Christian Queinnec, and José Piquer. Garbage Collecting the World. In *Proceedings of the Nineteenth Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, pages 39–50, Albuquerque, New Mexico, 1992.

[23] F. Le Fessant, Ian Piumarta, and Marc Shapiro. A Detection Algorithm for Distributed Cycles of Garbage. In *OOPSLA'97 Garbage Collection and Memory Management Workshop*. `http://www.dcs.gla.ac.uk/~huw/oopsla97/gc/papers.html`, 1997.

[24] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[25] Pattie Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):31–40, July 1994.

[26] Luc Moreau. A Distributed Garbage Collector with Diffusion Tree Reorganisation and Object Mobility. Technical Report M97/2, University of Southampton, October 1997.

[27] Luc Moreau, David DeRoure, and Ian Foster. NeXeme: a Distributed Scheme Based on Nexus. In *Third International Europar Conference (EURO-PAR'97)*, volume 1300 of *Lecture Notes in Computer Science*, pages 581–590, Passau, Germany, August 1997. Springer-Verlag.

[28] Theodor Nelson. Xanalogical structure: its paradigms and its renascence. Available from the author, 1997.

[29] Theodor Holm Nelson. *Literary Machines*. Project Xanadu, 1987.

[30] Aggelos Pikrakis, Tilemahos Bitsikas, Stelios Sfakianakis, Mike Hatzopoulos, Dave DeRoure, Wendy Hall, Sigi Reich, Gary Hill, and Mark Stairmand. Memoir — software agents for finding similar users by trails. In *The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents (PAAM'98)*, London, UK, March 1998.

[31] David Plainfossé and Marc Shapiro. A Survey of Distributed Garbage Collection Techniques. In Henry G. Baker, editor, *International Workshop on Memory Management (IWMM95)*, number 986 in Lecture Notes in Computer Science, pages 211–249, Kinross, Scotland, 1995.

[32] Marc Shapiro, Peter Dickman, and David Plainfossé. SSP Chains: Robust, Distributed References Supporting Acyclic Garbage Collection. Rapport de Recherche 1799, INRIA-Rocquencourt, November 1992.

[33] K. Sollins and L. Masinter. Functional requirements for uniform resource names. Technical Report rfc1737, World Wide Web Consortium, December 1994. Available from `http://www.ics.uci.edu/pub/ietf/uri/`.

[34] Wes Thomas, Angus Davis, and Paul Dreyfus. NETCASTER: Push Technology Everywhere. Technical report, Nestcape, inc., 1997. Available from `www.netscape.com`.

[35] Ellen M. Voorhees. Agent Collaboration as a Resource Discovery Technique. In *CIKM Workshop on Intelligent Information Agents*, Gaithersburg, Maryland, December 1994. Also available from `http://www.cs.umbc.edu/ cikm/1994/iia/papers/`.

[36] Paul R. Wilson. Uniprocessor Gargage Collection Techniques. In *Internatinal Workshop on Memory Management*, Lecture Notes in Computer Science, Saint-Malo, France, September 1992.

[37] M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2), June 1995.