UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering and Applied Science
Department of Electronics and Computer Science

A progress report submitted for continuation towards a PhD

Supervisor: Professor Wendy Hall & Dr Leslie Carr
Examiner: Hugh Glaser

Preserving Linked Data Integrity on the Semantic Web by application of techniques from Hypermedia

by Robert A. Vesse

June 24, 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING AND APPLIED SCIENCE
DEPARTMENT OF ELECTRONICS AND COMPUTER SCIENCE

A progress report submitted for continuation towards a PhD

by Robert A. Vesse

This report presents a Literature Review of past work in hypertext link integrity and current work in the emerging area of Semantic Web link integrity. A design and prototype for a system which applies some ideas from hypertext link integrity to the Semantic Web is presented alongside plans for future enhancements of this system. In addition other possible avenues of research regarding ideas from traditional hypertext link integrity are briefly discussed.

Contents

Chapter 1

Introduction

Hypertext is a technology which evolved from an idea first proposed by Vannevar Bush that 'the human mind does not work by alphabetical or numerical linking but through association of thoughts' Bush (1945). In his article he presented an idea for a device called the 'memex' which would allow a person to browse large collections of information and link items together and add annotations to items. From this concept the idea of hypertext was born with its aim being to provide a mechanism to link together collections of accumulated knowledge in an interesting and useful way in order to improve access to them and express the relationships between information.

Thus there is an obvious problem in hypertext with regards to what happens when links do not work as intended. Links are unfortunately susceptible to becoming 'broken' in a number of different ways and this has become an open research question, particularly since the 1990s and the advent of large scale hypertext systems like the World Wide Web (Berners-Lee et al., 1992). This problem is known as Link Integrity and can be divided into two main problems *a*) the 'dangling-link' problem and *b*) the editing problem .

The Semantic Web is an extension to the existing Web which is designed to be about data rather than documents. It is concerned with expressing large numbers of simple facts in an interlinked way such that software can reason across data on the web. On the Semantic Web objects and concepts are referred to using URIs which allows the data to be linked together in meaningful ways and allows for following links to discover more about something. This also means that the Semantic Web is potentially highly susceptible to link integrity issues since to get real value from it you need to be able to traverse the web of linked data reliably. The Semantic Web also introduces two additional issues in link integrity which are *a*) URI identity & meaning and *b*) co-reference.

1.1 The 'Dangling-Link' Problem

The most commonly addressed issue in link integrity is the 'dangling-link' problem. This is when traversing a link results in the user's browser presenting a 404 Error (or more generally a 4xx Error) to them indicating the requested resource cannot be found, the link points to nowhere so it is considered to 'dangle'. Links can 'dangle' for several reasons but usually it is because the owner of the site where the target resource was located has renamed/moved/deleted the resource in question. Typically the owner will have no idea that they have caused links to 'dangle' by changing their site since it can be difficult to find out which sites link to your site. Unless the other sites are under your control you cannot physically change the site in question even if you knew their links were now wrong. In other cases a link may appear to 'dangle' either due to network problems or because a link has been made to a resource which the author had access to but is actually subject to access restrictions which the author was unaware of. Additional causes of 'dangling' links can include a change of server technologies meaning that file extensions in the URLs have changed or the change of domain/subdomain for a site. This is the type of problem which most research into Link Integrity attempts to solve since it is simpler to address than the editing problem.

1.2 The Editing Problem

The other (arguably) more minor issue is the editing problem (or content reference problem) which occurs when a resource is modified in such a way that although a link to the resource itself will work the resource itself has changed in such a way that the link is incorrect. This may either be that the link pointed to an embedded anchor in the resource which no longer exists or that the content of the resource is no longer relevant to the context from which the link came. For example consider the scenario in which you linked to the products page or a company that produced a product you were reviewing. Six months later the company may produce a completely new product and decides to remove the old product from their products page, your link will continue to work but it no longer references the correct content. This issue has been the focus of less research since it is a much harder problem to solve as it requires machines to understand both the semantics of links and the resources being linked. In terms of the Semantic Web this is an issue since if you link your data to some concept in someone else's data there is nothing to stop them changing the meaning of that concept and therefore indirectly change the meaning of your data.

1.3 URI Identity & Meaning

The Semantic Web introduces an issue of URI identity & meaning since URIs are no longer referring simply to documents or other HTTP accessible resources but to potentially anything. As a result there has been considerable debate in the Semantic Web community about what the identity & meaning of a URI is. Does a URI identify only one thing or can it identify many things and what are the practical repercussions of this? Does a URI have a fixed meaning or is its meaning contextual, and perhaps more importantly to what extent does the meaning of a URI matter to a Semantic Web application? In the event that dereferencing a URI fails - it's a 'dangling-link' - then it would appear that we have missing meaning. Without a clearer idea of whether missing meaning matters it is hard to say what if anything we should be doing to prevent this situation arising. This issue is somewhat beyond the scope of my current work but Halpin (2009) provides a good overview of the issues.

1.4 The Coreference Problem

Co-reference is an issue in link integrity specific to the Semantic Web though it originates in established issues from the fields of natural language processing (Bagga, 1998) and Databases. The basic issue is that on the Semantic Web everything is referred to using URIs and often you will end up with multiple URIs for the same thing since various different organisations will be creating URIs for their data in their own formats. This is somewhat inevitable since many organisations (especially businesses) want to control their data as much as possible even if they do publish it semantically. Unfortunately this makes it difficult for Semantic Web applications to find all the data that relates to a particular thing since that thing may have many URIs and there is not necessarily any source of information which will tell you this. Research into the co-reference problem looks at ways in which co-referent URIs can be determined and how this information can be conveyed to Semantic Web applications.

1.5 Report Overview

In this report a Literature Review of work from hypermedia research into link integrity and recent work on Semantic Web link integrity is given in Appendix A. The prototype software developed as the focus of my research so far is detailed in Section 2 and proposals for extending this work and other research avenues in this area are discussed in Section 3.

Chapter 2

Work Report

2.1 All About That

The concept for All About That (AAT) came about from a conversation about how you might take the concept of JIT resolution (see section A.2.2) and apply it in a Semantic Web application. The original idea was that as an application was resolving URIs in order to build up a graph about some concept and resolving some URI failed the application could use some kind of service to find alternate sources of data about that URI. It was proposed that a system like CRS could be leveraged to provide this lookup service. While this idea was interesting in itself it was proposed that it would be more useful to think about applying link integrity to an actual task a user might wish to perform on the Semantic Web; to this end the idea of AAT as a URI Profiling tool was conceived.

The Tool is designed such that an end user enters a number of URIs that they are interesting in knowing about and the tool builds a local profile of the RDF at those URIs. The application periodically resolves the URIs that are being monitored and from this creates a version history of the RDF and a record of changes in the RDF. This allows end users to view past versions of the URIs profile and for them to see change reports on the RDF.

In order to achieve this a C#.Net Library called dotNetRDF (see Appendix B) was developed to provide a lightweight API for RDF. It was decided to implement my own library rather than use an existing one primarily because there was only one viable open source candidate - SemWeb [1] - in my chosen language. I chose not to use SemWeb since it is written in the previous version of C#.Net which means it cannot leverage some of the newer features of the language which make some code much simpler. For example my own library makes heavy use of LINQ (Language Integrated Natural Query) to provide

[1] http://razor.occams.info/code/semweb/

programmatic selection and query over RDF graphs in a very efficient way. AAT makes extensive use of this library in order to implement its core features.

2.1.1 Terminology

In this section I use the following terms/phrases to refer to concepts within All About That:

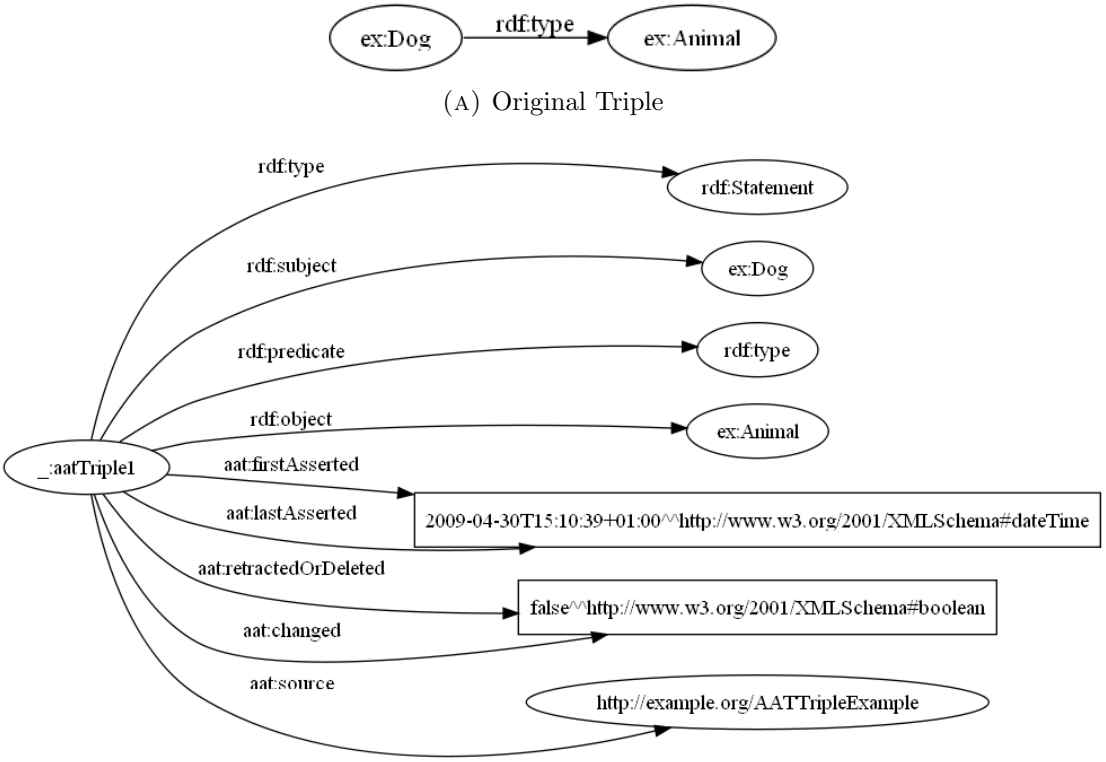
- Profile - The (Local) Profile of a URI is the set of triples in the annotated AAT format which make up AATs knowledge about a particular URI.
- Update a Profile - An Update of a Profile is when AAT dereferences the profiled URI to retrieve the current RDF at that URI and then updates the profile accordingly.
- Update Service - The service that runs in the background and periodically updates all profiles.
- Export - An Export is when the annotated triples are transformed back into the original triples.

2.1.2 Triple Annotation

All About That stores triples locally in a transformed annotated form from which it can recreate the original triples when necessary; it does not store the original RDF itself. The basic idea behind the annotation is to represent the components of a triple as a blank node which has `rdf:type` of `rdf:Statement`, i.e. the RDF reification mechanism is used as the base unit of the annotation. Further annotation elements from the AAT schema are then used to add the pertinent information for AAT as properties of this blank node, Figure 2.1a shows an example triple and Figure 2.1b its AAT annotated equivalent.

As can be seen in Figure 2.1b AAT uses a number of predicates to annotate the triples sufficiently for it to be able to both produce versions of the original RDF and to compute reports on changes in the RDF. The role of each of these predicates is explained in the following list:

- `aat:firstAsserted` indicates the date at which AAT first asserted this triple into the local profile of the URI. This is used to find new triples and for versioning.
- `aat:lastAsserted` indicates the date at which AAT last asserted this triple into the local profile of the URI i.e. when did we last update the profile and see this triple. This is used to find new triples, missing/deleted triples and for versioning.



(A) Original Triple

(B) AAT Annotated Triple

FIGURE 2.1: An Example Triple and it's AAT Annotated equivalent

- `aat:retractedOrDeleted` indicates whether AAT considers the triple to have been retracted/deleted from the profile of the URI. This means that the triple was seen in the past in the profile of the URI but has been missing for sufficiently long that it is now considered as a retracted/deleted triple. This is used for retracted/deleted triples and versioning.

- `aat:changed` indicates whether AAT considers the triple to have changed in the sense that there is now a newer triple with the same subject and predicate but a different object which supersedes this triple. This is only relevant where the particular predicate has its cardinality restricted to 1; in the case where a profile contains many triples with the same subject-predicate pair it is impossible to tell if the value of one of those has changed since each triple is effectively indistinguishable. For example if you have triples A, B and C all of which have the same subject-predicate pair you have three distinct facts, if when you next update the profile you get A, B and D you cannot say C has changed to D since D is just an expression of one of many values which may be given for that particular subject-predicate pair. If on the other hand you had a single triple A which when the profile was updated became B you can say that A changed to B since you know that it is only permissible to have one value for the given subject-predicate pair.

- `aat:source` indicates the URI of the original RDF that the triple came from.

Currently this permits only one value

Given that triples are represented in the annotated format as reified triples it proves to be very easy to export back to the original triples. For each subject which has `rdf:type` of `rdf:Statement` it is easy to select the triples in the profile which define the `rdf:Subject`, `rdf:Predicate` and `rdf:Object` of that triple and recreate the original triple. Since a profile may contain multiple versions of triples there is some logic involved in selecting which annotated triples will need exporting; this is explained more in Section 2.1.4.

2.1.3 Change Reporting

One of the core requirements of All About That is that given a profile which has been updated at least once since its creation AAT should be able to tell the user what has changed in that profile at the most recent update. Given that the profile is stored as AAT annotated triples it actually proves to be relatively simple to find different changes in the profile simply by examining the annotations. Four possible types of changes are looked for:

1. New Knowledge - Triples that are newly asserted in the profile
2. Altered Knowledge - Triples whose object has changed where we know that the subject-predicate pair has a cardinality of 1 i.e. only 1 triple with this subject-predicate pair may exist
3. Missing Knowledge - Triples that are no longer present in the source RDF but which have been seen in recent updates but not the latest update
4. Retracted/Deleted Knowledge - Triples that are no longer present in the source RDF and which haven't been seen for a sufficient period to consider the facts they represent to be defunct.

Listing 2.1 shows the pseudocode for the algorithm responsible for creating the change reports. In the actual implementation this is done by first querying over the profiles RDF graph to find triples that satisfy the relevant conditions that indicate they may represent changes in the RDF before more specific conditions are evaluated. This is purely a design choice since it makes little difference efficiency-wise since both approaches require iterating over the graph contents. Change reports viewed in the web interface are always generated on the fly and currently are not persisted to storage in any form; when a user requests a change report it is based on the profile at that exact moment in time. Figure 2.2 shows an example of the change report as presented to a user viewing it through the web interface.

```
1  new = changed = missing = deleted = {};
2  lastUpdated = Profile.LastUpdated;
3  foreach (Triple t in Profile.Triples) {
4          if (t.DateCreated == lastUpdated && t.DateUpdated == lastUpdated) {
5                  versions = Profile.VersionsOf(t);
6                  if (versions.Count == 1) {
7                          //This is the only Version of this Triple
8                          new.Push(t);
9                  } else {
10                         if (Profile.AllowsMultipleValues(t.Predicate)) {
11                                 //New Fact on this Subject-Predicate Pair
12                                 new.Push(t);
13                         } else {
14                                 //Altered Fact about this Subject-Predicate Pair
15                                 changed.Push(t);
16                         }
17                 }
18         } else if (t.DateUpdated < lastUpdated) {
19                 if (t.Changed == false && t.RetractedOrDeleted == false) {
20                         if (t.DateUpdated < (lastUpdated - MISSINGTHRESHOLD)) {
21                                 //Been missing for sufficiently long to be
       considered Retracted/Deleted
22                                 deleted.Push(t);
23                                 Profile.MarkRetracted(t);
24                         } else {
25                                 //Missing
26                                 missing.Push(t);
27                         }
28                 }
29         }
30
31 }
```

LISTING 2.1: Pseudocode Algorithm for Change Reports

FIGURE 2.2: Change Report Web User Interface

Change reports can also be generated as a simple text file by the Update Service if the user enabled this option. This means that reports can be preserved over time creating a history of changes since as already mentioned change reports are normally generated on the fly.

2.1.4 Versioning

All About That also allows users to query the system to view what AAT perceived the Profile of a URI to be on a particular date - this effectively allows users to see versions of the source RDF. Versioning is done primarily by extracting triples from the profile whose `aat:firstAsserted` and `aat:lastAsserted` annotations indicate that they existed in the RDF on the date the user specified. There is also a need for some additional logic regarding inclusion of triples that have since been retracted/deleted and triples that have changed. The changed case seems tricky on paper as it is possible that you may get multiple triples where the RDF should permit only 1 triple with that subject-predicate pair. In practice though this never actually occurs since during a profile update if a triple changes then we don't update the old version's `aat:lastAsserted` property. This means you can't ever get the situation where you have two versions of a triple marked changed where the `aat:lastAsserted` of one matches the `aat:firstAsserted` of the other and thus both would be selected by a version from that date and time.

For the case of the basic export where the user wants to see the RDF that All About That saw at the last update - which is in essence the latest version - it is much easier. Retracted/Deleted triples can be ignored completely in that case since they do not form part of the latest version and neither do any changed triples. It's sufficient just to select annotated triples where `aat:lastAsserted` matches the profiles last updated date and `aat:retractedOrDeleted` and `aat:changed` are both false.

2.1.5 Testing

In order to test All About That I started by profiling a small number of ECS People RDF[2] documents including my own. This meant that I could introduce changes into the RDF by editing my ECS profile which causes the RDF to be updated and thus allows me to see if AAT was correctly detecting the changes. Testing so far has shown that changes are detected and reported correctly and that the system handles updates well.

In terms of scalability the prototype can handle large numbers of profiles effectively; currently I have profiled all the URIs from the Southampton RKBExplorer[3] which are URIs for people with current ECS IDs. By this I mean that they are current members of ECS who have corresponding ECS People RDF documents. With this and other data imported I have approximately 516,000 triples which annotate the RDF originating from about 1050 URIs taken from ECS People, Southampton RKBExplorer and a small assortment of DBPedia[4] documents. There is noticeable performance impact on the web interface on initial loading and when performing some tasks but most of this is mitigated

[2] http://id.ecs.soton.ac.uk/docs/
[3] http://southampton.rkbexplorer.com
[4] http://dbpedia.org

by use of caching. Average initial load time is currently in the range of 10-15 seconds for this interface which includes application start-up time.

Chapter 3

Proposals

3.1 Extensions to All About That

My research focus currently is looking at ways to extend All About That to see both how well it scales and what can be added to it to increase the link integrity it can potentially provide. One of the first extensions I'm interested in is creating profiles that are based on multiple URIs so that the concept of profiling matches its name more. It would be more useful if you could enter multiple URIs that represent the same thing and AAT would build a combined profile of the RDF from all those URIs and monitor it over time. This would be especially useful in the academic domain where often there are many sources of RDF about a person/concept such that monitoring the combination of those would have value both in seeing which sources change their data regularly and in preserving the integrity of the data. The current RDF Schema for AAT (see Appendix C) easily supports this since it already defines `aat:source` meaning that annotated triples can be associated with the source they come from and it has `aat:profileSource` which could be used to specify multiple sources for a profile.

In terms of scaling it is mainly a case of optimising code where possible to speed up operations over larger datasets and load-testing by importing increasingly large amounts of RDF into the prototype system to see at what point it fails/becomes unusable. For example I have recently significantly improved the performance of reading from a SQL Store gaining an 8 fold speed increase by improving use of caching and threading. Performance figures can be seen in Section B.3.

3.1.1 Enhanced Change Reporting

One limitation of the current prototype is that change report information is only made available to users in a human readable format. Ideally this information should be machine readable such that other applications can consume the data and find out which

profiles have changed and how they've changed. These applications could then either use this information to inform their own data updates or present it in some form to their own users. In the first case there is potential for looking at how & if AAT could be integrated into projects such as ReSIST where it is important both that the data is up to date and that the provenance of the data can be recorded.

As described in Section 2.1.3 the current change reports are also limited since they are generated on the fly and document only the changes in the profile that were observed at the last update. Though to a certain degree they do look further back when considering triples that are missing or retracted/deleted. It would be useful if a user could request two versions of the profile and have a change report generated indicating the changes between the two, in essence a `diff` of the versions. Incorporating this into AAT would allow for the generation of detailed revision histories of profiles since you could easily produce a sequence of versions and from those a sequence of change reports. This would give the user much greater power in examining how the data changes over time.

3.1.2 Using a Co-reference Resolution Service (CRS)

Related to the idea of building a single profile from multiple URIs and the original motivation for the prototype (see Section 2.1) is the possibility of using CRS data in AAT. Instead of a user having to provide multiple URIs that comprise a profile they could provide just one and the system would use CRSs to discover equivalent URIs and include them in the profile. This removes the onus on users to know all the URIs that describe a thing and allows them to easily build complex profiles from just the one URI they know. The existing web interface already includes an experimental feature which takes the profile URI and looks it up using the CRS provided by RKBExplorer[1] and presents the results to the user.

3.2 Other Research Directions

The other possible research direction to take is to look at developing a JIT resolution service for missing Semantic Web documents in the style of the Opal system developed by Harrison and Nelson (2006). Since there are now a number of effective semantic search engines and an increasing number of public SPARQL endpoints it should be possible to develop a system that can find information about a particular URI even if that URI is not dereferencable i.e. it's a 'dangling-link'. The basic idea would be that even if you can't dereference a URI directly you should be able to rebuild the RDF that was originally there (assuming of course that at some point there was RDF there) and use this as necessary in your Semantic Web application.

[1] http://www.rkbexplorer.com/sameAs/

Appendix A

Literature Review

A.1 Key Papers

The Key Papers in Link Integrity can be divided into two main groups - Surveys and Proposed Solutions. The Survey Papers tend to outline the issues and give possible solutions without any concrete implementation details though they may well refer to existing examples of other work which uses the solution. The Proposed Solutions are descriptions of actual systems that people have implemented in an attempt to solve some of the issues in link integrity. Key Papers covered in this section were all written prior to 2003, anything from the last 5 years is considered as Current Research and covered in Section A.2. Since the Semantic Web only really began to truly emerge as an actual system from 2001 onwards with the publishing of an article in Scientific American (Berners-Lee et al., 2001) all the papers relating to link integrity on the Semantic Web are also covered in Section A.2.

A.1.1 Survey Papers

A.1.1.1 Davis's Work

One of the most frequently referenced survey papers covering Link Integrity is Davis (1998) which provides a comprehensive survey of the issues in link integrity with particular reference to Open Hypermedia Systems. This paper is useful in that it doesn't just discuss the issues but gives an excellent introduction to all the key concepts such as link storage models. Link storage models are important since the model used actually effects how link integrity will affect a given system, in brief the models are as follows:

- *Embedded Link Model* - In the embedded link model all link information is embedded in the document's data, this may include specifiers for an anchor within a

document as well as the actual document to link to.

- *External Links with Embedded Anchors* - In this model the link information is stored externally to the documents. Links may specify anchors in a document and the actual anchors are embedded in the documents data.
- *External Link Model* - In the external link model all link information including anchors is stored externally. Anchors are typically specified in some document specific way to point to appropriate sections of the document without requiring any data embedded in the document.

These models are also explained very briefly but effectively in his 1995 article (Davis, 1995b) in the Communications of the ACM.

The paper then goes onto cover a list of possible solutions to the two problems giving a description of each and brief comments on their actual/likely effectiveness as a solution. Most interestingly the paper concludes with an elucidation of the five main philosophies on dealing with link integrity in Hypermedia systems which are:

- *Don't Bother* - the philosophy that most people seem to adopt which is that broken links aren't worth the effort of fixing
- *Avoid the Problem* - use methods for resolving links and anchors that are resistant to changes in documents
- *Loosely coupled* - the system provides tools which allow a user to maintain the link integrity manually
- *Automated link repairs* - the system provides tool which automatically perform Just-in-time link repairs when necessary
- *Tightly coupled* - a system owns all the information contained within it and thus can enforce link integrity

The Proposed Solution papers that are covered in the next section all take one of the latter 3 philosophies since the first philosophy has no value to research and the second is more related to a much wider issue of Network architecture and addressing. While the second philosophy is part of wider issue possible solutions based upon it are often mentioned as possible panaceas to the problem of Link Integrity. A persistent naming scheme would solve the 'dangling-link' problem completely though this would of course leave the much tougher editing problem as an outstanding research issue.

A much briefer discussion of the problems and solutions can be found in the later paper Davis (1999) which considers the World Wide Web more but is considerably less detailed and so rarely referenced. At the same time a more detailed version of much of the

contents of this paper can be found in Davis (1995a) which has the added value of discussing the issues by example through how the problems affected and were (partially) solved/addressed in the Microcosm system (Davis et al., 1993; Fountain et al., 1992).

A.1.1.2 Other Work

Another key survey paper is Ashman (2000) in which she discusses Link Integrity in similar detail to Davis (1998). In her paper she gives more detail than Davis about how link integrity errors can occur even when the links are contained within a stable document collection e.g. an archive. A distinction is made between whole-document changes - moving/deleting the document - which may break a link completely ('dangling-link' problem) and part-document changes - editing the document contents - which may break anchors in or change the meaning of the target document (editing problem).

This paper is useful since like Davis's it gives good explanations of the possible solutions but it makes more reference to actual Proposed Solution papers in which researchers have tried and evaluated the solutions. Additionally a further distinction is made between corrective and preventative solutions which Davis doesn't make clear. This is actually important in understanding the value of any solution since preventative solutions are preferable but typically far harder to actually implement than corrective ones. While various preventative solutions have been tried the solutions actually applied outside of research are primarily corrective ones.

A much earlier paper which gives a more simplistic survey focused primarily on comparing three rival Hypermedia systems is Vanzyl et al. (1994). The paper is a general comparison of the Microcosm (Fountain et al., 1992), HyperTED and World Wide Web (Berners-Lee et al., 1992) Hypermedia systems which includes coverage of how the systems manage link integrity. The conclusions of the paper cover a few of the possible solutions in brief and it makes the obvious point that the World Wide Web is far more susceptible to link integrity issues than other Hypermedia systems due primarily to its distributivity.

There are also a variety of statistical surveys on link integrity which cover the extent of the problem by seeing how long URLs remain valid before becoming broken. These provide insight into how widespread link integrity problems actually are even in well authored material like Scientific Literature. One of the best examples is Spinellis's study (Spinellis, 2003) which found that of the URLs contained in the entire output of Communications of the ACM and IEEE Computer for the years 1995-1999, only 72% of links remained operable. His initial results also showed that within 4 years of publication almost 50% of links are broken and that even within a year of publication around 20% of links would be broken. This survey is of value since he repeated the experiment two years later and found that 4 year old URLs were actually heading towards a 60% failure

rate. From this it is safe to say that we cannot readily guarantee that any URL will last even a relatively short amount of time, Tim Berners-Lee's vision of URIs remaining constant for decades/centuries as espoused in writings like Berners-Lee (1998) proves to be highly unrealistic.

A.1.2 Proposed Solution Papers

A.1.2.1 flood-d and p-flood

In Danzig et al. (1994) the authors detail their flood-d algorithm which was designed to support emerging web technologies since the authors noted that replication was key in the effectiveness of technologies like NNTP. They hypothesised that future services would require massive replication and that more efficient algorithms for this would be required, therefore they designed the flood-d algorithm. Flood-d itself has no direct relevance to link integrity since it is designed purely for replication but the paper is important in that its demonstration of the potential effectiveness of this class of algorithm would influence Kappe's algorithm design in Kappe (1995).

Kappe proposed a system whereby each participating server would maintain an independent linkbase containing all the links in & out of the server. Documents which contain links to external servers are referred to as 'surface' documents; it is only these documents with which Kappe's system is concerned since he assumes that a site manages its internal link integrity itself. Links to external servers are referred to as 'surface' links and the information on these has to be stored at both involved servers - which incidentally provides bi-directionality of links as a side effect - as well as meta data about the documents which link to them. When an update to surface documents or links occurs the server where the change occurred needs to notify the other servers in order that they can update their linkbases and take any necessary action. To achieve the updating he uses an algorithm called p-flood which probabilistically propagates update information through the network of all participating servers. As stated in the paper this provides for *weak consistency* of links since there will be periods when parts of the network are inconsistent e.g. during update propagation or when a server is unavailable since the servers yet to receive the updates can still have links to documents that no longer exist.

This proposed solution is valuable since it or variations upon it can be implemented for any set of servers which manage their internal integrity. While Kappe's system was designed primarily for the Hyper-G (Kappe et al., 1994) network it can be applied to other systems like the World-Wide Web, though as servers are required to manage internal link integrity only portions of the current web could be enabled with this system.

A.1.2.2 WebLinker

WebLinker (Aimar et al., 1995) was a system proposed in 1995 which aims to manage link integrity on the server side by introducing a logical naming scheme for resources. The idea of this was to allow the physical locations of resources to change without breaking integrity since the server would interpret the logical names of resources and direct the user to the appropriate resource. Their naming scheme - Local Resource Names (LRNs) - is URI based and each LRN is composed of some logical name for the group of resources to which something belongs and a Fragment identifier which points to a particular resource/part-resource within that group. The system contains a tool which will analyse normal HTML documents and convert links in them to LRN based links. The server then maintains a table of LRN to physical locations in order that it can resolve requests for resources specified using LRNs (achieved via CGI scripting).

The problem is that no alternative naming scheme for resources has yet been produced that sufficiently large numbers of people are willing to use such that it supersedes/upgrades the existing URL scheme. Other naming schemes have been proposed such as URNs (Moats, 1997) and Persistent URLs (OCLC, 1995) but like the LRN scheme they have never been widely adopted, though PURL is beginning to be used on the Semantic Web (see section A.2.4.1). Naming schemes have the potential to solve the 'dangling-link' problem providing that the registries that map persistent names to actual locations are well-maintained; unfortunately it is this maintenance overhead that is one of the key barriers to adoption. The other problem with such approaches is the need to rewrite all links to use the alternative naming schemes in order that the links gain the benefits of the Link Integrity provided by the scheme.

A.1.2.3 Author-oriented link management

Creech proposed a system (Creech, 1996) whereby the emphasis is on human input from document authors to correct detected link integrity issues. The system is based on web crawling combined with a change log table (CLT) which is used to record changes to pages which may affect links. When the web crawler looks at pages it checks each link, where it finds broken links it consults the CLT to see what has happened to the page in question. The human input element of the system comes from the fact that in many cases the system either doesn't know what happened to a document, the change to the document doesn't permit for automatic correction of the issue, or it doesn't have permission to alter documents itself and therefore needs to ask a human to take corrective action.

The major flaw of this solution is that human input is too heavily relied upon, the ideal aim of Link Integrity is for the process to be handled entirely by the system without recourse to human input. There are a number of other flaws inherent in this proposal

including *a*) the need to constantly crawl the website; *b*) the inability to deal with dynamic content since it changes all the time (CLT approach is useless for this); and *c*) multi-website implementations of this require complex coordination. While there is an argument to be made for the role of authors in maintaining Link Integrity it's unrealistic to expect authors to constantly update their documents since often a document is created, published and then becomes effectively unchanging. This is quite normal particularly in small independently maintained sites and is akin to the Publishing model described in (Davis, 1998), although the full model does call for all authors to make their work read-only and versioned. Author involvement in maintenance is best done at design and publish time when the author is most able and likely to correct link integrity problems pointed out to them.

The solution is also unable to deal with dynamic content as it relies on being able to detect when content changes. Since dynamic content by definition is constantly changing the system cannot cope even though most changes will likely be minor e.g. displaying the current date on the page or the current user's Login name. Potentially this approach could be adapted to better deal with dynamic content by using *diff* or some other comparison algorithm to detect the size of changes and only add entries to the CLT when significant changes occur. Yet in reality this approach still has too many other flaws to be truly viable for application on the scales necessary to solve the link integrity problem on the web.

A.1.2.4 W3Objects

In Ingham et al. (1996) they proposed the W3Objects approach as a way of solving the problem of broken links. In their approach all resources on the web are encapsulated in Objects which have well defined standard interfaces – objects may be polymorphic in that they implement multiple interfaces e.g. HTTP and FTP. In this model links become references to Objects which means that link integrity becomes a question of reference integrity. All Objects maintain a count of how many incoming references they have in order to support the migration and deletion protocols in the W3Objects model.

When a document is moved (migrated) on the web a responsible Administrator might leave a HTTP redirect to the new location, but they have no way of knowing when they can safely remove this redirect since they have don't know who is referencing that document. By treating Links as References the HTTP redirects concept can be replaced with one of Forward References, when an object is moved a forwarding Object is left in its place which references the new location of this Object. In this way clients can still find the existing Object but they can also learn its new location and update their references such that they don't need to follow the forward Reference in future but can access it directly; Ingham et al call this 'Short-cutting'. Like all objects the forward Reference object keeps track of incoming references, when all referees have bypassed it

(by short-cutting) and it is no longer referenced it can be removed from the system.

When a document is deleted on the web Administrators typically do nothing although they could in theory leave a HTTP 410 Gone in its place. In W3Objects when an Object is deleted a 'Gravestone' Object is left in its place which informs referees that the Object has been deleted and that they should no longer reference it. As with all Objects the gravestone tracks how many incoming references it has and once this drops to zero it can remove itself from the system.

While this model is a very effective one in maintaining link integrity its main limitation is that it calls for an architectural reworking of the web which even in 1996 was already becoming too large for that to be truly practical. It would be relatively easy to implement this as a back-end for your Hypermedia system and then have a Web Server which translates objects to HTML for unaware clients but then you'd lose the actual benefit of the system. In addition the system still relies on using URLs for addressing objects so it's still possible for the URLs to break outside of the system e.g. Domain renaming. The concepts of the model are essentially sound but they are just not sufficiently robust and scalable to be applicable to the real web.

A.1.2.5 Agents

Moreau and Gray (1998) proposed an Agent based approach to maintaining link integrity which relied on a variety of types of agents interacting to ensure the availability of resources as long as someone was linking to it. At the centre of this approach is a concept of Publication Contracts whereby the author of a resource agrees to publish it for a fixed/indeterminate period of time allowing end users to assume that the publisher will hold to this and notify them should the resource be updated/removed. Agents for Users, Authors and Administrators provide information about who is using what resources and handle notifications of various parties of events affecting them e.g. User bookmarking a resource. The Agents also provide for versioning of resources so that a user can always retrieve the particular version of a resource they linked to/bookmarked. In terms of implementation the approach is achieved by the use of web proxies on both the client & server sides in addition to browser extensions.

The problem with their approach is that like with W3 Objects (section A.1.2.4) it requires an architectural reworking of the web. While the addition of web proxies and browser extensions is not as extreme as Ingham et al's approach it does place a large additional overhead on the network since many proxies have to be introduced which would slow traffic. The real problem with the approach is the idea of Publishing Contracts – much of the web is essentially anarchic, created by those who have the desire, time & money to publish all sorts of information. While large chunks of the web are organised & well maintained the vast majority isn't and doesn't necessarily want to be.

Publishing contracts would never work unless everyone who publishes on the web abides by them which is clearly never going to happen. Another criticism of this approach is that like other work it has a degree of reliance on replication of data in order to ensure link integrity. If an effective solution to the Link Integrity problem is found it should avoid replicating data to give the appearance of reliability and instead concentrate on making links reliable in the first place.

A.2 Current Research

A.2.1 Replication

As previously mentioned (see Section A.1.2.5) several pieces of work have used replication as part of their strategy for maintaining link integrity. By replicating the parts of the web that people are linking to and tracking what is being used you can effectively maintain resources as long as they are in use.

A good example of this is Veiga and Ferreira (2003) work on their RepWeb system and their subsequent paper on Knowledge Repositories (Veiga and Ferreira, 2004). In the RepWeb system they class resources into two groups: 1. HTML content documents; and 2. all other content (which cannot contain links) . The memory of the system is therefore organized as a distributed, partially replicated graph of web resources connected by references (links). As a user browses the web they use browser extensions to indicate which parts of the web they want to replicate, they can then work with these replicas locally and if desired publish the altered versions via their own server. The server side of the system uses Java servlets which handle the requests for replication; it is these servlets that are able to track what resources are in use and ensure they remain available as long as references to them exist.

Their algorithm for maintaining integrity is based on Distributed Garbage Collection and uses reference listing and tracing. It must ensure that resources remain available as long as they are referenced and it follows strict rules to do this. A resource can only be deleted if the union of all replicas of the source objects (those which have previously referred to the target) no longer contains any references to the target object. Note that one advantage of this approach is that a RepWeb server appears as a completely normal Web Server to unaware clients and so doesn't have the same architectural issues as previous work.

Subsequently they produced a paper on Knowledge Repositories (Veiga and Ferreira, 2004) which extends their work from RepWeb to focus more on replication and preservation of dynamic content. This is an area which brings new issues for Link Integrity since with dynamic content a resource may be different every time it is accessed and the parameters which cause a given configuration may change over time. This means a link

to a dynamic resource can become broken because the parameters it provides are no longer valid for the system generating the dynamic content even if the dynamic resource itself exists – essentially the Editing problem (see Section 1.2). Therefore they extend their previous model by using web proxies to cache specific versions of dynamic content as long as they are being referenced. As with RepWeb the user makes use of a browser extension to indicate they want to preserve a particular web page and the system takes care of implementing this. This research is impressive as it lays the foundations for a system which potentially solves both the issues in Link Integrity with only a minimal performance cost to the user of approximately 12ms increase in retrieval time.

A.2.2 JIT Resolution

JIT (Just-In-Time) Resolution refers to systems which don't rely on any prior warning of a link being broken and simply try to 'fix' it in some sense in real time when the user requests it. In practice this generally means attempting to locate a new location for the requested resource or retrieving the resource from a cache/archive. The systems described in this section are examples of current variants on JIT Resolution of broken links.

A.2.2.1 Lexical Signatures

One current approach to JIT resolution involves the development of a method to make hyperlinks more robust by attaching additional information to them. Phelps & Wilensky proposed in their Robust Hyperlinks paper (2004) the idea of adding Lexical Signatures to hyperlinks. A Lexical Signature is essentially a number of words/terms which capture the content of a given document such that when the URL no longer functions this signature is sufficiently unique to the resource to allow content based relocation of it. The signatures they use contain 5 terms since their experiments showed this to be sufficient. Terms are chosen by heuristic analysis of the document and favours rare terms which are used several times; terms are also chosen such that they represent the whole of the document i.e. terms located close together won't both be used in the signature. This allows the signatures to remain relevant to the document even when it is changed although there will always be a threshold of change beyond which the signature becomes irrelevant.

Hyperlinks are made Robust by adding the lexical signature as a query string argument to existing links. In the event that the URL fails a Robust Hyperlink aware browser/server will take the lexical signature and submit it to several search engines in an attempt to locate a new location for the document. Several search engines are used so that the system can pick the most likely new location of the document based on search engine consensus since different engines have different search and ranking algorithms.

While this approach works well its main problem is that the extra information has to be added to all hyperlinks ahead of time, meaning that not only do existing documents have to be adapted to use the Robust Hyperlink syntax but that in order to attach appropriate lexical signatures to links you must pre-compute the signatures of the document being linked to. While this process can be automated, applying this technique over the entire web would be impractical; in addition if a document changes significantly then its signature will change and all links to it require updating. In essence by solving one link integrity problem you'll have created another which is not at all desirable. Another limitation in this approach is that you can only really generate signatures for documents which can be full text indexed since you are using full text search engines to do the content based relocation should the URL fail. Even when you can generate signatures certain documents are unsuited to the process such as small documents and frequently changing documents. The approach only offers a partial solution to the Link Integrity problem since it cannot do anything to relocate documents if the document has been deleted rather than moved.

A.2.2.2 Opal

Harrison and Nelson (2006) created Opal which is a JIT system that uses the ideas of Lexical Signatures from Phelps and Wilensky (2004) but applies it JIT rather than requiring pre-computation of signatures. They compute Lexical Signatures for resources by using cached copies of pages from major search engines at the time when the system is asked to relocate a broken URL, once computed signatures are stored for future reuse. In the event that there are no cached copies of a resource then the system cannot compute a lexical signature and will be unable to offer the full range of relocation options.

To make use of this system all webmasters need do is customise their Custom 404 error page to redirect users to an Opal server supplying the URL that the user was attempting to access. An Opal server has a user friendly web front-end which presents the user with their options for relocating the resource they were looking for. Opal allows users to select from Cached & Archived copies of the Resource and query search engines to find possible new locations for resources. Interestingly the system is designed to learn about new locations for content by allowing users to vote on whether new locations it finds are relevant/irrelevant, then if asked to relocate the same URL in the future the system can show & rank results by previous user votes. In order to facilitate the learning process multiple Opal servers can replicate information between themselves using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH). This replication allows servers to learn about new locations for broken URLs, lexical signatures and user votes increasing the effectiveness of the Opal system as a whole since it reduces learning overhead on individual servers.

This is a much more effective implementation of the Lexical Signatures idea and requires

far more minimal adjustment to websites to make use of the system than the full Robust Hyperlinks concept. Opal does still have the problem that it is limited in what it can do to resolve URLs to resources which have simply been removed from the web. Unlike Robust Hyperlinks it can offer archived versions of the resource but that still relies on the resource having been on the web for a sufficient period of time such that it actually gets archived. It also suffers from the other problems associated with Lexical Signatures as described in A.2.2.1 on page 21.

A.2.2.3 PageChaser

Morishima et al. (2008) recently proposed the PageChaser system which is a monitoring based approach to link integrity management and JIT resolution. Users register the URLs of pages whose links they wish to maintain with a Link Integrity Manager which then scans those pages to locate the links in them. It then regularly checks that those Links still work and when it finds a broken link invokes the actual PageChaser tool to fix the link. PageChaser is a bot which uses a series of heuristic rules to crawl what are perceived as being likely locations for the resource to be found. Part of this is done by looking for pages which they term 'Link Authorities'; these are frequently updated pages containing large numbers of links e.g. tables of contents or site maps whose links can be checked and seen to be reliable. Over time PageChaser learns of link authorities for given domains and will use these in the future when it needs to relocate resources which it considers likely to be located in that domain.

The main drawback of this approach is that it is fairly simplistic in terms of where it will look for content since it primarily assumes that content is only relocated locally i.e. within a domain or that an entire site moves domain. Unlike the other two systems presented in this section it doesn't use search engines which severely limits its ability to actually relocate content. The concept of Link Authorities is interesting but adds a requirement for an analytic assessment of web pages to decide whether they can be used as Link Authorities, while in practise the use of search engines would be less costly and likely more effective. They are considering to pursue this work and recently presented a poster on this system at WWW 2009 (Morishima et al., 2009) though little progress appears to have been made since their original paper.

A.2.3 Autonomous Systems

Bustard et al. (2007) take an alternative approach to Link Integrity considering it conceptually through use of Soft Systems Methodology. They use this to develop a notion of how link integrity fits into the general concept of an organisation's website and the actual goals an organisation wants to achieve with it e.g. minimise broken links. From this model they propose an Autonomic Link Integrity manager which is able to identify

and repair broken links automatically using various methods as described in other papers in this review. Though they note that a fully autonomic system is not possible since there are some situations which will still require human intervention. To achieve this the autonomic systems needs some knowledge of its environment such as which human owns which website and who are the administrators and content providers for a given website.

In the paper they describe a prototype system called LinkMan II which generates reports of Broken Links found on a given site and then allows a user to work through this report and correct the problems semi-automatically. Somewhat ironically this prototype is not an autonomic system since it requires outside intervention to complete its task. The paper serves more as an outline of their proposed research directions than as evidence of a potentially viable solution. They do produce some interesting ideas with regards to how a autonomic system might find and correct link integrity problems. This includes allowing the system to analyse the web server logs to see what incoming links are broken and to take appropriate actions – whether this being contacting the external site owners or providing redirects to appropriate local content. Also considered is whether it would be possible to detect information that is outdated in terms of content and links and repair such material. Finally they consider whether a full blown versioning strategy wherein a full history of all pages is kept would aid recovery of broken links.

Their ideas for possible approaches are interesting but as yet they have not produced any further work showing these ideas actually in action.

A.2.4 Persistent Naming Schemes

It has been proposed several times over the years since the link integrity problem became apparent that one way of solving the problem would be to replace the existing naming scheme with a scheme of persistent and permanent addressing. The major W3C effort Uniform Resource Names (URNs) (Moats, 1997) has never really got to a stage where it could be an acceptable solution since the scheme called for all URNs to be centrally administered by ICANN or some other central registry. This goes against the very nature of the web which is decentralisation and freedom of publication that the web was founded on and thus such a scheme will never be widely adopted. Despite this failure other schemes have proven to be useful such as PURL.

A.2.4.1 PURL

Persistent Uniform Resource Locators (PURL) is a service provided by the Online Computer Library Centre (OCLC) which provides persistent naming for resources on the web (OCLC, 1995). A PURL is basically a URL alias for some actual URL, so that

when you access a PURL the service resolves it into an ordinary URL. The benefit of this is that the owner of a PURL can change the URL that the PURL points to as the resources move/change over time without the end users who make use of the PURLs being aware of this. PURLs can also be used to achieve more complex redirections such as creating entire PURL based domains through the use of the PURL services partial redirection concept. This has the potential to make URLs more reliable since an author who makes use of PURLs in the first place is far more likely to maintain the resources being targeted and the relevant PURL records. Also as more people adopt the PURL system more PURL services (other than the OCLCs) are likely to come online and there will be more ways to ensure the robustness of URLs.

A.2.5 Coreference Resolution Systems

Coreference Resolution Systems are a class of software for the Semantic Web (and also in natural language processing) which are designed to determine which URIs refer to the same concept. This is useful since most sources of Semantic Web data assign their own URIs to particular things and Semantic Web applications often need to know what URIs are equivalent in order to produce useful results or to increase the accuracy of their results.

A.2.5.1 Entity Naming

One approach that has been suggested is an Entity Naming Service (ENS) which is very similar in principle to the PURL service described in the preceding section except that it provides the ability to search for the URI of an entity to see if one already exists and generate a new URI if not (Bouquet et al., 2008). They believe that everyone should use centrally issued URIs for their Semantic Web applications in order that all applications can easily reason across Semantic Web data. As with systems like PURL and URNs you have the issue that the system is centralised and semi-regulated which is at odds with the web's decentralised deregulated nature and thus there has been limited take-up of ENS to date.

A.2.5.2 Coreference Lookup

A common approach is simply to create lots of `owl:sameAs` links asserting that various URIs are considered to be the same in your system. Semantically `owl:sameAs` implies that the things represented by the two URIs are exactly identical when in most contexts of use this is false and for this reason many Semantic Web researchers are against using it in this manner. Once two URIs are defined as `owl:sameAs` then they become indistinguishable to a Semantic Web application which is not desirable if the two things

had different distinct properties which are lost by asserting `owl:sameAs`. For examples of systems that use this approach see the W3C Wiki page on Equivalence Mining (Bizer, 2007).

Those researchers against the previous approach have proposed CRS as a solution in a number of papers (Glaser et al., 2007; Jaffri et al., 2007, 2008a,b) which are related to the ReSiST[1] project. CRS is a service that is complementary to a Semantic Web application which provides information about equivalent URIs in a way that isn't as semantically strong as `owl:sameAs`. A CRS provides information about URIs in bundles of URIs; all the URIs in a particular bundle are considered to be equivalent. It is also possible to indicate which URI is the canonical URI i.e. the one that should preferably be used to refer to the thing in question. CRS aware applications like RKBExplorer (Glaser and Millard, 2008) can use this information when they reason over Semantic Web data to find all the data that refers to a particular URI by finding all the equivalent URIs and retrieving data about them as well as the original URI provided.

A.3 Research Directions

Future Work in Link Integrity can be seen as having two key focuses with a certain degree of cross-over of techniques and approaches. Firstly there is the continuation of existing research into solutions that we can deploy to better maintain link integrity on the traditional hypertext web. Secondly is a new branch of research into how link integrity can be applied on the Semantic Web where we are primarily interested in heavily interlinked data. In this section the issues that are active areas of investigation in the research community are explored and the likely future challenges are considered.

A.3.1 Hypertext Research

In Hypertext Link Integrity there are three main themes of the research going forward:

1. Improved JIT Resolution

   Systems like Opal (see Section A.2.2.2) show that JIT resolution can be an effective solution to the 'dangling link' problem though it's primary drawback is that is only fixes links after they break. Despite this it is clear that JIT resolution can be useful to end users and so there is a need for further work in improving the ability of systems like Opal to relocate moved resources. One challenge in this area will be in agreeing a common style of user interface for such systems so that when users encounter one they know what to expect and how to use such a system to relocate the desired resource. There is also the potential to conduct research into how such

[1] http://www.resist-noe.org/

systems can handle finding cached copies of resources which have been deleted from the web by expanding their use of archive services. Other research in this area may look at how JIT resolution can be applied to non-textual resources such as video and images.

2. Persistent Naming Schemes

   As discussed earlier in Section A.2.4 the use of a Persistent naming scheme has the potential to mostly solve the 'dangling link' problem. Systems like PURL are proving to be very effective and the W3C and IETF continue to work on their URN scheme, in fact PURL is designed such that an easy transition to URNs will be possible in the future. In terms of research there is little need for further technological development in this area and it is more a case of promoting widespread adoption of these schemes. URNs will remain an active topic of discussion though whether they will eventually replace URIs remains to be seen.

3. Automated Validation and Correction

   Ideally link integrity is something that should be automatically maintained without the need for human intervention - systems should regularly check their links and make automatic corrections. While some modern web applications are capable of doing this within their own content there is a need to research into how such systems can be effectively scaled to a multi-website/whole web scale. In order to do this there will potentially need to be some research into semantic analysis of pages such that links can be checked to be contextually correct (the Editing Problem) as well as physically correct (the 'dangling link' problem). There has been little research to date into solving the Editing problem and this is the biggest barrier to achieving large scale automated validation and correction of links.

In general there is a need to investigate whether natural language processing and semantic analysis techniques being developed for use on the Semantic Web and bootstrapping of existing data to the Semantic Web can be used to solve the Editing problem. As should be apparent in this literature review very little work has been done to address the editing problem beyond things like versioning/replication (as seen in Section A.2.1) to preserve old pages so the intended version of a page can always be retrieved. While this behaviour may be useful in some cases it is not always what the author of the resource being versioned desired, for example what value has an old version of a Breaking News page on a major news outlets website got? This is probably the single biggest challenge in link integrity for hypertext and is not easily solvable without substantial research effort.

A.3.2 Semantic Web Research

The Semantic Web is based entirely on the notion of interlinked data which introduces several new integrity problems as discussed in sections 1.3 and 1.4. Therefore it becomes far more vital that authors take care in choosing their URIs since a URI needs to be far more permanent than traditional URIs, Sauermann et al. (2007) describe the issues associated with URIs for the Semantic Web.

As mentioned earlier (see A.1.2.2 and A.2.4.1) it may be necessary to introduce some form of permanent naming scheme in order to ensure that URIs remain dereferencable. At the moment the PURL scheme is proving to be the system of choice for a limited number of authors on the Semantic Web as it allows the potential for data to move at some point in the future without the URL changing[2]. Continued use of services like PURL and the potential new naming schemes like URN are an active research area since the requirement for permanent URIs on the Semantic Web drives demand for them.

Another issue (see A.2.5) is that you often end up with multiple URIs for the same object/concept since organisations will use different URI design conventions in their systems and while finding it necessary to represent the same/similar data. While ideally on the Semantic Web you should have one URI for every concept this is unlikely to happen since people cannot agree on strict definitions for concepts or common names for common objects e.g the differences between UK and US English. In fact it may well be desirable to have multiple URIs for things since each organisation 'minting' URIs for concepts will want to control that URI and the associated data. There have already been a number of papers written which begin to address this issue (Alani et al., 2002; Glaser et al., 2007) but there is much more to be done in this area. There is also continuing debate whether systems like the Entity Naming Service (Bouquet et al., 2008) from the OKKAM project (OKKAM, 2008) are useful to the Semantic Web.

In terms of the applicability of general link integrity to the Semantic Web existing techniques and tools such as link validation and JIT resolution have direct applicability to the Semantic Web. Obviously there is a need for more maturity in the technologies that will support this such as automated link validators for RDF and Semantic Search engines that JIT resolvers can use. With the essence of the Semantic Web being linked data it will be essential to ensure that the integrity of links is maintained in order to have useful and meaningful data. There is considerable work to be done in developing these tools for the Semantic Web in order to begin to address the issue.

[2]For example the Dublin Core Metadata Initiative use PURL for their URIs, see http://dublincore.org/documents/dcmi-terms/

Appendix B

dotNetRDF Overview

dotNetRDF is my own implementation of an RDF Library for the Microsoft .Net Framework built using C#.Net 3.0 on version 3.5 of the .Net Framework. It is designed to be a relatively lightweight but powerful API that makes it simple to get and manipulate RDF data. This Appendix gives a brief overview of the features currently supported in the library, the majority of which are used by All About That (see Section 2.1) in order to implement its functionality.

B.1 Namespaces

The dotNetRDF Library consists of 1 main namespace with 5 child namespaces. The base namespace (**VDS.RDF**) consists of the basic classes for representing RDF such as Graphs & Nodes as well as a variety of interfaces for these and related classes to allow for easy extension of the API. Child namespaces are as follows:

- **VDS.RDF.Parsing** - The Parsing Namespace contains the various parser classes which allow the library to read RDF data in all the standard formats: RDF/XML, Notation 3, Turtle and NTriples. It also contains specialised support classes and interfaces for these.
  - **VDS.RDF.Parsing.Events** - The Events Namespace consists of classes that support event based parsing.
  - **VDS.RDF.Parsing.Tokens** - The Tokens Namespace consists of classes that support token based parsing.
- **VDS.RDF.Query** - The Query Namespace contains the classes that form the current implementation of querying over graphs which is done programatically via implementation of a generic interface `ISelector<T>`. It also contains support for accessing remote SPARQL endpoints and retrieving Results Sets or Graphs

from those, partial incomplete support for SPARQL queries over local data is also provided.

  - **VDS.RDF.Query.Patterns** - The Patterns Namespace consist of classes that support Graph Pattern matching, this forms part of the unfinished SPARQL implementation.

- **VDS.RDF.Storage** - The Storage Namespaces contains the classes that support the use of database based storage of RDF Data. It permits for generic implementations of the `ISQLIOManager` class which allows arbitrary backing databases to be used with specialised Graph and Triple Store classes which use SQL backed storage. The library itself provides implementations for using stores in the dotNetRDF format.

- **VDS.RDF.Translation** - The Translation Namespaces contains classes that do direct translation between two RDF syntaxes.

- **VDS.RDF.Writing** - The Writing Namespace contains serializers which can serialize into any of the main RDF formats and into GraphViz Dot Format for creating graph visualizations.

B.2 Store Format

As can be seen in Figure B.1 the format of the dotNetRDF Store is relatively self-explanatory and has been designed to minimise replication of data and disk space requirements. The store is currently Microsoft SQL Server based though as mentioned previously the library has been designed such that the Store can easily be based on another database format or alternative stores used. The library also contains support for using MySQL as the database with SQL creation scripts for both Microsoft SQL Server and MySQL available.

B.3 Performance

As dotNetRDF is still quite an immature library its performance varies depending on the features being used; the following are rough performance figures. The library has not yet been used on any of the standard benchmarks as they mostly rely on using SPARQL over the data which is not yet supported. Figures given in Table B.1 are all approximate averages and vary depending on complexity of the syntax used.

For the dotNetRDF Store the figures are calculated by loading the same store (approximately 390,000 triples from the AAT test store described in Section 2.1.5 at the time of the benchmarking) 30 times and averaging the read speed over all the load operations.

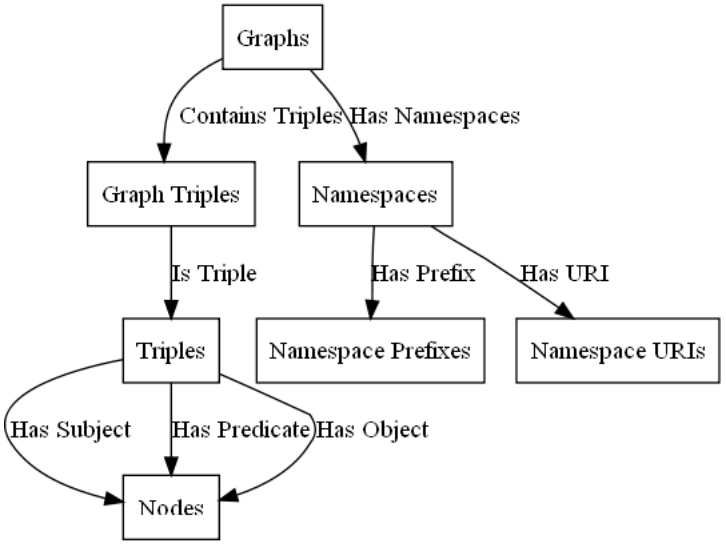


FIGURE B.1: dotNetRDF Store Format

A number of the tests on the dotNetRDF Store make use of advanced features of the API to improve the speeds.

For the standard serializers the figures are calculated by reading & writing a graph that contains 10,000 distinct triples to and from disk 30 times and averaging the speed. This graph was chosen since it is an atypical graph that doesn't lend itself to the syntax compressions that the serializers for the more expressive syntaxes support and is intended to show worse case performance. All serializers are run using their default settings with no pretty printing (RDF/XML Serializer automatically indents the output). High Speed Write Modes are a feature of some serializers that engage if they determine that the graph is not well-suited to syntax compressions which take more time to compute and output.

Data Format	Read	Write	Threads
NTriples	346	279,850	
NTriples (Using Turtle Parser in NTriples Mode)	337	N/A	
Turtle	370	210	
Turtle (High Speed Write Mode)	N/A	344,036	
Notation 3	365	129	
Notation 3 (High Speed Write Mode)	N/A	155,359	
RDF/XML	281	256	
dotNetRDF Store	8,789	65	
dotNetRDF Store (Manager Reuse)	11,473	N/A	
dotNetRDF Store (Multi-threaded)	17,445	138	8
dotNetRDF Store (Multi-threaded)	16,806	160	16
dotNetRDF Store (Multi-threaded)	15,853	161	32
dotNetRDF Store (Multi-threaded with Manager Reuse)	30,565	N/A	8
dotNetRDF Store (Multi-threaded with Manager Reuse)	30,005	N/A	16
dotNetRDF Store (Multi-threaded with Manager Reuse)	26,470	N/A	32

TABLE B.1: dotNetRDF Performance in Triples/second

As can be seen using too many threads can actually degrade performance in some cases due to the need for extra concurrency checks and thread management required.

B.4 Downloading

Currently the Library is still pre-Alpha and is not yet available publicly. It is being developed as a SourceForge project under the GNU GPL and will be available for download from there and from it's website http://www.dotnetrdf.org once it reaches sufficient maturity for an Alpha release to be made.

Appendix C

All About That Schema

The following is the formal RDFS format schema for All About That given in Turtle syntax. This schema is published as Turtle and as RDF/XML at http://www.dotnetrdf.org/AllAboutThat

```
# Turtle serialization of RDF Scheme for AllAboutThat

@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix aat: <http://www.dotnetrdf.org/AllAboutThat/>.

aat:firstAsserted rdf:type rdf:Property;
rdfs:range xsd:dateTime;
rdfs:domain rdf:Statement;
rdfs:label "Date First Asserted".

aat:lastAsserted rdf:type rdf:Property;
rdfs:range xsd:dateTime;
rdfs:domain rdf:Statement;
rdfs:label "Date Last Asserted".

aat:retractedOrDeleted rdf:type rdf:Property;
rdfs:range xsd:boolean;
rdfs:domain rdf:Statement;
rdfs:label "Retracted/Deleted?".

aat:changed rdf:type rdf:Property;
rdfs:range xsd:boolean;
rdfs:domain rdf:Statement;
rdfs:label "Object Changed?".

aat:source rdf:type rdf:Property;
rdfs:range xsd:anyURI;
rdfs:domain rdf:Statement;
rdfs:label "Source URI".

aat:Profile rdfs:type rdfs:Class;
rdfs:label "Class of All About That Profiles".

aat:created rdf:type rdf:Property;
```

```
rdfs:range xsd:dateTime;
rdfs:domain aat:Profile;
rdfs:label "Date Profile Created".

aat:updated rdf:type rdf:Property;
rdfs:range xsd:dateTime;
rdfs:domain aat:Profile;
rdfs:label "Date Profile Updated".

aat:name rdf:type rdf:Property;
rdfs:range xsd:string;
rdfs:domain aat:Profile;
rdfs:label "Friendly Name of Profile".

aat:profileSource rdf:type rdf:Property;
rdfs:range xsd:anyURI;
rdfs:domain aat:Profile;
rdfs:label "Source URI of Profile".

aat:localCopy rdf:type rdf:Property;
rdfs:range xsd:string;
rdfs:domain aat:Profile;
rdfs:label "Local Copy Filename/Identifier for Profile".

aat:maxTripleID rdf:type rdf:Property;
rdfs:range xsd:integer;
rdfs:domain aat:Profile;
rdfs:label "Maximum ID used for Triples in the Profile".

aat:allowsMultipleValues rdf:type rdf:Property;
rdfs:range xsd:boolean;
rdfs:domain rdf:predicate;
rdfs:label "Predicate Allows Multiple Values?".
```

Bibliography

Alberto Aimar, James Casey, Nikos Drakos, Ian Hannell, Arash Khodabandeh, Paolo Palazzi, Bertrand Rousseau, and Mario Ruggier. Weblinker, a tool for managing www cross-references. *Comput. Netw. ISDN Syst.*, 28(1-2):99–107, 1995.

Harith Alani, Srinandan Dasmahapatra, Nicholas Gibbins, Hugh Glaser, Steve Harris, Yannis Kalfoglou, Kieron O'Hara, and Nigel Shadbolt. Managing reference: Ensuring referential integrity of ontologies for the semantic web. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW'02)*, pages 317–334. Lecture Notes in Artificial Intelligence, 2002.

Helen Ashman. Electronic document addressing: dealing with change. *ACM Comput. Surv.*, 32(3):201–212, 2000. ISSN 0360-0300.

Amit Bagga. Evaluation of coreferences and coreference resolution systems. In *Proceedings of the First Language Resource and Evaluation Conference*, pages 563–566, 1998.

Tim Berners-Lee. Cool uris don't change, 1998. http://www.w3.org/Provider/Style/URI.

Tim Berners-Lee, Robert Cailliau, Jean-Francois Groff, and Bernd Pollermann. World-wide web: The information universe. *Electronic Networking: Research, Applications and Policy*, 1(2):74–82, 1992. citeseer.ist.psu.edu/berners-lee92worldwide.html.

Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, May 2001. http://www.sciam.com/article.cfm?id=the-semantic-web.

Chris Bizer. Equivalence mining and matching frameworks, 2007. http://esw.w3.org/topic/TaskForces/CommunityProjects/LinkingOpenData/EquivalenceMining.

P. Bouquet, H. Stoermer, and B. Bazzanella. An entity name system (ens) for the semantic web. *Lecture Notes in Computer Science*, 5021:258, 2008.

V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

David Bustard, Adrian Moore, Declan Higgins, and David Ayre. Towards self-managing web sites: The link integrity problem. *Engineering of Autonomic and Autonomous Systems, IEEE International Workshop on*, 0:61–67, 2007.

Michael L. Creech. Author-oriented link management. *Comput. Netw. ISDN Syst.*, 28 (7-11):1015–1025, 1996. ISSN 0169-7552.

P. Danzig, D. Delucia, and K. Obraczka. Massively replicating services in wide-area internetworks. *Univ. Southern California, Los Angeles, CA, Tech. Rep*, 1994.

HC Davis, JA Pickering, W Hall, and RJ Wilkins. Microcosm: An open hypermedia system. In *Proceedings of InterCHI'93 Conference*, ACM SIGGRAPH video series, page 526. ACM Press, 1993. Refereed video.

Hugh Davis. *Data Integrity Problems in an Open Hypermedia Link Service*. PhD thesis, University of Southampton, November 1995a. http://eprints.ecs.soton.ac.uk/6597/.

Hugh Davis. To embed or not to embed.... *Commun. ACM*, 38(8):108–109, 1995b. ISSN 0001-0782.

Hugh C. Davis. Referential integrity of links in open hypermedia systems. In *HYPERTEXT '98: Proceedings of the ninth ACM conference on Hypertext and hypermedia : links, objects, time and space—structure in hypermedia systems*, pages 207–216, New York, NY, USA, 1998. ACM. ISBN 0-89791-972-6.

Hugh C. Davis. Hypertext link integrity. *ACM Comput. Surv.*, page 28, 1999. ISSN 0360-0300.

Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. Microcosm: an open model for hypermedia with dynamic linking. In *Hypertext: concepts, systems and applications*, pages 298–311, New York, NY, USA, 1992. Cambridge University Press. ISBN 0-521-40517-3.

Hugh Glaser, Tim Lewy, Ian Millard, and Ben Dowling. On coreference and the semantic web. In *5th Annual European Semantic Web Conference (ESWC 2008)*, December 2007.

Hugh Glaser and Ian Millard. Rkbexplorer.com: Anatomy of a semantic web application. In *KISTI Workshop*, December 2008.

Harry Halpin. Social meaning on the web: From wittgenstein to search engines. In *WebSci'09: Society On-Line*, 2009.

Terry L. Harrison and Michael L. Nelson. Just-in-time recovery of missing web pages. In *HYPERTEXT '06: Proceedings of the seventeenth conference on Hypertext and hypermedia*, pages 145–156, New York, NY, USA, 2006. ACM. ISBN 1-59593-417-0.

David Ingham, Steve Caughey, and Mark Little. Fixing the "broken-link" problem: the w3objects approach. *Comput. Netw. ISDN Syst.*, 28(7-11):1255–1268, 1996. ISSN 0169-7552.

Afraz Jaffri, Hugh Glaser, and Ian Millard. Uri identity management for semantic web data integration and linkage. In *3rd International Workshop On Scalable Semantic Web Knowledge Base Systems*. Springer, 2007.

Afraz Jaffri, Hugh Glaser, and Ian Millard. Managing uri synonymity to enable consistent reference on the semantic web. In *IRSW2008 - Identity and Reference on the Semantic Web 2008*, 2008a.

Afraz Jaffri, Hugh Glaser, and Ian Millard. Uri disambiguation in the context of linked data. In *Linked Data on the Web (LDOW2008)*, April 2008b.

F. Kappe. A scalable architecture for maintaining referential integrity in distributed information systems. *Journal of Universal Computer Science*, 1(2):84–104, 1995. http://www.jucs.org/jucs_1_2/a_scalable_architecture_for.

F. Kappe, K. Andrews, J. Faschingbauer, M. Gaisbauer, M. Pichler, and J. Schipflinger. *Hyper-G: A new tool for distributed hypermedia*. Institutes for Information Processing Graz, 1994.

R. Moats. Urn syntax (rfc 2141), 1997. http://www.ietf.org/rfc/rfc2141.txt.

Luc Moreau and Nicholas Gray. A Community of Agents Maintaining Links in the World Wide Web (Preliminary Report). In *The Third International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agents*, pages 221–235, London, UK, March 1998. http://www.ecs.soton.ac.uk/~lavm/papers/gcWWW.ps.gz.

A. Morishima, A. Nakamizo, T. Iida, S. Sugimoto, and H. Kitagawa. : A tool for the automatic correction of broken web links. *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1486–1488, April 2008.

Atsuyuki Morishima, Akiyoshi Nakamizo, Toshinari Iida, Shigeo Sugimoto, and Hiroyuki Kitagawa. Why are moved web pages difficult to find?: the wish approach. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 1117–1118, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-487-4.

OCLC. Persistent url home page, 1995. http://purl.org.

OKKAM. Okkam project homepage, 2008. http://www.okkam.org/.

Thomas A. Phelps and Robert Wilensky. Robust hyperlinks: Cheap, everywhere, now. In *Digital Documents: Systems and Principles*, pages 514–549. Springer, 2004.

Leo Sauermann, Richard Cyganiak, and Max Vlkel. Cool uris for the semantic web, 2007. http://www.dfki.uni-kl.de/~sauermann/2006/11/cooluris/.

Diomidis Spinellis. The decay and failures of web references. *Commun. ACM*, 46(1): 71–77, 2003. ISSN 0001-0782.

A.J. Vanzyl, B. Cesnik, I. Heath, and H. Davis. Open Hypertext Systems, An Examination of Requirements, and Analysis of Implementation Strategies, comparing Microcosm, HyperTED, and the World Wide Web. *Internet Article,' Online*, pages 1–13, 1994.

L. Veiga and P. Ferreira. Turning the web into an effective knowledge repository. *ICEIS 2004: Software Agents and Internet Computing*, 14(17), 2004.

Luís Veiga and Paulo Ferreira. Repweb: replicated web with referential integrity. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 1206–1211, New York, NY, USA, 2003. ACM. ISBN 1-58113-624-2.