UNIVERSITY OF SOUTHAMPTON

Link Integrity for the Semantic Web

by

Robert Vesse

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Physical and Applied Science
Electronics and Computer Science

December 2012

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF PHYSICAL AND APPLIED SCIENCE
ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Robert Vesse

The usefulness and usability of data on the Semantic Web is ultimately reliant on the ability of clients to retrieve Resource Description Framework (RDF) data from the Web. When RDF data is unavailable clients reliant on that data may either fail to function entirely or behave incorrectly. As a result there is a need to investigate and develop techniques that aim to ensure that some data is still retrievable, even in the event that the primary source of the data is unavailable. Since this problem is essentially the classic link integrity problem from hypermedia and the Web we look at the range of techniques that have been suggested by past research and attempt to adapt these to the Semantic Web.

Having studied past research we identified two potentially promising strategies for solving the problem: 1) Replication and Preservation; and 2) Recovery. Using techniques developed to implement these strategies for hypermedia and the Web as a starting point we designed our own implementations which adapted these appropriately for the Semantic Web. We describe the design, implementation and evaluation of our adaptations before going on to discuss the implications of the usage of such techniques. In this research we show that such approaches can be used to successfully apply link integrity to the Semantic Web for a variety of datasets on the Semantic Web but that further research is needed before such solutions can be widely deployed.

Contents

List of Figures

List of Tables

Listings

Glossary

`Accept` Header The `Accept` Header is a Hypertext Transfer Protocol (HTTP) Header used to indicate the formats of content that the client making the request would like to receive and is given as a list of comma separated Multipurpose Internal Mail Extension (MIME) types. Wildcards can be used in the header to indicate ranges of MIME types e.g. `text/*` for all textual formats or `*/*` for any format. xviii, 11

AAT All About That. 63–68, 71–73, 75, 80, 82–84, 128, 129, 137–139, 143–145

AKT Advanced Knowledge Technologies was a research project conducted in collaboration with several UK universities. One of the results of this research was a variety of Resource Description Framework (RDF) vocabularies used in other projects such as the resilient knowledge base. 96

API Application Programmers Interface. 26, 47, 63, 65, 66, 71, 72, 91, 92, 140–143, 151

ARPANET ARPANET was the worlds first operational packet switching network initially created to link US universities together for research purposes. It was first deployed in the late 1960s and went on to become the backbone of the network that would evolve into the modern internet. 21

BBC British Broadcasting Corporation. 76, 77, 93, 95, 97, 98, 102–104, 108, 109, 120, 123, 142, 149, 163

CLT/WW change log table/web walk. 49, 55

CMS content management system. 50, 134

content negotiation Content Negotiation is a feature of HTTP whereby a client making a request to server states the data formats they understand using a particular header. This will typically be an `Accept` Header header but there are other headers that can be used to negotiate based on other factors such as client language. Properly configured servers should attempt to return the response to a request in a format the client understands if at all possible. 51, 54, 73

CRS coreference resolution service. 57–59, 90

delta A Delta is the description of some change(s) in a document. By applying a series of deltas a version control system can provide a user with any recorded version of a document. xxi, 63, 65, 78

dereferenced Deferencing is the act of the retrieving the data behind a reference, a reference whose data is retrieved is said to have been **dereferenced**. In terms of Uniform Resource Identifiers (URIs) this means issuing a HTTP request (or whatever form of request is appropriate for the URI) in order to retrieve the data. 37, 68

DOI Digital Object Identifier. 43, 44

DOS Denial of Service (DOS) is a type of attack that can be made against any service provided by a server though most commonly discussed in the context of a web server. If you make very large volumes of requests to a server with minimal delay between them a server may be unable to cope with the volume of requests and fail to fulfill some or all of them resulting in denying service to clients accessing that server. While this attack is typically malicious it can often occur unintentionally due to poorly written web crawlers. 81, 138, 139, 146

ECS Electronics and Computer Science is one of the schools that makes up the University of Southampton. 96, 97, 102, 120, 124

FOAF Friend of a Friend. 14

FTP File Transfer Protocol. 10

GPL The General Public License is a permissive open source license created by the Free Software Foundation. It requires that the source code be publicly available but allows users freedom to modify the code however they see fit provided that they release their code under the same license. 36

HTF Hyper-G Text Format. 26, 31, 32

HTML Hypertext Markup Language. 31, 36, 38, 72

HTTP Hypertext Transfer Protocol. xvii, xviii, xx, 8–10, 13, 37, 39, 46, 51, 52, 54, 65, 66, 71, 72, 91, 137

IETF The IETF is a standards organisation which helps define and maintain standards for the internet. These are typically published through their Request For Comments (RFC) process. xx, 37

IO IO is an abbreviation for Input/Output and refers to to the reading and writing of data. 81, 84

IP Internet Protocol. 139, 146

ISO International Standards Organisation. 37

JIT Just-in-Time. 46, 47, 59, 85, 108, 123, 135, 141

linkbase Linkbase is a term originating from the open hypermedia movement used to denote a collection of links within a system. Depending on the system there may be multiple distinct linkbases for different users, purposes etc. The actual implementation detail of a linkbase is system dependent. 32

MIME Multipurpose Internal Mail Extension. xvii

NLS oN-Line System. 20, 21

NQuads NQuads is an extension to NTriples which serializers a quad per line rather than a triple per line. This allows it to encode several RDF graphs in one file as opposed to NTriples which can only serialize a single graph per file. 93

NTriples NTriples is a plain text based serialization of a RDF graph. It serializes a single triple per line and has no syntax compression so is a very verbose serialization. xix, xxi, 92, 93

OKF Open Knowledge Foundation. 10

OWL Web Ontology Language. 7

PURL Persistent Uniform Resource Locator. 43

QName Qualified Name. 13

RDBMS Relational Database Management System. 75, 83, 138

RDF Resource Description Framework. iii, xvii, xix–xxi, 4, 5, 12, 14, 17, 29, 30, 52, 59, 64–73, 77, 86, 87, 89, 91, 92, 113, 114, 117, 132, 138, 140, 142–144

RDF serialization As RDF is an abstract data model it does not define a single data format for serializing a RDF graph. There are a variety of common formats that can be used for this including the official Extensible Markup Language (XML) based format (RDF/XML) and various plain text formats like Turtle. 52, 73, 93

RDF/XML RDF/XML is a XML based serialization of a RDF graph. It is one of the few standard serializations for RDF that is officially recognised by the World Wide Web Consortium (W3C). xx

RDFS RDF Schema. 163, 165

RFC An RFC is a memorandum document published through the Internet Engineering Task Force (IETF) which defines a protocol, process or other standard related to the internet. While RFCs are not formal standards in the same way as those created by a committee process through a standards body like the W3C they often become de facto standards or best practise over time. xix, 37, 43

RKB The resilient knowledge base (RKB) was a product of the ReSIST project developed by researchers at the University of Southampton and provides a selection of Semantic Web datasets covering primarily academic and bibliographic data. xvii, 96, 102

SGML Standard Generalized Markup Language. 32

SPARQL SPARQL is a recursive acronym standing for SPARQL Protocol and RDF Query Language. It is comprised of a pattern matching based language for querying RDF data and a protocol for defining how a query service is exposed via HTTP. 89–91, 95, 101, 104, 107–110, 113, 117, 142, 143, 147, 148, 163, 165, 168, 170

Turtle Turtle is a plain text based serialization of a RDF graph. The name comes from the typical human pronunication of its acronym TTL which stands for Terse RDF Triples Language. The terse part refers to the fact that Turtle provides various syntax constructs which allow data to be significantly compressed when compared to more verbose plain text serializations like NTriples. xx

URI Uniform Resource Identifier. xviii, 2, 3, 6–10, 12–14, 24, 30, 37, 41, 50, 52, 53, 57–59, 64–66, 68–73, 75, 77, 80–83, 86, 87, 89–93, 95, 96, 98, 101, 102, 105, 106, 113, 116, 117, 121, 124, 125, 129, 135, 136, 140–142, 146–148, 150

URL Uniform Resource Locator. 9, 10, 37, 41–44, 46–48

URN Uniform Resource Name. 41–43

version control system A Version control system is a piece of software that maintains the version history of documents allowing changes to be browsed and attributed to specific contributors. Typically such systems store a base version of the document and then store deltas for each change. xviii, 54, 63, 65

VoID Vocabulary of Interlinked Datasets. 90, 148

W3C World Wide Web Consortium. xx, 12, 38, 132

WAIS Wide area information server. 40

WWW World Wide Web. 30, 31, 33, 35, 38, 39, 45, 49, 53–55, 64, 134

XML Extensible Markup Language. xx

Chapter 1

Introduction

Hypermedia is a technology that is generally stated as having 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). This is not to say that the concept of organising human knowledge into a browsable form was new, after all libraries have existed since we invented writing, but he was one of the first people to describe how such a system might work in computerised form. In his article he presented an idea for a device called the Memex which would allow a person to browse large collections of information, link information items together and add annotations to the items. From this article the notion of hypermedia 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. The term hypermedia itself was coined by one of the later pioneers of the technology Ted Nelson who first published the term (Nelson, 1965).

Since hypermedia is fundamentally concerned with the interlinking of knowledge there is an obvious problem 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 distributed hypermedia 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 (Davis, 1999).

A limitation of hypermedia is that most systems were designed with a document-centric interaction and navigation model as seen with the Web and in many of the early systems described in Conklin's survey (Conklin, 1987). As a result content is very much aimed at humans and information discovery often relies on users searching for topics they are interested in. Halasz, who was one of the developers behind the NoteCards system (Halasz et al., 1987) listed search and query as one of his seven issues for the next generation of hypermedia (Halasz, 1988).

The Semantic Web is an extension to the existing document web inspired by ideas articulated by Tim Berners-Lee (Berners-Lee, 1998b), that aims to augment the existing Web with machine-readable data. The value of this is that it allows machines to retrieve and process structured data from the Web without relying on complex and often inaccurate data extraction techniques such as natural language processing.

As a result of the interlinked nature of the Semantic Web it becomes more important than ever to be able to maintain and preserve links, and to be able to recover useful data in the event of failure. Due to this interlinked nature a couple of additional link integrity problems are introduced that must also be considered. Since a Uniform Resource Identifier (URI) can be minted (see Definition 1.1) by anyone and used to refer to any concept they want, how do you determine what the meaning of a URI is? Additionally it is possible to say whether the concept identified by some URI A is the same as the concept identified by some URI B? If you can say this then what does it mean for applications? These two problems are known respectively as *1*) URI identity & meaning (Halpin, 2009) and *2*) coreference.

Definition 1.1. Minting a URI is the act of establishing the association between the URI and the resource it denotes. A URI *MUST* only be minted by the URI's owner or delegate. Minting a URI from someone else's URI space is known as URI squatting (Booth, 2009).

1.1 Research Hypotheses

Our hypothesis in this work is based around our assertion that link integrity for the Semantic Web is already an issue as we shall demonstrate in the subsequent chapter in Section 2.2. Given that the Semantic Web is built upon the same infrastructure as the traditional hypermedia web it should be possible to adapt existing techniques for maintaining link integrity developed for use on the Semantic Web. Therefore our hypothesis has two parts which are as follows:

1. Existing approaches from hypermedia research can be adapted to the Semantic Web and shown to provide some level of link integrity.
2. That approaches can be demonstrated to be sufficiently viable and scalable that they could be deployed in real production scenario.

In terms of viability we simply mean that a system must provide effective link integrity with minimal user involvement. The effectiveness of the link integrity provided should be provable, note that the methodology for this will vary depending on the techniques used. Minimal user involvement is somewhat harder to quantify, we consider user involvement to be minimal if once configured a system requires no user intervention, and if a user can use the link integrity functions within a couple of clicks.

With respect to scalability, it is important that any techniques developed should be accessible to as many users as possible. Therefore to be considered scalable we require that a system must be runnable on standard consumer grade desktop hardware and not adversely impact other day to day functions of the hardware. Additionally the system must demonstrate the ability or potential to scale to large datasets, ideally using a single machine. For our purposes we define large to be anything above 10,000 URIs. Note that we use the URI rather than triple as our measurement of size for a dataset because we are interested in links, each URI in a dataset may potentially be the target of a link. Triple counts are not relevant to us since we are concerned with whether we can provide link integrity for large numbers of links using everyday hardware.

1.2 Contributions

The contribution of this research is primarily a number of working proof of concept systems which provide link integrity to the Semantic Web. As well as presenting detailed descriptions of the design and evaluation of these systems we pay particular attention to discussing the pros and cons of each technique and the implications of their usage.

We appreciate that our solutions are not in any sense perfect and that realistically there is no perfect solution to the problem. Links will always fail because the Web itself is not designed to be perfect, but when links do fail our research shows that there are effective ways to combat these failures.

1.3 Chapter Summary

Firstly in Chapter 2 we introduce in more detail the various link integrity problems and also provide the reader with a more detailed introduction to Resource Description Framework (RDF) and the Semantic Web. Then in Chapters 3 and 4 we discuss the evolution of hypermedia and the variety of existing approaches to link integrity problems covering both existing hypermedia research and more recent Semantic Web based research. We identify a couple of approaches from these which we feel are applicable to the Semantic Web and we describe our implementations of these in Chapters 5 and 6.

Chapter 8 discusses in more detail the results of evaluating the systems we presented in the preceding chapters and looks at whether we have succeeded in validating our hypothesis. One particular issue we raise in this chapter is the social and technological implications of using such systems. Finally in this chapter we discuss possible future avenues in this area of research.

Chapter 2

Background

In this chapter we introduce some of the key concepts that are referred to frequently throughout our research. We start by introducing the different link integrity problems in more detail before providing a quick guide to Resource Description Framework (RDF) in order that the reader gains a high level understanding of how the Semantic Web functions.

2.1 The Problems

As alluded to earlier the vision of the Semantic Web first espoused in Berners-Lee (1998b) is of a Web where clients - whether humans or software agents - can retrieve structured data about things of interest and use this data however they please. It can be said that the ability of these clients to perform their intended functions and fulfil this vision is limited only by their ability to retrieve the data from the Web. As the quantity of data on the Semantic Web continues to explode - as evidenced by Figures 2.4 and 2.5[1] which show the difference in size of the linked data cloud between October 2007 and September 2011 - the issue of data integrity will become increasingly important. We are still some way off fulfilling the vision of the Semantic Web and one of the missing pieces currently is fault tolerant clients. Our position is that clients must be able to work around failures on the Web since assuming 100% uptime of data on the Web is unrealistic.

[1]Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

As previously stated the problems we are attempting to address are the well known problems of hypermedia link integrity i.e. the problem of ensuring that links *1*) resolve and *2*) return the expected data (Davis, 1999). These two problems are generally known as the 'dangling-link' problem and the editing problem respectively. In addition to these we also have to consider two additional semantic web specific issues *1*) Uniform Resource Identifier (URI) identity & meaning and *2*) coreference.

2.1.1 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, therefore 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 or 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 the author was unaware of. Additional causes of 'dangling' links can include a change of server technologies meaning that file extensions in the URIs have changed or the change of domain or subdomain for a site. This is the type of problem which most research into link integrity has attempted to solve since it is simpler to address than the editing problem.

2.1.2 The Editing Problem

The other (arguably) more minor issue in hypermedia is the editing problem (or content reference problem) which occurs when a resource is modified such that links to the resource will work but the resource 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.

Also on the Semantic Web using a specific URI may have unintended consequences especially when reasoning or inference is applied to your data. For example if you had some data in which you talked about London (as in the capital of the UK) and referred to it using the DBPedia identifier for London[2] this would be a sensible reuse of that identifier. However, if you were to use your data combined with some other data sources you might find that someone else had stated in their data that what you meant as London should be considered the same as some other London (e.g. London, Minnesota, USA[3]) which suddenly changes the meaning of your data.

This is a fairly crude example of the problem and there are much more subtle issues that can arise particularly when you start applying reasoning over the data. For example the `owl:sameAs` relationship which is commonly used to say that two URIs represent the same concept actually has quite strict semantics if you apply full Web Ontology Language (OWL) reasoning. As has been argued in work such as Glaser et al. (2007) this is actually unwise and inappropriate in many cases since one bad assertion can quickly lead to many incorrect reasoning conclusions. Due to this problem the issues of trust and provenance with regards to data are open questions on the semantic web, this problem is outside the scope of my work but Golbeck provides an overview of this area in (Golbeck, 2006).

[2] `http://dbpedia.org/resource/London`
[3] `http://dbpedia.org/resource/London%2CMN`

2.1.3 URI Identity and Meaning

The Semantic Web introduces an issue of URI identity and meaning since URIs are no longer referring simply to documents or other Hypertext Transfer Protocol (HTTP) accessible resources, but to potentially anything. As a result there has been considerable debate in the semantic web community about what the identity and 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 or at the very least missing data.

There is some debate in this area around the now infamous HTTP range-14 issue (Fielding, 2005) and the difference between retrieving a resource and the description of a resource. Without a clearer idea of whether missing meaning or data matters it is hard to say what, if anything, we should be doing to prevent this situation arising. Most of these issues are well beyond the scope of this work but Halpin (2009) provides an excellent overview of some of the discussions in this area.

2.1.4 The Coreference Problem

Coreference 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 (Winkler, 1999). The basic issue is that because everything is referred to using URIs you will often 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 want to control their data as much as possible even if they do publish it semantically. This is a particular issue with businesses as they often need to take care what they say as a company in order to protect their brand image, and if the data is out of your control it is subject to manipulation in undesirable ways.

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 coreference problem (Glaser et al., 2007; Jaffri et al., 2007; Bouquet et al., 2007) looks at ways in which coreferent URIs can be determined and how this information can be conveyed to Semantic Web applications.

2.2 Prevalence of Broken Links

Over the years there have been a number of studies which have looked into the persistence of links i.e. how long links remain working after they are first published, which have shown that links fail at an impressive rate. Surveys such as Spinellis (2003) found that 28% of the URIs published in two scientific publications in the period 1995-1999 were no longer functioning in 2000. They repeated the experiment two years later to find that the proportion of broken links had increased to 41% and therefore they were able to caluclate that the half-life of a URI was approximately 4 years.

Koehler's study (Koehler, 2002) showed staggeringly high levels of Uniform Resource Locators (URLs) ceasing to function showing that only 34.4% were still functional after a 5 year period, though this may be exacerbated by the small sample size used. One particular problem raised by his study is what he termed *phantom* web pages. There are server generated error pages which indicate a 404 or similar error yet don't send an appropriate HTTP status code, thus making it appear to be a functional web page to software. In his study he was forced to use a manual checking procedure to determine whether pages were actual content page or these *phantom* error pages. Another useful point he makes is the notion of *intermittence* which is that resources on the Web may appear to be unavailable due to temporary or transient issues e.g. network errors and power outages and so links cannot always be immediately deemed broken. Given his figures he arrives at a half life of 2 years for links on the Web which is very short, again we must be mindful that his small sample size may have exaggerated the issue.

Another large scale study that demonstrates the problem more realistically is Lawrence et al. (2001) which looked at the links contained in scientific publications in a similar vein to the already discussed study (Spinellis, 2003). The main differences in the studies are scale, while Spinellis' study used only 4,224 URLs this study used 67,577 URLs. Their initial findings found that the percentage of invalid links ranged from 23% for

1999 to 53% for 1994 which is in line with the findings from other surveys. One notable aspect of their survey is that they took a random sample of 300 invalid URLs and attempted to relocate them manually. Of the sampled portion that were valid URLs (68%) i.e. not syntactically invalid or toy examples like `http://example.org` they successfully relocated 97% of the links. As we will show in Chapter 4 there are a variety of systems that use automated retrieval techniques that can do this kind of relocation with reasonable success levels which demonstrates that recovery style approaches are viable.

2.2.1 Broken Links on the Semantic Web

In a similar vein to these studies we carried out a simple experiment designed to demonstrate that faulty links on the Semantic Web are already prevalent and therefore a real issue that must be at least discussed if not addressed. To do this we conducted a crawl of CKAN[4], which is a website created by the Open Knowledge Foundation (OKF)[5] that styles itself as 'the Data Hub', and allows people to publish the details of data available on the Web. Using the CKAN API we carried out a crawl of all the datasets available to see whether the links to the actual data functioned correctly. We found that out of 1850 packages only 1389 provided links to the data and of these 265 reported an error. This means that already 14.3% (or 19.1% if you ignore datasets without a link to the data) of the links to the data are non-functional. Considering that CKAN only represents a small portion of the ever expanding Web of Data there is clearly already an issue which merits discussion and research.

A detailed breakdown of the types of HTTP responses encountered can be found in Table 2.1. Note that *Unknown Error* refers to cases where a non-HTTP error occurred, this includes errors such as:

- The download URI being malformed i.e. the URI itself is broken.
- The URI not being a HTTP URI, for example 25 packages had File Transfer Protocol (FTP) URIs instead.
- Fundamental connection issues e.g. the domain name being unresolvable.

[4]`http://ckan.net`
[5]`http://okfn.org`

HTTP Status Code	HEAD Request	GET Request
OK Responses		
200 OK	1120	1140
301 Temporary Redirect	0	0
302 Moved Permanently	0	1
303 See Also	0	0
Error Responses		
400 Bad Request	16	21
401 Unauthorized	8	7
403 Forbidden	8	5
404 Not Found	93	92
405 Method Not Allowed	20	0
406 Not Acceptable	0	5
410 Gone	1	1
500 Internal Server Error	51	34
501 Not Implemented	1	0
502 Bad Gateway	1	1
503 Service Unavailable	0	0
Unknown Error	65	71
Total Responses	1389	1389

Table 2.1: CKAN Crawl Statistics (May 2011)

As can be seen from the table the majority of errors can be accounted for by the download link dangling (404 Not Found or 410 Gone response) with a total of 94. The aforementioned unknown errors with a total of 65 or server errors (5xx responses) with a total of 53. Interestingly a small number of servers responded 406 Not Acceptable which a server uses to say it cannot return content in the desired format. This is despite us sending an `Accept` Header header of `*/*` which indicates that we are willing to receive any format.

This small experiment on just one such aggregation point for a small segment of the Semantic Web shows that broken links are already rife. Given past studies conducted for the Web such as those introduced in the preceding section, broken links have a tendency to increase over time. For example in Pitkow (1998), which is a survey of various aspects of Web related research, Pitkow cites studies which suggest the average lifetime of a document on the web is only 50 days and that 5-8% of links encountered during surfing are broken. Given that this simple experiment shows links to Semantic Web data have a 14.3% failure rate in our sample, it is reasonable to assume that such findings will also prove to be accurate on the Semantic Web.

2.3 The Resource Description Framework (RDF)

The standard model for data on the Semantic Web is the Resource Description Framework (RDF) which is a syntax independent abstract model specified by the World Wide Web Consortium (W3C) (Klyne and Carroll, 2004), which represents data in the form of graphs. Each relationship between two nodes in the graph is a triple formed of a subject, predicate and object where the subject and objects are nodes representing some resource or value, and the predicate represents the relationship between them. Each triple represents a statement of some fact about something, hence you will often see the term statement used interchangeably with triple. A simple example is given in Figure 2.1 which models that my age is 25. A RDF graph can be built up from a set of triples (an example of this is given in Figure 2.2) which adds the facts that the object Rob, has a surname of Vesse and lives in Southampton to the graph.

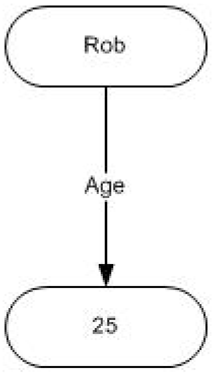


Figure 2.1: Example RDF Triple

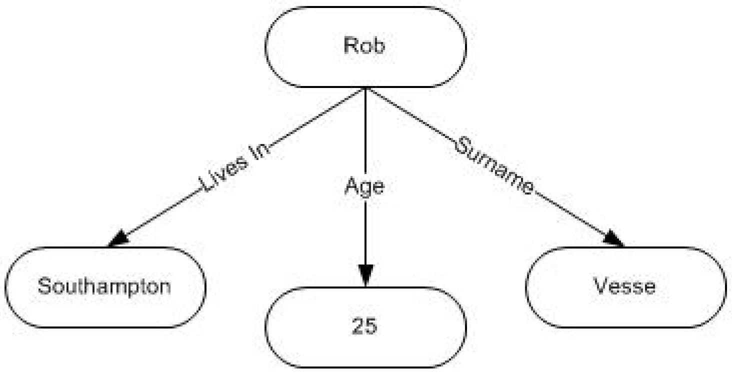


Figure 2.2: Example RDF Graph

This on its own does not give us a Semantic Web, what makes it a Web is that we name the resources and relationships - such as *Rob* and *Age* - by using Uniform Resource

Identifiers (URIs). This means that on the Semantic Web every triple can represent a link between two resources. Importantly it is not required that a URI be dereferenceable since:

1. There are many URI schemes other than Hypertext Transfer Protocol (HTTP) which do include a dereferencing mechanism e.g. tag URIs (Kindberg and Hawke, 2005).
2. They may be used purely as identifiers to name things i.e. the publisher does not intend them to be dereferenceable.

Despite this not everything needs to be a URI since otherwise you would have to create an infinite number of URIs to represent things such as numbers, instead these are represented by literals. Literals can just be a simple textual value or they may have either a language specifier or data type URI to attach additional meaning to the literal. The main limitation on the use of literals is that they may only be used as the object of a triple, in essence you can say that some resource has a value for some property but you cannot directly state properties about a literal itself.

Finally you may want to give something an anonymous identifier i.e. you want to name it but do not need to refer to the name outside of a specific graph, in which case you can use a blank node. A blank node is an anonymous identifier scoped to a specific graph which can be used as the subject or object of a triple. This is typically used to create intermediate resources which group sets of values together in the graph or where you do not need to create a URI.

By naming things with URIs and using literals to represent values we can get a graph like the one shown in Figure 2.3. In the graph the main resources (people and locations) and the relationships (name, age, location) are replaced with appropriate URIs. The things which it does not make sense to assign URIs to such as the value 25 for my age are represented with literals. For succinctness in the example relationship and data type URIs were expressed in Qualified Name (QName) form. This is a technique for abbreviating URIs, for example the QName `foaf:age` is shorthand for the URI `http://xmlns.com/foaf/0.1/age`.

Due to the fact that URIs are used to identify things it becomes possible to agree on common identifiers for well defined things such as places, people and relationships. In

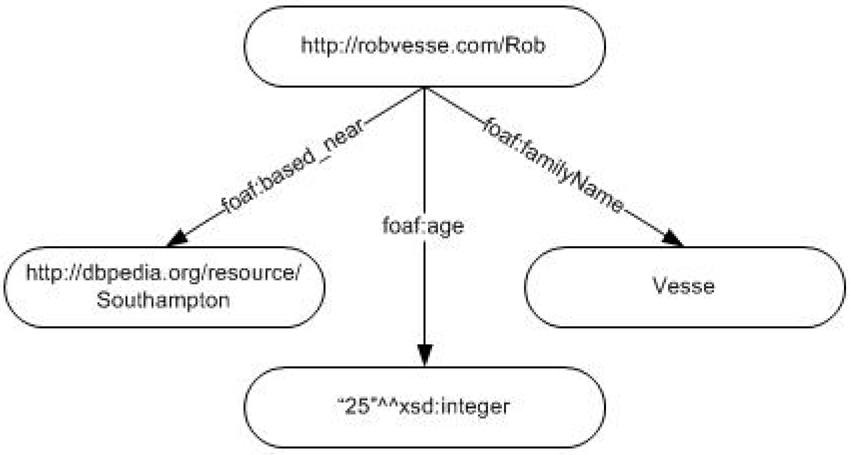


Figure 2.3: Example RDF Graph with URI

the preceding example we used the Friend of a Friend (FOAF) vocabulary which is a standard vocabulary for expressing properties of people and social relationships between them. Again an important point here is that just because the Semantic Web encourages URI reuse it does not mandate it. While you can reuse existing identifiers there are times when this is not appropriate or when you may prefer to create your own identifier instead.

One of the biggest segments of the Semantic Web currently has been created by the linked open data[6] project. This is a community movement aimed at bootstrapping the semantic web by getting large data sources out on the web in the form of RDF and making links between them. It started out by converting a number of large freely available data sources, such as Wikipedia[7], into RDF. Gradually many smaller datasets have grown up around these initial hubs as seen in Figures 2.4 and 2.5[8]. The linked data project is of particular interest since it provides a large amount of RDF data where the applications built upon it are heavily reliant on the interlinkings between different datasets. This provides a comprehensive selection of sources of real world data to test possible solutions against and is a domain where link integrity tools would most benefit end users.

[6] `http://linkeddata.org`
[7] `http://wikipedia.org`
[8] Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. `http://lod-cloud.net/`

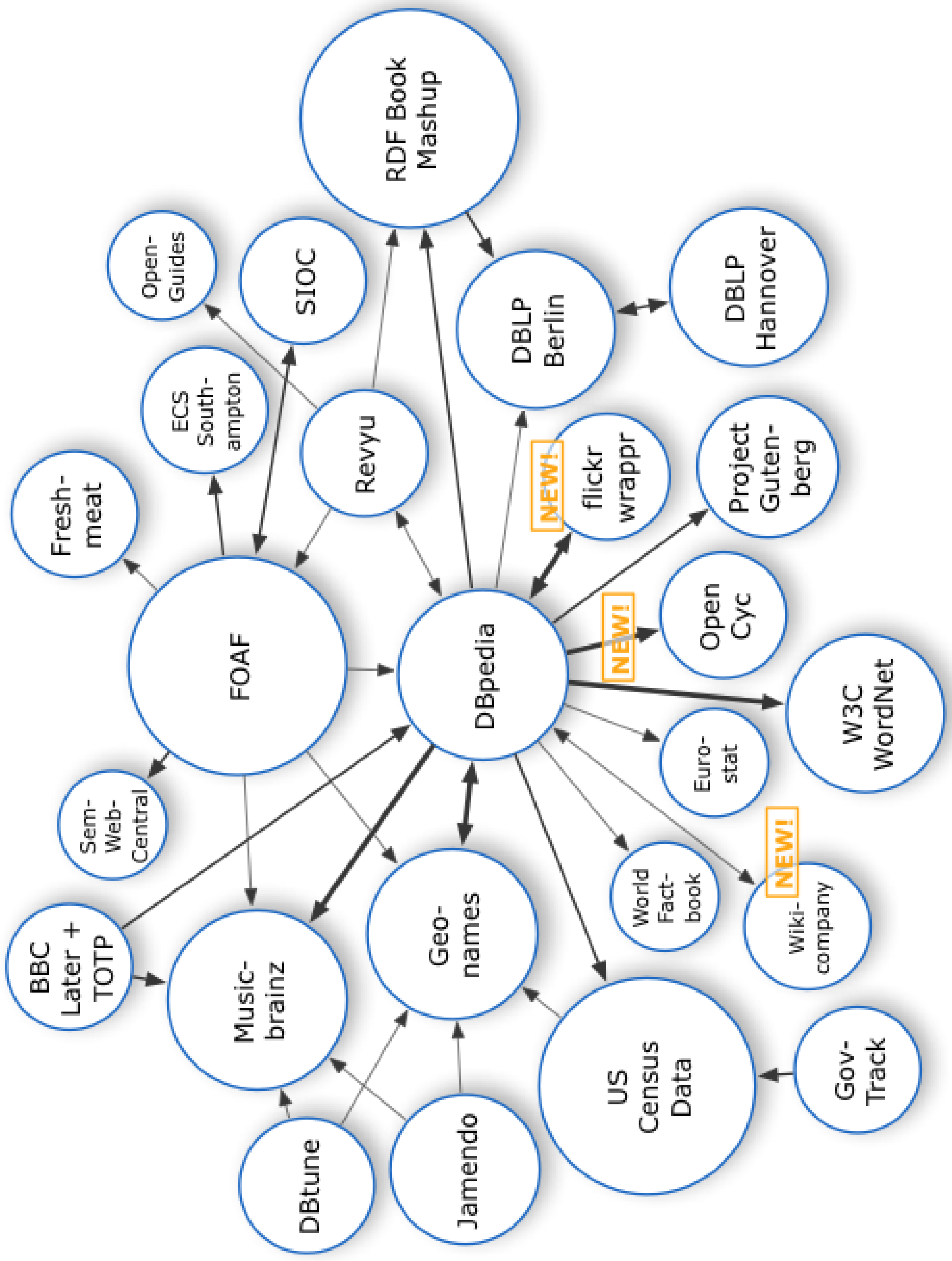


Figure 2.4: Linked Data Cloud October 2007

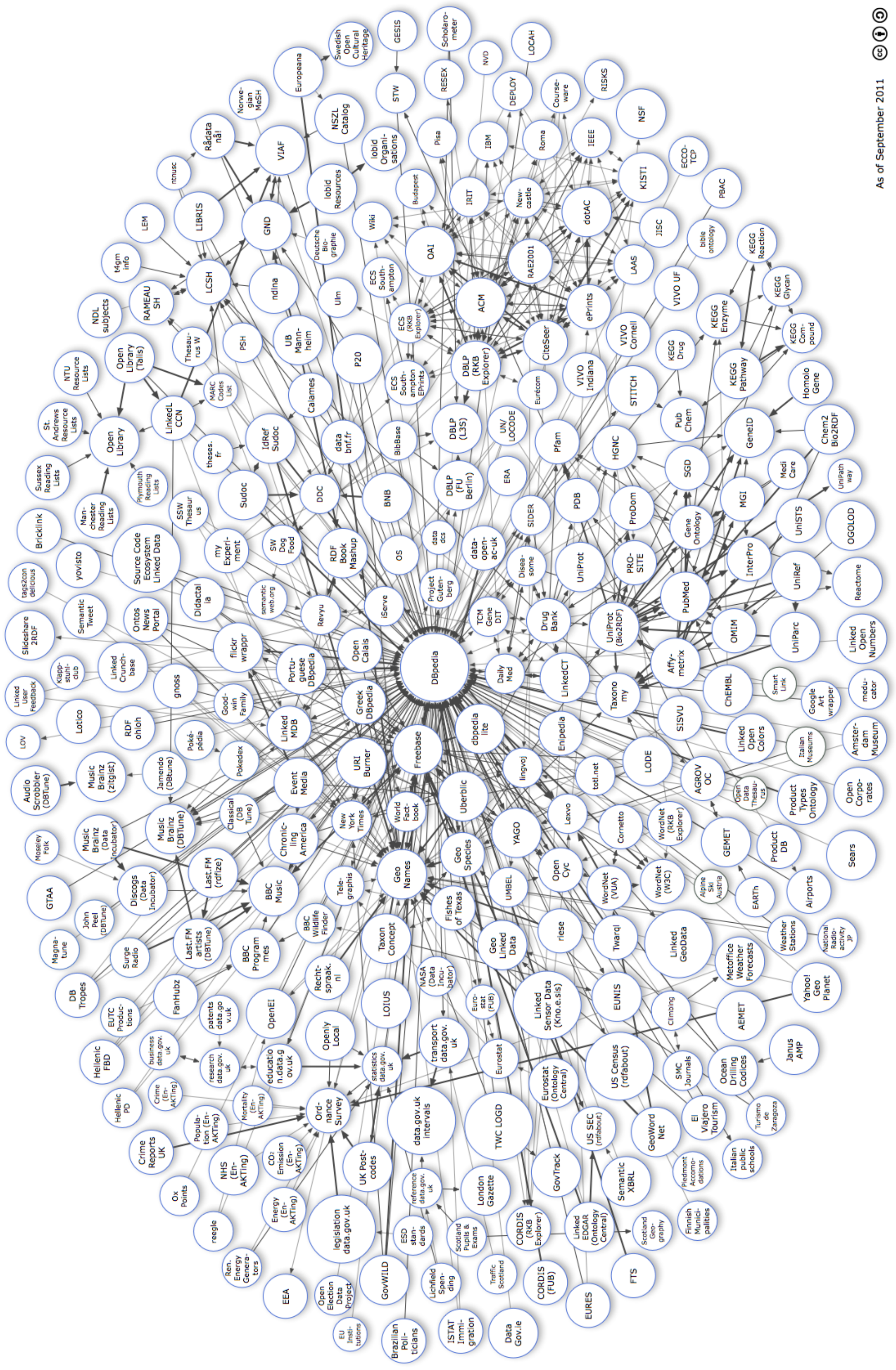


Figure 2.5: Linked Data Cloud September 2011

2.4 Summary

In this Chapter we have introduced the problem of link integrity in more detail and provided a demonstration of why it is already a problem on the Semantic Web. We have also introduced the RDF data model which is key to the Semantic Web and shown how it naturally makes almost every bit of data a link between resources.

In the next chapter we look at how the notion of hypermedia evolved from the early pioneers through to the open hypermedia systems of the later 1980s and early 1990s, and also look at how research into link integrity was fundamental to some of those systems.

Chapter 3

The Evolution of Hypermedia and Link Integrity

As mentioned in the introduction of this thesis the notion of hypermedia is generally attributed to Vannevar Bush who proposed it in his seminal article 'As we may think' (Bush, 1945). He proposed a device called the Memex which would act as an aid in the organisation and navigation of information. The key features that he envisaged in such a system were the ability to link together any two pieces of information and the ability to create what he termed 'trails' which allowed a user to navigate a specific sequence of information. This ability to link information together is the fundamental characteristic of hypermedia that has defined all subsequent implementations of hypermedia systems. The main focus of this chapter is to look at the evolution of the notions of links and link integrity through the generations of hypermedia that lead up to the Semantic Web itself.

In this chapter we discuss the history of hypermedia starting with the early pioneers who first implemented the ideas Bush espoused. We look briefly at their systems before discussing the open hypermedia movement whose research and development did much to advance the adoption of hypermedia. In particular they contributed much to the early development of solutions for link integrity. We then proceed to look at the emergence of the World Wide Web as the de facto globally distributed hypermedia system and the effects that it had on link integrity research. Finally we go on to explore the evolution of the Web into a Semantic Web and the existing efforts aimed at applying link integrity to it.

3.1 Early Hypermedia

The notion of hypermedia as proposed by Vannevar Bush (Bush, 1945) was well ahead of its time when first published in 1945. Though early computers had been developed around that time they were still very limited and certainly did not have the human friendly user interfaces that Bush envisaged the hypothetical Memex device to have. The first hypermedia systems to be developed are generally acknowledged to be those of Ted Nelson and Douglas Engelbart in the 1960s (Engelbart and English, 1968; Nelson, 1980). In fact it was Ted Nelson who actually coined the terms hypertext and hypermedia[1] and would go on to be revered as one of the founding fathers of hypermedia.

In this section we look at the first generation systems created by these early pioneers before going on to discuss the second generation systems which popularised hypermedia in the 1980s. We conclude with a discussion of Halasz's Seven Issues (Halasz, 1988) which was his critique of that generation of hypermedia systems.

3.1.1 NLS

The oN-Line System (NLS) was designed by Douglas Engelbart and implemented by researchers at Stanford during the 1960s and represents the first tangible hypermedia system. The system was designed to both link content together and to provide functions like video-conferencing. As well as being the first real hypermedia system it was also famous for being one of the most advanced computer systems of its time and was famously demonstrated by Engelbart at the Fall Joint Computer Conference in 1968 (Engelbart and English, 1968). This demo was so elaborate and impressive that it has become known as the 'Mother of all Demos'[2], as well as introducing hypertext and hypermedia it also marked the debut of the mouse, video-conferencing, teleconferencing, email and collaborative real-time editing. All of these have since become common place technologies but were all decades ahead of their time.

[1]The first published reference to hypertext is considered to be from the Vassar College newspaper Miscellany News (Wedeles, 1965)

[2]Available from Stanford University at `http://sloan.stanford.edu/MouseSite/1968Demo.html`

NLS is very important to the history of hypermedia because it was a practical, usable implementation of the concept with working links. As with Nelson's Xanadu, which we will discuss shortly, it laid the foundations for the many hypermedia systems that would come after it. It is also important in that it was the first properly distributed hypermedia system being deployed onto the fledgling Advanced Research Projects Agency Network (ARPANET) and proved that hypermedia could work in a distributed environment. This distributed nature is part of the power of links but also one of the reasons link integrity is such a difficult problem. Although there are many ways to provide link integrity, as we shall detail later in this chapter, they are often complicated by the distributed nature of the systems. Once you have distributed the system you often need the collaboration of multiple parties in order to provide link integrity, you will see from our subsequent discussions that this was one of the main reasons many proposed solutions were unworkable in reality.

3.1.2 Project Xanadu

Project Xanadu[3] was started in 1960 by Ted Nelson and has gone on to become one of the most well known and controversial hypermedia systems in existence. Though Nelson started work on it in the 1960s as a graduate student an actual release of the software did not take place until 1998 and even at that point it was incomplete. Nelson's 1980 book Literary Machines (Nelson, 1980) offers an extensive overview of his notion of hypertext and Project Xanadu and other related research.

The Xanadu system itself was designed to be an advanced hypermedia system whose features included the following:

1. **Non-sequential navigation:** As in Bush's vision for the Memex the system was designed to allow the user to browse a collection of information by navigating links in any sequence they desired.
2. **Bi-directional Links:** It supported true bi-directional links which could be followed from either end of the link.
3. **Versioning:** Proper versioning of content was supported, you could maintain

[3]http://xanadu.com/

multiple versions of content and use existing content as the basis for new content easily.

4. **Transclusion:** New content could be authored which transcluded all or part of existing content.

The definition of links is of particular relevance to us since you cannot define link integrity without some notion of what a link is. Nelson thought of links as being bi-directional since it was important to him that a person could follow them in either direction. For us the directionality of links has minimal bearing since regardless of the direction of the links it should be possible to ensure that links continue to work over time and thus provide link integrity. However, directionality will always have some bearing on how difficult maintaining link integrity is since if you have bi-directional links you have two points of failure as opposed to one with uni-directional links.

As part of Project Xanadu Nelson created a custom file format (Nelson, 1965) which was designed to facilitate non-sequential writing and tranclusion. He referred to this concept as zippered lists and they were supposed to provide for easy creation of compound documents using his notion of tranclusion. The problem with the project was that its complexity made it very difficult to actually implement and despite ongoing efforts by Nelson the system was never properly deployed. In the time that it has taken to partially implement Xanadu several generations of simpler hypermedia systems have been designed, implemented, deployed and superseded. Despite this we cannot discount the importance of Xanadu in influencing the hypermedia research community. The notions of bi-directional links, versioning and tranclusion were incorporated into many later hypermedia systems.

Despite the fact that most later hypermedia systems use the basic ideas introduced by Nelson, Engelbart and other early pioneers, Nelson has often been quite critical of these systems. His view is that later systems like the World Wide Web did not fulfil his visions of hypermedia since they provided a system which 'trivializes our original hypertext model with one-way ever-breaking links and no management of version or contents'[4]. In some senses he is correct in this view since the one-way nature of links on the Web and the Semantic Web is the root cause of the link integrity problem we have attempted to address in our research.

[4]Quotation taken from the Project Xanadu website at http://xanadu.com

3.1.3 Mainstream Hypermedia in the 1980s

The first hypertext and hypermedia systems which were widely used began to appear in the late 1980s and were initially designed to function from a single machine e.g. NoteCards (Halasz et al., 1987), Neptune (Delisle and Schwartz, 1986) and Intermedia (Garrett et al., 1986) and are generally referred to in the literature as second generation systems. These systems generally focused on using proprietary file formats and viewer applications which allowed users to author and link together related information.

For example NoteCards focused on creating ‘Cards’ as you would with writing traditional paper note cards. These ‘Cards’ could be organised into ‘Fileboxes’ as you would with paper cards but with the advantage that each ‘Card’ could be linked to any number of other ‘Cards’. Neptune is similar is style to NoteCards but was designed primarily for use with CAD applications to link together different aspects of the design. Intermedia and Neptune were more advanced than NoteCards in that they supported distributed working and therefore allowed users across a network to work on a common set of documents.

The key advantages of NoteCards and similar systems is that being non-distributed the link integrity problem was to a certain degree much simpler to address. While it was still possible for files to be deleted or moved on a single machine this can be detected with minimal effort i.e. checking for existence when attempting to access a file and then searching to relocate it (provided it’s not been deleted). This means the ‘dangling-link’ problem is of minimal concern to such systems. More problematic still is the Editing problem since these systems cannot guarantee that the information at the end of a link is the correct information with regards to the source of the link and its intended purpose. As will be discussed later distributed systems such as Intermedia and Neptune are far more problematic and require much more complex systems to enforce link integrity.

3.1.3.1 Halasz’s Seven Issues

NoteCards is of particular interest in the history of hypertext research since one of the creators Frank Halasz wrote a seminal critique of his own system in which he discussed seven issues for the next generation of hypertext (Halasz, 1988). The issues are of particular importance since later developments in hypertext research were often measured against these issues to gauge the state of the research and the maturity of the technology. Additionally several of these issues have a bearing on the problem of link integrity. The seven issues are as follows:

- *Search and Query*

  Most early hypertext systems only permitted the user to navigate either by looking at the organisational structure as a whole or by following links from the hypertext nodes. This proved to be insufficient for users since they could easily miss relevant information because it wasn't necessarily linked to related information correctly. The ability for users to simply query the system for information on a given topic didn't exist in systems of the time.

  Search does not provide a fix for link integrity directly but as we will see with research like Phelps and Wilensky (2004) and Harrison and Nelson (2006), which we will discuss in detail later, it can be used as part of the solution. Interestingly search may actually introduce more potential for broken links since Uniform Resource Identifiers (URIs) of searches are highly susceptible to the editing problem as search results will change over time unless a system is completely static.

- *Composition*

  Some early systems did not provide any mechanism for different forms of content to be composed into one object e.g. a document containing graphics and other multimedia. This issue can be considered to be problematic in terms of link integrity since composition is often achieved by dynamic content generation typically driven by query string parameters in links.

  Just as with search this is susceptible to the editing problem since content can change over time or parts of the content to be composed may become unavailable.

- *Virtual Structures*

  The ability to create dynamic structures at the time the user accesses a system was considered an important development. The modern Web is now composed primarily of content which is all or partially of this type and his presents its own issues for link integrity. This is particularly with regards to the editing problem, since if you link to a dynamically generated page there is not necessarily any guarantee that the content at the end of the link is what you expect it to be.

Virtual structures in modern systems are often analogous or closely linked with composition and so are susceptible to the same issues of parts of the content becoming unavailable.

- *Computation over Networks*

  Halasz believed it should be possible to perform computation over a hypermedia network in order to produce new information or modify existing information. Essentially this is the inference of implicit information from existing explicit information. Many modern systems now achieve this to some degree and it is one of the hot research topics in Semantic Web research and within the emerging discipline of Web Science.

- *Versioning*

  Some researchers believed that in order to preserve linking between objects it should ideally be the case that all versions of an object in a hypermedia system are preserved so that a link can always be resolved to its intended destination.

  Versioning provides one possible means of fixing link integrity since if everything is versioned you can always retrieve a past version of the data once it no longer exists. Research in this area such as Moreau and Gray (1998); Veiga and Ferreira (2003) has used this technique and will be discussed in more detail later in this chapter.

- *Collaborative Work*

  Since early systems were not distributed, they made any form of collaborative working difficult or impossible. While modern systems generally handle collaborative working well, it can pose issues for link integrity since two users can make changes to a system independently that individually have no effect on link integrity but when combined may break link integrity. A simple example of this is one user editing a page to link to another while the other edits the target page such that the link is no longer relevant, this is a classic example of the editing problem.

- *Extensibility and Tailorability*

  The ability to extend and customise a system as desired was not really an option with early hypertext since most systems were proprietary and few people were allowed to work with the actual source of the systems. NoteCards itself was unusual in having an Application Programmers Interface (API). Halasz believed that the very generic nature of hypertext systems posed a problem for them since they often were too broad to serve the purposes to which users wished to put them. Modern systems now provide these features as standard with comprehensive documentation in order that users can specialise their systems as desired. Extensibility can potentially cause problems for link integrity if it changes the way in which the system handles linking or if it introduces forms of content which the system cannot control sufficiently to ensure the integrity of links to that form of content.

You will recognize that several of these issues are directly related to the features that Nelson had envisaged in his vision of hypermedia. In particular the concepts of composition, virtual structures and versioning are direct descendants (if somewhat simplified) of the notions of versioning and tranclusion that he first introduced in Project Xanadu.

Note that many of these issues have to some degree been solved or mitigated by solutions developed for the Web e.g. Search and Query has been mostly solved by the rise of the search engines (see Section 4.1).

3.2 Open Hypermedia

Open hypermedia was a movement in the late 1980s and early 1990s around systems designed to link together arbitrary documents to create hypermedia systems. The prime examples of this are systems such as Microcosm (Fountain et al., 1990) and Hyper-G (Kappe et al., 1994), which were designed around the time when the World-Wide Web was not yet the dominant hypermedia system it is today. These systems were similar to the early hypermedia systems discussed in the previous section but they were no longer completely tied to proprietary formats. Although system specific formats were still used such as Hyper-G Text Format (HTF) in Hyper-G (Kappe, 1993), these were often standardised and open so that other systems could understand them.

Open hypermedia systems differed from earlier systems such as NoteCards which had a specific document format, because they were typically designed to link any kind of document that a user had on their computer together. Most early systems required the user to do all their content authoring and management within the system. In open hypermedia the user had the freedom to create content using whatever software they desired and then make links between documents later using the hypermedia systems standalone tools, plugins etc. as appropriate. While some were initially intended only to be run on single machines (e.g. Microcosm) most eventually evolved to support some form of distributed system in an effort to compete with the Web. The other key difference between these systems and earlier ones is that these typically stored the links entirely externally to the documents. It was this design decision that allowed them to link together arbitrary documents, as with no need to embed links directly in documents there was no need to restrict the allowable document types.

One of the earliest examples of such a system was the Sun Link Service (Pearl, 1989) which described a protocol based approach where the management of the data i.e. the documents and content is loosely coupled with the management of links. Their design proposed that link editing and management be layered on top of existing software so that it didn't tie users to specific formats or software. It was also quite influential in that it was among the first systems to describe linking as a service offered by the system rather than as the core function of the system.

While this may seem to be a tiny difference it has a lot of practical ramifications. In offering linking as a service you will typically have to separate the links from the documents which changes how you implement linking itself, but also how you approach link integrity. For example, as we'll see shortly with Microcosm, you can monitor the documents that you know are the sources of targets of links and automatically fix or flag broken links as they occur. On the other hand having the links external to the document makes them more difficult to maintain in some cases, such as when you want to link to specific portions of the documents but the document can change over time.

The open hypermedia movement produced some useful link integrity work which is discussed later in Section 3.2.2 as this laid the foundations for later research. In broader terms of their place in the evolution of hypermedia and the notion of the link we first need to consider the Dexter Model.

3.2.1 Dexter Model

As the number of different systems proliferated in the late 1980s the hypertext community felt there was a need to formalise the concepts of hypertext. This resulted in a meeting of the major US Research groups in order to formulate a standard model for hypertext. As a result the Dexter Hypertext Reference Model (Halasz and Schwartz, 1994) was created as a high level model of hypertext, in fact it proved to be too high level and no system ever fully implemented this model. The interesting result of the Dexter model was the formalisation of the concept of links which would influence the next few years of hypertext research.

In the Dexter model a link is as an entity which represents relationships between components, these are more commonly referred to as nodes or documents. The model defines a link as being a *sequence of two or more "end-point specifications" each of which refers to (a part of) a component in the hypertext*, this means that a link connects 1 location to potentially many locations. Locations for links in Dexter are specified by anchors which are composed of a unique ID and some arbitrary value specifying in some way the region of interest, in the model the system need not understand the anchor value but defers this to the relevant viewer. This means that a link is actually composed of a component ID specifying the node or document in question and an anchor ID specifying where in the node or document you're linking to or from. The directionality of links is important since the model allows for links to be one or two way - strictly speaking the definition of a link is a two way relationship between two objects but in practise most systems past and present only use one way links, and therefore are technically pointers and not links. Note that this notion of linking is essentially the same as that of Nelson's original notion of hyperlinks from his Xanadu research.

The problem with this notion of links is that it imposes a number of constraints which many hypertext systems regularly violate and which unnecessarily restrict the functionality of systems. Firstly the model requires that all links must resolve to a specific resource i.e. they cannot dangle which raises two issues:

1. When authoring hypertext initially it is quite normal to link to a node or document which doesn't exist because you haven't produced it yet. According to the model you can't do this, which will impose extra authorship effort on the hypertext

author. Yet this is very common on the current Web, for example the MediaWiki[5] content management system which powers Wikipedia[6] and many other websites allows for you to create links to pages that don't yet exist, and without this feature it would be very difficult to author these sites.

2. Dynamic links are not supported by the model since they don't resolve to a fixed resource and may not in fact resolve at all. These kind of links are essential for providing the navigation by querying, discussed by Halasz in his Seven Issues (Halasz, 1988) and mentioned earlier in this chapter, see Section 3.1.3.1

Secondly the model's use of the anchor concept means that it doesn't have the ability to express the use of embedded links since it requires links and anchors to be fully external to the nodes. While this may be desirable in some systems most will use some level of compromise with some or all of the data being embedded in the actual node. Thirdly leaving the processing of anchors entirely to the viewer applications and not doing it within the system is problematic. It's up to viewers to maintain the anchors when in fact they may be completely unaware of the hypertext system and unable to fulfil this function, thus rendering anchors quickly useless as a document changes.

As already mentioned no systems ever met the Dexter model fully since in trying to express a broad model to accommodate most people it ended up being too high level to be of real use. In particular the notion of links was not suitable for the actual systems people were building and wanted to build - especially the requirement that links must be resolvable. As we will discuss later in Section 4.1, without the ability for links to fail the World Wide Web likely wouldn't have been able to expand as quickly as it did or become the pre-eminent hypermedia system that it is today.

One interesting feature of the Dexter model is that it contained the concept of typed links, which were generally ignored on the document Web until the recent push in certain parts of the hypertext authoring community to produce 'Plain Old Semantic HTML' (POSH)[7].

Yet in the context of the Semantic Web typed links are very relevant, as Resource Description Framework (RDF) is based upon linking information together and all links

[5]http://www.mediawiki.org

[6]http://wikipedia.org

[7]See http://en.wikipedia.org/wiki/Microformat#Plain_Old_Semantic_HTML_.28POSH.29 for a definition and technical overview of this

(i.e. predicates) have a URI they are implicitly typed, therefore it should be obvious that ultimately the semantic web is composed entirely of typed link. There are no untyped links because predicates can only be URIs as defined by the RDF abstract syntax (Klyne and Carroll, 2004). As we will show later in Chapter 6 typed links can be exploited as part of a strategy for fixing broken links on the Semantic Web.

3.2.2 Link Integrity in Open Hypermedia

Microcosm (Fountain et al., 1990) and Hyper-G (Kappe et al., 1994) were among the first hypermedia systems to really consider the issue of link integrity in depth. Microcosm is notable in that it was initially designed as a single machine system so considered the issue of link integrity in-depth for non-distributed environments. Hyper-G is important since it was a distributed system that competed against the early World Wide Web (WWW) and so looked at the issue in terms of how it applies in a distributed system.

3.2.2.1 Microcosm

Microcosm was designed originally as a system to be run on a single machine, and in common with other open hypermedia systems it was designed to allow linking between arbitrary documents regardless of their format. Since for many formats it was not possible to embed links directly in the documents in question, the links were stored externally and managed by a component known as the link server in the style of the Sun Link Service (Pearl, 1989). Though in later revisions of the system Microcosm evolved into a distributed system most of the link integrity work focused on a single machine environment.

While making the links external to the documents had many advantages in terms of being able to link between any documents you desired it also had its disadvantages. Microcosm allowed for links which pointed to specific parts of documents, but since these were stored externally they were susceptible to breaking when the document was edited. In his thesis Davis (1995) describes techniques that can be used to solve this issue such as storing a copy of the data around the target of the link, in order that it can be relocated when the document is edited.

A tool called LinkEdit was utilised by users to help fix links when they were detected to be broken. One limitation of this approach is that it relies upon the system understanding the document format sufficiently that it can extract data around the target of the link, and use that to relocate the link target in future. This may prove easy for some formats but much more difficult for others, particularly if they are proprietary or closed formats, not to mention the tool needs to understand every format you want it to work with. The other more fundamental limitation is the need for human involvement, a human could maintain a small dataset assuming the data changed at a sufficiently slow frequency. Yet a human can clearly only maintain so much. Any system that requires significant human involvement in the loop will not be scalable, nor will it be viable as per the definitions we laid out in part 2 of our hypothesis (see Section 1.1).

One idea that Microcosm provides and which has been used in subsequent approaches to link integrity is that of monitoring the data you are interested in. If you know precisely the set of documents that you wish to maintain links between (as they did in Microcosm) you can monitor these and spot link integrity problems as they occur e.g. the deletion of a document which is the source or target of links. Depending on the system you may then be able to fix these problems automatically or inform a user who can decide what action to take. For example, if you can detect the act of a file being moved from one location to another you can reroute all links to that document to the new location without any user involvement being required. This approach is not without its disadvantages since it is enabled by the external storage of links. In systems with embedded links this is much harder to achieve since the tooling needs to understand the document formats in order to fix the links. Once again scalability will be an issue in that monitoring the data requires some amount of computing resources to perform, and however well your monitoring system scales there will always be a limitation on the amount of data you can monitor. This will likely be dictated by the amount of resources you have available or the cost of those resources.

3.2.2.2 Hyper-G

Hyper-G was designed to be a true distributed hypermedia system from the start, unlike Microcosm, and though contemporary to the WWW had some important differences from it (Pam and Vermeer, 1995). Hyper-G documents were primarily stored in their HTF format (Kappe, 1993) which like the fledgling WWW's Hypertext Markup

Language (HTML) format was based upon Standard Generalized Markup Language (SGML). A key difference in these formats is that HTF permitted overlapping links i.e. one part of a document could link to more than one other document. The most fundamental difference between Hyper-G and the Web was that all links in it were bidirectional, as we've mentioned previously this presents some additional challenges in maintaining links since there are two points of possible failure. Also Hyper-G was characteristically an open hypermedia system in that all its links were stored in a separate database. Though Hyper-G's authors do not refer to it as such this is essentially a linkbase and so we shall refer to it as such in this section.

It is of interest to us because Kappe (1995) proposed one of the first solutions for maintaining link integrity in a distributed system . Kappe's concept is similar to the idea from Microcosm in that you monitor the data you are interested in, but it's applied in a much more scalable way. Firstly it makes a distinction between internal links which can be maintained using techniques like those in Microcosm, and the external links to other servers in the network which must be maintained differently.

The p-flood algorithm which he proposed was designed expressly for this purpose and implemented within the Hyper-G system. Note that while the algorithm as described in his work is used for maintaining link integrity it is actually more general and could be used to propagate practically any kind of notification you desired across a distributed system.

In the p-flood architecture each participating server has its own linkbase which contains all the links both local to itself and those pointing to external resources. As is discussed in his paper the maintenance of local integrity is relatively easy and can be done using the techniques discussed earlier in this section, whereas maintaining distributed integrity is more complex. In his algorithm documents which contain links to external servers are referred to as surface documents and the links referred to as surface links. Each server maintains a linkbase of surface links and the meta-data of the surface documents including those from external servers, so that when a change occurs which affects the integrity of the links e.g. the deletion of a surface document other servers can be informed of this change.

Getting all the servers that are affected to coordinate in real-time is infeasible for various reasons as discussed in his analysis in Section 3.2 of the paper. In that section he discussed multi-server transactions as based on the work of Coulouris and Dollimore (1988) on distributed systems, and as a result of his analysis he proposed a weak-consistency approach instead (see Definition 3.1). The p-flood algorithm he presents is optimized for scalability since the amount of traffic generated is not dependent on the number of references to the altered object and the recipients are not required to be available at update time. The algorithm meets the weak-consistency requirement since it guarantees eventual delivery of every update message to every server, for example even if a server is unavailable for a considerable period when it becomes available again it will receive all messages sent during the period of downtime. To achieve this a probabilistic design for the algorithm is used to provide both the necessary scalability of update propagation and to be entirely automatic i.e. no need to configure manually how updates are distributes amongst the servers.

Definition 3.1. A weak-consistency approach accepts that the hyperweb being maintained may be inconsistent for a certain period of time but guarantees that it converges to a consistent state eventually.

The advantages of a weak-consistency approach are clear in that it provides for a system which is truly scalable since there is no need for all the servers affected by a change to be aware of it immediately. As long as users are willing to accept temporary broken links - which clearly they do since they encounter and accept permanently broken ones regularly on the Web - then a weak-consistency approach is sufficient to solve the problem. The disadvantage of such a system is that it requires everyone to buy into it, in that in order for the system to work the vast majority of servers in a system must participate or it will simply be ineffective. It is this disadvantage which meant that systems such as these have never been applied on a truly large scale because getting them implemented and deployed sufficiently has not been feasible, let alone possible.

3.2.3 The rise of the World Wide Web

As we have just discussed open hypermedia systems were contemporaries of the fledgling WWW. Though open hypermedia systems were arguably more technologically advanced and powerful than the Web, it was ultimately the Web that would grow to be the largest hypermedia system in the world. In the next chapter we look at the rise of the Web and the efforts to solve the link integrity problem for it.

Chapter 4

The World Wide Web, Semantic Web and Link Integrity

4.1 The World Wide Web

The World Wide Web (WWW) is a distributed hypermedia system designed at CERN by Berners-Lee et al. (1992). It was created originally as a means of collaboration between scientists but it quickly became a global phenomenon and ultimately the most widely used hypermedia system on the internet today. The advantage that the Web had over previous distributed systems was twofold: *1*) it was based on open standards; and *2*) it did not enforce link integrity. Without these two advantages it is doubtful whether the Web would or could have grown to its current size today.

Firstly we must note that the Web can be characterised to a certain degree as an open hypermedia system. Like open hypermedia it permits the linking of arbitrary documents and is primarily built around encoding data in an open format. One of the main areas where it differs is that it was in many ways far more open than the existing open hypermedia systems since its infrastructure, as well as the data contained in it, was embodied in open standards.

Another key difference was that the Web only used uni-directional links unlike Hyper-G and Microcosm which both supported bi-directional links (Hyper-G mandated bi-directional links while Microcosm could use uni/bi-directional links). While this was seen as a major flaw by some (e.g. Nelson as discussed earlier) it actually has advantages since it makes it much easier to distribute the system. This is because creating a uni-directional link does not require central management of the links as they can be embedded directly in the source document. Therefore no coordination between the owners of the source and target of the link is required to create a link as may be the case systems with bi-directional links. This does have a downside for us as far as link integrity goes because it can be far harder to ensure the integrity of such links. Since there is no coordination or central management present in the system there is no central place to manage the integrity of links.

Of course the Web was not the only global distributed hypermedia system to be deployed in the early 1990s but it quickly grew to become the largest and most well known hypermedia system in existence. One of its main competitors at the time was Gopher, as of March 1994 it outstripped the Web in terms of raw bytes of traffic (see Figure 1 in Berghel (1995)) but was in significant decline just a year later. Gopher, like the Web, was built on open standards (Anklesaria et al., 1993) and proved very popular initially but was quickly superseded by developments in the Web. This was primarily in terms of the emergence of early web browsers like Mosaic (Andreessen and Bina, 1994) which subsumed the features of the protocol, but also due to the fact that the Web's document format (Hypertext Markup Language (HTML)) provided for more flexibility than Gopher's did.

A further difference between the Web and its competitors - both open hypermedia and Gopher - is that the creators of the Web made all the software they developed available for free to anyone. Most open hypermedia systems had either been pure research projects i.e. not generally available, or had been spun out into commercial products by the researchers involved e.g. Microcosm was commercialised as the company Multicosm[1]. As is often the case with any product users will gravitate to a particular technology because of the price or perception of it as much as any other factor e.g. Betamax versus VHS (Cusumano et al., 1992). The Web was often perceived to be more open and less commercial than competing systems, for example the University of Minnesota's move to commercialise their Gopher server in 1993 is said to have dissuaded many potential adopters. The University of Minnesota eventually reversed this decision releasing Gopher under the GNU General Public License (GPL) in 2000 by which time it was already far

[1]`http://www.multicosm.com`

too late since the Web was clearly dominant and expanding ever faster.

4.1.1 The Web technology stack

Returning to our earlier point about the Web being based on open standards we must first point out that these were not formal rigid standards defined by a body like the International Standards Organisation (ISO) but organic informal standards created through the Internet Engineering Task Force (IETF)'s Request For Comments (RFC) process. This is in of itself an important factor in the success of the Web since such standards are able to evolve much quicker in response to user and implementer feedback, as well as offering the flexibility to add custom extensions which can then be standardised in later revisions.

The Web is in fact composed of a stack of technologies each of which has its own standards which have evolved over time:

- *Uniform Resource Locators (URLs)*

  The URL is a fundamental part of the Web's architecture, it is an identifier for a resource used to indicate the address from which it can be retrieved. URLs were originally defined in RFC 1738 (Berners-Lee et al., 1994) and later evolved into the generalised notion of Uniform Resource Identifiers (URIs) (Berners-Lee et al., 1998, 2005) which are central to the Semantic Web.

  One criticism of URLs regarding link integrity is that they are limited in that they directly encode the address of the resource in the identifier, so if you move the resource the identifier you immediately break any links to that identifier. As we will discuss shortly there has been some proposals for persistent identifier schemes that can be used to mitigate the link integrity problem.

- *Hypertext Transfer Protocol (HTTP)*

  HTTP is the protocol that defines how servers and clients communicate with each other on the Web and the Semantic Web, it was standardised by a series of RFCs (Berners-Lee et al., 1996; Fielding et al., 1997, 1999). Given a URL a client can attempt to retrieve the resource it identifies by making a HTTP request to that address. The action of doing this is sometimes referred to, particularly in the context of the Semantic Web, as the URL or URI being dereferenced.

The protocol has some bearing on link integrity since one of its core features is the notion of status codes, these are the first part of a response sent by a server and indicate the status of the response. These are useful since there are a number of codes that can be used to indicate that a resource cannot be found (thus allowing detection of broken links), but also codes that indicate a new location for a resource which allows automatic maintenance of links. There have been some systems that have exploited this or used similar approaches proposed for providing link integrity and again we will discuss these later in this section.

- *Hypertext Markup Language (HTML)*

  HTML was the main document format for the Web initially specified very informally before later being specified via RFC 1866 (HTML 2.0 (Berners-Lee and Connolly, 1995)) and then through the formal World Wide Web Consortium (W3C) standards process. HTML was important to the success of the web since it provided a reasonable amount of flexibility in terms of the content you could create, but also that it relied upon direct embedding of links into the documents.

  Direct embedding of links was significant since it made the barrier to entry for authors very low, you needed no software other than your text editor to create a document for the Web and you could easily link to existing content elsewhere on the Web. As mentioned earlier embedding links directly in documents does have issues for link integrity since if a link breaks you potentially need to edit and correct every document that contained that link in order to correct the problem.

  Unlike other formats around at the time in competing systems the authors of HTML were keen to alter and extend the specification rapidly based on community feedback. This can be seen in the acknowledgements section of RFC 1866 (Berners-Lee and Connolly, 1995) where they cite the work of Dave Raggett (Raggett, 1993) in drafting a standard for new features that were incorporated into the 2.0 standard.

All these open, communally evolved standards meant that anyone who wished to get involved in the Web could easily do so. The standards were relatively simple to understand and free software existed at every level of the technology stack.

Arguably the most important advantage of the WWW was the second point - that link integrity was not enforced. Surveys of broken link prevalence such as Lawrence et al.

(2001) have made their own arguments of this point. Since the system was distributed its designers saw that it would be very difficult or impossible to enforce link integrity within the system and decided that it would be better if links were allowed to fail. As part of the design of HTTP they included the status codes functionality which was mentioned previously. These codes allow a server to inform a client that a resource cannot be found (the familiar 404 error status), inform them that a resource is permanently unavailable (the rare 410 error status) or that a resource has moved temporarily or permanently (various 3xx redirection statuses). It is hard to know whether this was an explicit decision by the people involved or not though this is somewhat irrelevant to our argument. The WWW allowed links to fail and thus it could scale much easier

This feature of HTTP is key as it allows for some level of link integrity to be introduced into the system by appropriately equipped servers and clients, and these can take action when certain HTTP statuses are encountered. As such functionality is optional it does not encumber clients and servers who are unconcerned about link integrity and therefore lowers the barrier for entry to those who wish to publish documents on the Web.

4.1.2 The Rise of Portals and Search Engines

As the system grew people found that the Web was quite difficult to navigate, this is sometimes referred to in the literature as the hypertext navigation problem. This problem comes from the observation, based upon studies of usage of hypermedia systems, that users often find that they get lost or disoriented in the system (Nielsen and Lyngbæk, 1989; Nielsen, 1990). Either they were unable to find the information they were looking for or they were unable to retrace their steps to find previously viewed information. While the early web browsers, and all modern browsers went on to address some of these issues with their provision of bookmarks, favourites and history functions this did not allow users to find brand new information.

One interesting insight on this problem is that most of the authors who highlighted it were talking either about the Web at a time when it was relatively small or about small closed hypermedia systems running on single machines or small networks. These authors did not really concieve of how massively hypermedia systems would eventually scale and that solutions for this problem could be found. Though this problem obviously existed it was arguably merely an artifact of relatively immature systems, as we discuss in this section new services quickly appeared on the Web and in browsers to mitigate the issue.

Powsner and Roderer (1994) gives an overview of search systems in various contemporary hypermedia systems. As they note in their paper '*WWW and Mosaic software systems alone do not facilitate browsing and searching. The user must know in advance where to find a document of interest.*' (Powsner and Roderer, 1994). Clearly there was a need for services which allowed users to discover new information that was of interest to them, and as a result the notions of Web portals and search engines came into being. As Powsner's paper shows these were not new ideas, as with much else in the make up of the Web the ideas were adapted from existing hypermedia systems. For example other contemporary systems either used these as a central part of their design e.g. Wide area information server (WAIS) which was fundamentally based around the concept of text searching, or they already had equivalent services e.g. the Veronica[2] search engine for Gopher.

In the early days Web portals such as Yahoo[3] started out as being the digital equivalent of phone books maintaining categorised and/or alphabetised hand curated directories of websites. The problem with these portals were that they relied on human curation of the dataset and as the Web expanded exponentially this became infeasible in terms of cost and time. This resulted in the design and development of search engines which indexed the web automatically and provided keyword based search for finding web pages (Schwartz (1998) provides a nice survey of this). Of course the most widely known of the search engines today is undoubtedly Google[4] though they are known as much for their other technology as their core search technology.

Portals and search engines are a big part of the history of the link integrity problem since these websites allowed users to find content they needed without relying on links. As a result of these developments research into link integrity fell somewhat by the wayside. When a user encountered a 404 error or didn't find the content they expected it was trivial for them to use a search engine or portal to discover alternative sources of the content, or to find related content. This lack of interest in the problem is reflected in the level of research into link integrity which drops significantly in the late 1990s and early 2000s. Researchers felt that it simply wasn't worth investigating the problem since

[2] http://en.wikipedia.org/wiki/Veronica_%28search_engine%29
[3] http://www.yahoo.com
[4] http://www.google.com

users did not appear to care and creating a general solution appeared intractable given the scale of the Web and its continuing exponential growth.

4.1.3 The Problem with URLs

The problem with URLs from the point of view of link integrity is that they directly encode the location of a resource in them. Therefore any link to that resource points to a fixed location, and should the location of the resource be changed all links to the resource immediately become broken. In theory it would be nice if people did keep their URLs persistent in line with the best practise guidelines laid out by Tim Berners-Lee himself in his oft-cited *Cool URIs don't change* design document (Berners-Lee, 1998a). Unfortunately in reality many people don't know or care enough about link integrity to feel compelled to do this. In most cases when people do this it is usually only because the software that powers their websites does it for them e.g. permalinks in Wordpress blogs[5].

This non-persistence is a major contributor to the problem of link integrity as highlighted in surveys such as Ashman (2000). Essentially people would like their links to be permanent or persistent but they are stuck using identifiers which by their very nature cannot meet this requirement because the location is directly encoded into them. As a result there have been various proposed solutions to link integrity over the years that have advocated using alternatives to URLs in order to provide persistent links.

4.1.3.1 URNs

One of the proposed solutions is the creation of new URI schemes where an identifier for the resource is encoded rather than its location. Then resolver services can turn these identifiers into URLs at resolution time so links always go to the correct location of the resource. The most well known, yet rarely used, scheme designed for this purpose is the Uniform Resource Name (URN) scheme (Moats, 1997) which provides a URI syntax for giving your resources names which are then resolved to URLs when necessary. URNs use namespaces to subdivide names into schemas such that formally registered schemas

[5]`http://codex.wordpress.org/Glossary#Permalink`

have known resolver services, therefore a client that understands URNs can query the correct resolver to get an appropriate URL and actually retrieve the target of the link.

URNs have the advantage that the owner of that URN can choose to move the resource to a different URL at any time provided they update the resolver service with the new URL for their URN. Provided that others on the Web have linked to the URN rather than the URL the act of relocating the resource does not break any links to its URN. In theory if their usage was widespread one of the most common causes of broken links could quickly be eliminated and you would have solved link integrity to some degree.

Unfortunately this is also their main disadvantage since you typically need a centralised resolver service which goes against the decentralised nature of the Web. It is possible to reduce the centralisation requirement by having multiple resolvers at different servers for the same URN schema, but this then introduces the need to keep the resolver services synchronised or aware of each other. As we have already seen in the work of Kappe (Kappe, 1995) the synchronisation of a distributed network of servers to provide link integrity is always complex no matter how good your algorithms are. Note that there are alternative persistent identifier systems that do not require centralised resolvers e.g. Handles which we shall discuss shortly.

There is also the potential that the use of URNs can lead to more widespread link integrity issues as in the event of a resolver failing vast numbers of links would be rendered broken. If the resolver service is unavailable then all clients trying to resolve them will be forced to return errors to their users. On the surface this seems very similar to the typical 404 error situation encountered in normal usage of the Web but is actually much worse. Without a working resolver service the client has no way of knowing whether the URN is valid so they cannot resolve it and return a resource to their user, but neither can they give a definitive answer as to whether the link is working or not.

4.1.3.2 PURLs

An alternative approach to the problem is simply to use a URL redirection service where you have a central service that provides you with URLs which you associate with the actual URLs of the resource. This way you can use the redirection URL as the public

URL for your resource and can move it freely between servers by simply updating the actual URL at your redirection service. Just like URNs this provides link integrity since as long as people link to the public URL of a resource its actual location is irrelevant, even if you move it links to the resource keep working. In essence this has the same benefits of URNs i.e. permanent identifiers without the extra overhead since there is no need to discover which resolver service to use since that is directly encoded in your public URL.

The most widely used service, particularly on the semantic web, is the Persistent Uniform Resource Locator (PURL) service provided by the OCLC (OCLC, 1995). Unfortunately such services have the disadvantage that they are reliant on a centralised service and unlike URNs they cannot have multiple resolvers since the URLs implicitly encode a single resolver in them.

4.1.3.3 Handles

The Handles system is similar in nature to the concept of URNs but is far more advanced as a technology in that it both specifies and provides a decentralised system for the resolution of handles. Just like a URN a handle is an identifier for a resource that can be resolved by a resolve service into an actual location from which the resource can be retrieved. Specified in a series of RFCs (Sun et al., 2003a,b,c) it is already widely deployed on the Web particularly in the area of scientific publishing. The most well known and common persistent identifier scheme is the Digital Object Identifier (DOI) scheme (Paskin, 2010) which is widely used with around 50 million identifiers assigned to date.

The Handle system avoids many of the potential disadvantages of a resolver system by being designed with reliability and redundancy in mind. Although it incorporates a global resolver as part of its architecture this is not in fact centralised, it is explicitly physically decentralised as well as being logically mirrored. Additionally this global resolver does not do most of the registration itself, rather it serves as a hub which issues identifier prefixes to local resolvers. Thus the global resolver need only redirect a client to the local resolver for a prefix in order that the local resolver can do the actual resolution. Since a specific prefix always maps to the same local resolver once a client knows of a resolver for a prefix they do not in practise need to use the global resolver for subsequent resolutions of identifiers with the same prefix. As a result the handle system is a very effective system for creating persistent identifiers and thus ensuring link integrity.

So if the system works so well why isn't its usage more widespread? While the DOI foundation[6] states that there are over 50 million identifiers issued to date that is clearly a drop in the ocean compared to the billions of Web pages, and therefore URLs, that exist on the Web today. This lack of uptake is likely due to the increased barrier to entry it provides for those wanting to create websites. If you wanted to use the Handle system then you would have to obtain identifiers for every document you published and provide a local resolver which knew how to resolve your identifiers. While this may all in reality be relatively simple to do since there is free software available that can do all of these things for you, it is enough extra effort to put off many from using it.

In addition personal use of handles will by no means ensure that people actually link to them since browsers don't understand handles directly. Therefore people visiting your documents would see the URLs of the documents rather than their handles in their browser and most likely create links to those. In doing this, links to your documents would not be persistent since when the location of a resource is moved the links would break as they would have used the URL rather than the handle.

4.1.4 Link Integrity for the Web

Despite the relative lack of interest in the link integrity problem since the rise of the Web there have still been small groups of researchers who have continued to try and address the problem over the years. In this section we will discuss some of the solutions that have been proposed and look at their relative advantages and disadvantages. As we outlined in the introduction the focus of our research was to look at these existing techniques and adapt those that were appropriate to the Semantic Web. To this end as part of this section we highlight a number of possible solutions which we think are suitable for this and give brief descriptions of how such adaptations might work.

[6] `http://www.doi.org`

4.1.4.1 The W3Objects approach

One of the earliest published pieces of research concerning the application of link integrity to the WWW was the W3Objects approach (Ingham et al., 1996). Their approach was to layer a higher level abstract model - the W3Objects of the title - on top of computer networks with the Web being just one possible network their objects could be exposed via. Then to ensure referential integrity, and thus link integrity, between the objects they have a distributed reference graph and garbage collection system which can be exploited to ensure the integrity of links. This approach can be classified as a maintenance approach since it is based upon the automatic maintainenance of links within the system.

Given this object model they describe various mechanisms within it that can be used to track the movement of objects and thus ensure that links always remain functional. The main means by which they do this is a forward referencing model, when an object is moved an object is left in its place which holds a reference to the new location. As long as you always replace the object with a forward reference to its new location when you relocate it links can always be resolved since they can follow the chain of forward references to reach the actual object. Additionally their system permits for the automatic maintenance of links i.e. when a link is resolved the resolver knows that it has followed forward references so in some cases it can update the source of the link to point directly to the new location thereby speeding up future link resolution.

As well as this forward referencing system for maintaining links when a resource is relocated they also provide a *gravestone* mechanism for dealing with the deletion of resources. Rather than a resource being deleted completely it may be replaced with a gravestone which acts as a marker to tell clients which attempt to retrieve the resource that it is no longer available. The intention of this being that clients can remove their links to the gravestone and eventually the gravestone may be removed permanently.

While these ideas all sound good on paper they have a fundamental flaw in that they require you to rearchitect the Web on a massive scale for this to be a viable option. Even by 1996 when this solution was proposed the Web was always growing too fast and its core architecture already firmly established to a point where such a solution would never garner widespread support or adoption. Despite this the idea of having forward references and gravestones is a sensible one which is already present in the Web.

As stated in the earlier description of the Web's technology stack (see Section 4.1.1) HTTP does in fact have a mechanism for doing exactly this. Forward references can be represented by configuring your server to issue 3xx redirect status codes for the relevant URLs which require forward redirects and 410 Gone status codes for those that require gravestones.

4.1.4.2 Recovery approaches

Phelps & Wilensky introduced the concept of lexical signatures for Web pages in their Robust Hyperlinks paper (Phelps and Wilensky, 2004). Their approach computes what they term the lexical signature of a page and appends it to all links to that page so that in the event of the link failing a browser plugin can use the signature to relocate the page using a search engine. A lexical signature is represented as a set of terms - their paper states that 5 terms is optimal - which characterise the resource sufficiently that when a search on those terms is performed across multiple search engines the consensus top result is the new location or a very close match to the originally intended document. The limitation is that this relies on the web page having been available on the web for a sufficient period of time to have been crawled by search engines. Plus it must not have been missing from the Web for long enough to have been removed from the search engine caches. Despite this limitation their approach proved remarkably effective in their testing. The obvious flaw in their work was that it required rewriting all the links on the Web to use this robust hyperlinks approach which was never going to be practical or scalable.

However their research is interesting as it allows for the recovery of links when they break, reducing the need to monitor the data regularly as with most maintenance or replication based approaches to link integrity. It is important to highlight this since this feature of the technique implies that these techniques will be far more scalable (and thus viable) than many of the other approaches which we have discussed in this section. Their technique was somewhat limited in that it still required you to apply it ahead of time i.e. you had to actively decide that you wanted your links to be made robust and the end user would have to have a browser which understood the system for it to work.

Later research by Harrison & Nelson built on Phelps & Wilensky's concept and showed that these signatures need only be computed Just-in-Time (JIT) when a link fails (Harrison and Nelson, 2006). In their Opal system the signatures can be computed JIT by retrieving cached copies of the pages from a search engine cache, computing the signature and then using search engines to relocate the page. Opal used multiple redundant servers which provided a user interface to this automatic retrieval of the intended content rather than just redirecting users to what it believed to be the correct alternative source of the desired content. Instead it offered users multiple options for alternative sources of the content and allowed them to give feedback as to whether the alternative sources were correct. This meant that the system gained feedback from users which let it refine its suggestions in the future, and the servers themselves were able to synchronise data between them so all servers learned from this user feedback.

One problem they had with their system was that the synchronisation and learning aspect of the servers did not scale particularly well. To start with servers were limited in the number of links they could recover each day by the Application Programmers Interface (API) limits imposed by the search engines used e.g. Google had a limit of 1,000 request per day at the time and today that has fallen to 100, though you can pay to receive up to a maximum of 10,000 request per day. Recovering just one web page may require anything from a couple to hundreds of requests depending on the number of unique terms on the page and whether Opal has prior knowledge of the terms or the broken URL. They calculated that it would take 10 servers harvesting each others data approximately 3 years to learn a sufficient corpus of term frequencies to be truly viable.

Their approach has clear advantages over the original robust hyperlinks work and over other approaches to link integrity in that it doesn't require the user or publisher to care about link integrity ahead of time. Yet it is still capable of providing a highly effective means to recover the desired content should the link to it break for any reason. The main disadvantage of such an approach is that it still requires the user, or at least their software, to be aware that there is such a system available and to make use of it when a 404 is encountered. The other disadvantage is the scalability problem which they themselves outlined.

Despite this we think that such techniques have real promise in delivering viable and scalable link integrity to the Semantic Web. Later in Chapter 6 we outline and evaluate a link integrity solution for the Semantic Web based on these techniques.

4.1.4.3 PageChaser

PageChaser was a system proposed in Morishima et al. (2008) and is a fairly standard monitoring based approach to the problem in a similar vein to the mechanisms provided by systems like Microcosm and Hyper-G. The user registered the URLs they are interested in and the system monitors those resources and gathers relevant information for use in relocating the resource in the event of a link failure. When a link actually fails this information is passed to the PageChaser of the name which attempts to relocate the resource using a variety of heuristics. Their paper details several heuristic rules which they determined through their experiments to be good at determining where the new location of a resource is likely to be. These heuristics are important in the context of link integrity since they codify various properties of moved resources on the web that have been used in various other link integrity approaches even if not explicitly stated.

For example **H1** in their paper states that '*It is expected that P(u′) is similar to P(u)*' i.e. that the new resource will be similar to the old resource. As we have already seen in the preceding section on robust hyperlinks the work of Phelps and Wilensky (2004) and that of Harrison and Nelson (2006) is based upon the same basic principle. If you know something about the resource you wish to relocate then you can find similar resources using search engines. Morishima et al. do in fact use search engines to implement the usage of this heuristic in their paper showing that this kind of approach really does work and is frequently used in link integrity solutions. As you will see in Chapter 6 our proposed recovery based solution for the Semantic Web also utilises search services on the Semantic Web as part of its behaviour.

Their approach is also interesting in general since they aim to exploit what they call '*the locality that exists in our problem i.e. a bias in where the place is likely to be*' which was a novel approach to the problem. Most systems like those we have already discussed focus only on using similarity metrics of some form to find moved resources and repair links. This proved to work well in their testing but it is unclear whether this *locality* that they mention extends to the Semantic Web. Typically in the Semantic Web domain we are not expecting that the data will be moved around by its publisher. Rather that the data may be temporarily unavailable due to maintenance or other transient issues, or that the original publisher may withdraw the data and a new publisher may publish it in an entirely different location.

On the whole their approach suffers from the same issues that other monitoring approaches suffer in that scalablity will always be limited to the amount of computing resources you have available for monitoring. Also as with similar solutions the user has to decide what they want to monitor and provide link integrity for, so if a link fails and it is not monitored the system is unable to help the user since it is reliant on the metadata it gathers during monitoring.

4.1.4.4 Author oriented approaches

In terms of more manual maintenance based approaches there have been a number of tools and systems developed to aid the author in ensuring that their links are working. While these may be able to affect some kinds of automated link repair they often require more significant involvement. A pre-WWW example of this is the LinkEdit tool from Microcosm (Davis, 1995), an interesting example of such a tool developed for the Web itself is Creech's change log table/web walk (CLT/WW) technique (Creech, 1996).

The idea behind his tool is that you model the common operations that an author enacts on their website e.g. move page, delete page etc. and use monitoring of the website to track these changes. By tracking the changes in the change log you can use this information to help the author correct broken links when they are detected. His approach requires a certain degree of author involvement in the process since he acknowledges that automated repair is not always possible and so in cases where it is not his system attempts to notify the relevant author instead. The CLT/WW approach periodically performs a web-walk which crawls the local website correcting links based on information found in the change log table e.g. if it finds a link to A and it knows that the author moved A to B it can either correct that link automatically or notify the author of the problem.

One issue that this has which Creech acknowledges in the paper is that if the website is updated during the course of a web-walk then you must either take account of those changes and risk inconsistencies in your repair actions or notifications or wait till the next web-walk to correct the issue. While this is a fairly minor implementation detail in the grand scheme of things this issue applies more widely to any monitoring based approach to link integrity. If you are relying on crawling of any kind of data for your monitoring you cannot guarantee that changes will be introduced during your crawl that may make your data incomplete or inconsistent. Whether you need to address this problem depends on the exact technique being used and whether these potential inconsistencies will actually affect your system. For example if you were monitoring some set of URIs and you found that a URI was working at time *T* then it may not matter to you if it stops working at time *T+1* provided you can detect that at a later date.

His system is notable in that it is designed to be integrated directly with the content management systems (CMSs) that many people and organisations use to build their websites, whereas most other systems we have described in this section are entirely separate from the process of creating content. In having this close integration the system makes the users more aware of link integrity and though not stated in the paper one hopes that this would have the effect of making users pay more attention to the issue in the first place. As we noted earlier typically the perception of users and authors is that the problem of link integrity is unimportant, systems like this which highlight it to the user should be welcomed.

Unfortunately his system is not without its issues, as we have already discussed in the context of other monitoring based systems it relies on continual monitoring of the content you want to maintain links for i.e. it cannot ensure that links to content outside of its monitoring are maintained. Also it has serious scalability issues in that its core monitoring behaviour is based upon web crawling which is expensive and time consuming in terms of computing resources, especially as the size of the data being monitored increases.

4.1.4.5 Replication approaches

An alternative approach to maintaining links is to maintain the data that you are linking to using replication and versioning. These approaches are useful in that they allow you as a user to replicate the data you are interested in and provide link integrity on your terms. Using such a system you can guarantee that links to the data you care about will always work since you can fallback to the replicated data whenever the actual data is not available. In some respects these approaches are similar to other monitoring approaches we have already discussed but they have the advantage that you can use them to maintain links in data that is otherwise outside of your control. Where they differ is that they are not maintaining the links rather maintaining a copy of the data that you are linking to.

In this vein Veiga and Ferreira (2003, 2004) discuss the possibility of turning the Web into an effective knowledge repository by using such an approach. Their work follows on from earlier work such as Moreau and Gray (1998) which proposed limited use of replication and versioning, but had significant reliance on author and user involvement in the process. The problem with Moreau and Gray's work was that it relied heavily on the author of the content be involved in the link integrity process. In their approach the author of the content had to state how long they were intending to make the content available and then the client could decide how and when to replicate or version the content according to the clients needs. It should be evident that this is quite unwieldy and imposes a lot of additional requirements on all parties involved in the publishing and consumption of data.

However in Veiga and Ferreira's work there is no requirement for author involvement in the process, only the end user need use a browser plugin to indicate the content they wish to replicate and preserve. Their results showed that the user could preserve the sections of the web they were interested in with no perceivable performance impact, on average there was only a 12ms increase in retrieval time for resources i.e. users would not notice that the system in operation. The advantage of such a system is that it does not place any onus on the publisher of content to provide this capability and instead provides it as the user level. Thus only users who want or need this capability need have the relevant software and there is no need for the architecture of the web to be changed or augmented. The disadvantage of such a system is that it still requires the user to know that they want or need to replicate or version a certain part of the web and they can't retroactively do this once the content they were interested in has been removed from the Web.

An interesting new approach to this problem has recently been proposed in the form of the Memento project (Van de Sompel et al., 2009, 2010) that advocates an HTTP based versioning mechanism. In their system, servers that wish to participate replicate and version their content however they see fit and then serve appropriate versions of the content depending on the HTTP headers. Essentially it provides for content negotiation by time with a HTTP server, which means that unaware clients need not be aware of this feature and will always receive the latest version of the content. However aware clients can use this to browse the version history of the data if they so desire. The advantages of such a system are clear in that it only requires those who want to participate to do so and that it does not require use of any special software or protocols in order to leverage it. Almost any HTTP client program or programming language with HTTP support already has the ability to send arbitrary HTTP headers with their requests. Therefore the entire HTTP ecosystem already has the ability to access this functionality when interacting with a server that provides it. The disadvantage of the approach is that the HTTP headers are not yet recognised as standard headers and that some poorly written servers may choose to reject requests if they don't understand a particular header.

On the whole such systems appear to be very useful and effective for certain use cases i.e. those where you know exactly the data you need to maintain link integrity for. As a result this replication approach is one that we have attempted to adapt to the Semantic Web and we present our efforts in this area in Chapter 5. It is worth noting that such systems do have a key drawback in that they are typically reliant on using a relatively large amount of computing resources both in terms of storage needed to replicate data and the processing power needed to monitor the data over time. We look at these issues in more detail in the context of evaluating our proposed approach in Sections 5.2-5.3.

4.2 The Semantic Web

As we stated in our introduction the Semantic Web is an evolution of the existing Web that aims to augment the primarily document centric model with machine-readable data. This data is typically expressed using the Resource Description Framework (RDF) model and published as part of a website in one of the various RDF serializations and things within the data are identified using URIs. The net result of this use of URIs as identifiers on the Semantic Web means that it is far more link-centric than the Web since every triple is potentially a link from one resource to another.

Also as we noted in our earlier discussion of the Dexter model (Halasz and Schwartz, 1994) the notion of typed links is much more central to the Semantic Web than on the document Web. As triples have a predicate which indicates the type of the link between two resources it is possible to have data which states multiple links between two resources. This provides new avenues in designing hypermedia, or at least reopens those that the limited WWW model of linking ignored, in that you can create and navigate links contextually. For example you might choose to create links between notable persons and their places of birth as is done in DBPedia[7] which would allow you to browse from a place of interest to all the notable persons born there or vice versa. Any resource may potentially be linked to any other resource with any number of typed links, this differs from the traditional WWW where one resource links to another only once. Though a link on the WWW may occur multiple times within a document it generally only represents one actual link since most links are untyped, note that there are a few exceptions to this e.g. links created via the `<link>` tag such as stylesheets.

One of the many advantages of typed linking is that it is possible to exploit them in link integrity as we will show later in this section and in our own work in Chapter 6. However, there is the potential that typed linking can exacerbate the link integrity problem on the Semantic Web since:

1. It is possible to create multiple links between the same resources so there are more links to be maintained.
2. The editing problem is increased because typed links potentially ties the link much more closely to the data that a resource returns, so there is a higher chance that data can change in such a way as to make the link invalid.

Another problem that affects linking on the Semantic Web is the coreference problem as it relates to URIs. The coreference problem itself is not a new problem and is well known in the fields of natural language processing (Bagga, 1998) and databases where it is often called record linkage (Winkler, 1999). The essence of the problem is that in many systems, including the Semantic Web, you may assign multiple identifiers (URIs in our case) which actually refer to the same thing. While there may often be very good reasons for doing this - e.g. an organisation may want to control their identifiers and what data they publish about them - it can create problems when it comes to linking since which identifier do you link to? As we shall describe shortly there have been a couple of competing approaches to this advocated for the Semantic Web and it is an active ongoing research area.

[7] `http://dbpedia.org`

4.2.1 Link Integrity for the Semantic Web

4.2.1.1 Memento for the Semantic Web

In the earlier sections on the WWW we discussed Memento (Van de Sompel et al., 2009) which is a system that aims to provide time based content negotiation. By combining replication of data with their time based negotiation a client can easily traverse versions of content on the Web. However, since their approach is based on HTTP it can be just as easily applied to the Semantic Web as they have shown in their subsequent research (Van de Sompel et al., 2010). In this paper they described how their approach could be applied to the Semantic Web and demonstrated it against the full DBPedia dataset. In doing so they demonstrated that their approach is potentially scalable and viable for use on the Semantic Web. As the approach is replication based it provides link integrity simply by providing a user the ability to fall back to the replicated version when a link cannot be resolved.

In terms of scalability the main barrier to the adoption of replication based approaches is the system resources required to implement as we have discussed in our examinations of pure WWW based replication systems. For example, in their experiments they loaded five versions of DBPedia[8] which required 81GB of disk space using a fairly crude database schema which stored minimal information. While this may not seem like much in the grand scheme of things when 2TB disks can currently be purchased for around £60 this represents just 5 versions of just one dataset. This approach could no doubt be deployed successfully on a much larger scale but is going to require large quantities of fast storage to be sufficiently scalable and performant. Clearly this will increasingly become a moot issue as the inevitable march of progress creates ever faster storage and more powerful computers to process the data, but we are still someway off a solution like this being viable at Web scale.

The disadvantage of their replication approach is that it is very simplistic, in that it assumes that you only wish to retrieve past versions of data and replicates only the information needed to do that. In reality you often want much more information about the data in such a system i.e. you want it to operate more like a version control system than a pure replication approach. From the way they describe their system in the paper

[8]http://dbpedia.org

it does not appear that they are versioning the data rather they are just storing a full copy of every version. This will harm the scalability of their approach especially if you have frequently changing data, for datasets like DBPedia that change infrequently this may not be an issue, but for data such as stock prices the amount of storage needed would rapidly skyrocket. Since the data is not versioned the system does not provide a means to determine what has changed from one version to another which may be important to clients consuming the data.

Additionally the system as presented does not appear to include any form of monitoring, most replication based systems we have discussed earlier in this chapter have included some kind of active monitoring of the data you wish to replicate. It appears that Memento is passive replication as the user has to load the versions of the data into the system themselves, there is no mechanism for ingesting the data automatically as present in other systems. This implies that the system is better suited to deployment by publishers of data rather than consumers, the problem with this being that it requires data publishers to buy into the Memento concept. As we have alluded to in our discussions of the history of the WWW this has typically not worked since most publishers of data have little or no awareness of link integrity, so it is unlikely that they would adopt this technology. On the other hand existing WWW systems like RepWeb (Veiga and Ferreira, 2003) have shown that you can implement a replication system purely from the point of view of the user quite succesfully and we feel that this approach is more viable. To some extent link integrity will always be a niche requirement of users and we would rather provide the capability to users who need or want it than try and convince publishers who are uninterested to adopt it.

Overall though their approach is very promising and is impressively simple in design and implementation. In Chapter 5 we present our own efforts at creating a replication based system for the Semantic Web.

4.2.1.2 DSNotify

DSNotify (Popitsch and Haslhofer, 2010; Haslhofer and Popitsch, 2009) is a more traditional monitoring based approach to link integrity that works in a similar way to the mechanisms in systems like Hyper-G (Kappe et al., 1994) and CLT/WW (Creech, 1996). Once configured it actively monitors the data the user has expressed interest in to ensure that links between the data remain valid. In their work they are not just looking at validity in terms of the link functioning, but also in terms of the editing problem i.e. they aim to ensure that the typed links between resources are valid. To monitor links between data they compute feature vectors for each piece of data that is a source or target of a link. These feature vectors serve two main purposes:

- In the event that the target of a link is moved it can be relocated by searching for data that matches the feature vector. If the feature vectors are a sufficiently close match then it is likely that this should be marked as the new target of the link. This is identical to the approach for relocating content seen in many other systems e.g. PageChaser and Opal.
- Feature vectors can be sufficiently rich that it is possible to determine whether a particular typed link is valid. As we have previously covered, all links on the Semantic Web are typed which exacerbates the editing problem since there is far more scope for links to become invalid as a link may only apply based on fairly limited criteria.

Their approach is interesting but in terms of scalability it is unclear how well their system would scale, since the computations required to determine whether links are valid are far more complex than those used in other systems. This is due to the fact that their system as they present it seems to be primarily concerned with the editing problem rather that the more frequently tackled dangling link problem. In the introduction of Haslhofer and Popitsch (2009) they state '*we define link integrity as a qualitative property that is given when all links within and between a set of data sources are valid and deliver the result data intended by the link creator*'. They have clearly chosen to focus on the editing problem as opposed to the dangling link problem. While this is an area that no doubt needs attention, we feel that with the current relative immaturity of the Semantic Web addressing the dangling link problem is a more practical and pressing issue at this time.

Their approach while fundamentally sound has some issues in that at some stage the user has to define the criteria which are used to determine whether a link is valid. In some simple cases you will be able to utilise general criterion to decide if a link is valid or not. Yet once the data gets complex, and in cases you are expressing links between disparately structured data the criteria will have to be manually defined. For evidence of this see Figure 2 in their paper and section II C of the paper (Haslhofer and Popitsch, 2009), four out of the six major components of their system shown in the figure may require domain specific implementations. It should be noted that this issue of requiring domain specific knowledge is not unique to their system, as you will see in Chapter 7 we found that one of our approaches could vary wildly in performance depending on whether or not domain specific knowledge was used as part of its input.

Their system can detect links that are broken in the dangling link sense almost as a by-product - i.e. situtations the target of the link is not available - but this is clearly not their primary concern. What they are doing certainly is valuable link integrity research, but seems to be more aimed at providing some guarantee of data integrity in terms of links in the same way you would guarantee data integrity in terms of constraints in a relational database. Most systems we have described in this and the preceding chapter have treated link integrity as a problem only in terms of success or failure of links i.e. can it be resolved or not. DSNotify is one of the few systems that actually tries to tackle link integrity in broader terms of whether a link is valid but because of this it does not really help us in our research because it is ultimately aiming to tackle a different problem. Though as we will discuss in our conclusions, systems which combine the ability to solve all the link integrity problems would be very useful and their systems and ours could potentially form part of a wider strategy to provide link integrity.

4.2.1.3 Tackling the Coreference Problem

There has been a lot of focus on the coreference problem as it applies to the Semantic Web since the proliferation of multiple identifier URIs for the same concept increases the difficulty of some kinds of problems e.g. reasoning. Since there are many organisations publishing similar data semantically (bibliographic databases being a prime example) there are frequently many URIs for a single entity such as an author. Coreference research aims to develop ways to efficiently and accurately determine URI equivalences and refactor the data or republish this information to help other semantic web applications. There are several competing philosophies in this area, one of which is the Okkam approach that advocates universally agreed URIs for each entity (Bouquet and Stoermer, 2008; Bouquet et al., 2008). A prominent alternative is the coreference resolution service (CRS) approach which determines coreferent URIs and republishes the information in dedicated triple stores (Glaser et al., 2007; Jaffri et al., 2008; Glaser et al., 2008).

The Okkam approach This approach is interesting in that it tends to favour the inclusion of centralised naming authorities in the process creating data for the Semantic Web. This has some practical benefits in that by having a central authority you can have a repository of possible link targets. Since one can use their service to determine what URI to use for a particular concept you can use it to decide what to link to. In practise this works by the client submitting some data that they want to get a URI to and the system tries to match the data against that of existing URIs to find a match. If there is no existing URI then it issues the client a new one and stores the data so that future requests can potentially return that URI when relevant.

The advantage of their approach is that linking to widely used URIs should hopefully reduce the liklihood of links failing since if everyone is using a particular URI that should encourage the publisher to ensure it remains functional in the long term. The disadvantage of their approach is of course the need for centralised control in the Web which has been shown by past research to be counter to the general culture of the Web. Though we have seen approaches like the Handles system which provide a similar kind of service for creating persistent identifiers we have also seen that these have not been widely adopted outside of certain niche areas e.g. scientific publishing.

The other problem inherent to this approach is shared with both DSNotify and the CRS approach in that you potentially require a degree of domain specific knowledge for the system to be effective. As the system has to determine whether one chunk of data is equivalent to the other, and thus the same URI should be used to refer to it, it must have a means of doing this. In some cases this may be relatively simple and require no domain specific knowledge but in other cases making the decision may be very difficult without it. For example, imagine being given some data that described a person named **John Smith**[9]. Without any knowledge of the domain in which you wanted to assign a URI it may be impossible for the system to return a usable result as it may have millions of potential candidate URIs and no way to determine which to return.

[9]Often stated colloquially as being one of the common names in English speaking countries http://en.wikipedia.org/wiki/John_Smith_%28name%29

The CRS approach This approach was created and championed by the ReSIST project[10] which uses it heavily within their RKB Explorer[11] application. The idea behind their approach is that you allow coreferent URIs to prolifereate as much as people desire and instead provide services which can tell clients the URIs that are considered coreferent to a given URI. In this way a client can discover alternative URIs to the one they are using and use this information to help them make links or for other purposes. To calculate coreference they apply standard techniques from the fields of natural language processing and database record linkage to the data. Note that as well as its usage in an academic conference they already provide a more general service called SameAs.org[12] which provides co-reference data in RDF formats allowing applications to consume and use this kind of information as needed.

As we have seen in other systems like Okkam and DSNotify such an approach requires a certain degree of domain specific knowledge as in order to calculate which URIs are coreferent you need to:

1. Be aware of the datasets that contain them.
2. Be able to express what metrics you will use to decide whether two URIs are coreferent.

This of course makes it harder to scale such systems rapidly since introducing new data requires a certain degree of human involvement in creating and testing the metrics used to determine coreference.

Despite this, systems of this kind have clear potential for use in link integrity as the information provided by a CRS or Okkam-like system could be utilised in a JIT fashion as in Harrison & Nelson's work (Harrison and Nelson, 2006). This could be used to help locate alternative sources of relevant data about a URI which is not resolvable. The recovery approach which we present in Chapter 6 is based in part around using services like this to provide some degree of link integrity.

[10]`http://www.resist-noe.org/`
[11]`http://www.rkbexplorer.com`
[12]`http://sameas.org`

4.3 Summary of Approaches

In these chapters we have covered the history of hypermedia and linking from the first proposals through to the rapidly evolving Semantic Web. In doing so we have discussed and evaluated a variety of different approaches to providing link integrity. In general these can be categorised broadly as follows:

- **Monitoring**

  Monitoring based approaches are one of the most common as seen in systems such as Microcosm (Fountain et al., 1990), Hyper-G (Kappe et al., 1994) and PageChaser (Morishima et al., 2008). The key characteristic of such systems is that they rely on knowing the links that the user wants to monitor in advance. Typically they gather and monitor relevant data from those links that they can use to repair links in the event of failure.

  While these systems can be very effective they have several weaknesses:

  1. They are only able to provide link integrity for the data the user tells them to.
  2. They rely on continuous monitoring of the data which can be costly in terms of computing resources required.
  3. They typically can only be applied to data which the user controls.

- **Replication**

  Replication based approaches are designed to replicate the data that a user wishes to maintain link integrity for as seen in systems such as RepWeb (Veiga and Ferreira, 2003) and Memento (Van de Sompel et al., 2009). While similar in some respects to monitoring approaches, they differ in that replication tends to require much more data to be gathered by a system and that they can monitor data outside of the direct control of the user. The one disadvantage they have over monitoring approaches in that they often require large amounts of storage to replicate the data effectively.

- **Recovery**

  Recovery approaches are quite different from the other types of approaches in that they are designed to operate at the point when a link fails rather than in advance of a link failing. While some systems may require some level of advance preparation e.g. Robust Hyperlinks (Phelps and Wilensky, 2004), others such as Opal (Harrison and Nelson, 2006) require no preparation other than awareness that a recovery service is available.

While these approaches may be among the most difficult to implement effectively they also show the most promise in terms of meeting the scalability and viability criteria and therefore the most potential to help us prove part 2 of our hypothesis (see Section 1.1).

4.3.1 Adaptation of these approaches to the Semantic Web

In this section we have outlined two promising approaches to link integrity from Hypermedia that we have worked to adapt for the Semantic Web in our research. In Chapter 5 we present our work in adapting the replication approach to the Semantic Web and then in Chapters 6 and 7 we detail our work in adapting the recovery approach.

Chapter 5

Replication and Preservation

As discussed in the previous chapter one of the main approaches that has been used in the past to provide some degree of link integrity to hypermedia is replication and preservation. This approach focuses on making and storing copies of the data that the user is interested in, then in the event of link failure the user can use the copied data instead of the original. Replication strategies can vary quite widely from keeping a copy of all content as in systems like the Internet Archive's[1] Wayback Machine[2] through to storing a base version of content and then subsequent deltas in a manner similar to version control systems.

The typical advantage of such systems is that they are relatively simple to design and develop since they need only replicate data in an efficient manner. Once developed systems only need to be configured to replicate the desired data after which they should be mostly self-maintaining. To actually provide link integrity they need only have a user interface or client Application Programmers Interface (API) which allows a user/client application to access the replicated data when they require it. The main disadvantage of these systems are that they require forward planning on the part of the user as you cannot replicate data that is already unavailable.

In this chapter we present All About That (AAT) our replication and preservation system for linked data. We first describe the design behind it before presenting some data on its usage. This chapter concludes with a discussion of the scalability problems

[1]http://www.archive.org
[2]http://archive-access.sourceforge.net/projects/wayback/

inherent in such approaches and explains why we consider this approach to not be truly viable for providing link integrity to the Semantic Web.

5.1 All About That

As we saw with Memento (Van de Sompel et al., 2010) replication systems can be applied to the Semantic Web succesfully, but as we discussed in Section 4.2.1.1 their approach has some issues in terms of its implementation and lack of features versus existing World Wide Web (WWW) systems like RepWeb (Veiga and Ferreira, 2003). Since getting data publishers to provide link integrity is difficult, our intention is to follow an approach more like RepWeb. Whereby the user indicates the data they are interested in and lets the system replicate it for use in the event that a link should fail. In making the user responsible for link integrity we do require that the user cares about link integrity and is prepared to invest the minimal effort required to setup such a system. Despite this effort we expect that a user who cares about link integrity will be prepared to make it. We do not expect publishers to deploy the system themselves in most cases because as we'll discuss later it requires alternative usage of computing resources that most will prefer to use for publishing data and other computing tasks.

Inspired by these existing replication and preservation approaches that had proved reasonably effective for hypermedia and the Semantic Web we designed a system called All About That (AAT) that provides replication based link integrity for the Semantic Web. Our system allows an end user to monitor and replicate a set of linked data that they are interested in. The data is preserved not at the data source but rather at a local level on the users server, and the user is able to republish this data as they desire via the system (assuming licensing of the data sources permits this). This is in line with the ideas of Veiga and Ferreira (2004) in that the end user specifies the parts of the web they want to preserve, and then the software takes care of this for them. The data must be replicated in such a way that the original data can be efficiently extracted from it and sufficient information to provide versioning over the data is kept.

Since in the semantic web domain the objects of interest are Uniform Resource Identifiers (URIs) our design revolves around the concept of preserving the profile of a URI (see Definition 5.1). As the data being processed is Resource Description Framework (RDF) it is logically divided into triples which can be preserved and monitored individually. In order to store the required level of provenance data it was deemed necessary to store information pertaining to the temporality and provenance of each triple - when it was first seen, last updated, source URIs and whether it has changed or been retracted or deleted from the RDF.

Definition 5.1. The profile of a URI is the transformed and annotated form of the linked data retrievable about a given URI such that the temporality and provenance of the triples contained therein are inferable from the profile.

A key requirement of any such replication system is that it must monitor the original data source over time updating the profiles as necessary. This will then allow the system to fulfil the core requirement of providing the user with the ability to use a local copy of the data in the event that links to the original copy stop functioning. In monitoring the data over time such a system will additionally have the ability to provide a user with specific versions of the content, unless it has been designed to store only the most recent version of the data. This functionality while not essential to providing link integrity is a useful added feature which is important for some applications such as data provenance e.g. determining when and where data came from.

One problem to address with regards to the implementation of such a system is how you store the data, particularly if you are updating it frequently. The naive approach is just to store the whole of the data each time but this obviously can become extremely costly in terms of storage. A more sensible approach, and the one we take in our system, is to use a version control system approach where you store a base version of the data and then store deltas which describe the changes that have been been detected over time.

In terms of user interface it was deemed desirable to provide a human friendly means for a user to view a profile both in the stored form and in its original form, but the primary goal was to provide a Hypertext Transfer Protocol (HTTP) API so that client programs could retrieve the data directly. Since a URI profile will contain versioning information the interface should allow a user to view a particular version of the profile.

5.1.1 Design and Implementation

The design of AAT can be divided into three main parts:

1. Schema - The schema defines a RDF vocabulary which is used to store the replicated data within AAT.
2. Data Retrieval - This is the core of the system and is responsible for the replication of data.
3. Web Interface - This provides the ability for a user to interact with the system and also provides a HTTP API which clients can use to access the replicated data.

Firstly we describe the design of these parts before discussing the overall architecture of the full system. The initial design for the system was presented in a poster (Vesse et al., 2009), the design described here represents a later iteration of the system (Vesse et al., 2010). Finally we present data gathered during testing of the system and discuss briefly the feasibility of this approach.

5.1.1.1 Schema

As the first stage of implementation a RDF schema for AAT[3] was defined that embodies classes and properties which allow the description and annotation of triples. The properties were chosen to encode all the provenance and temporarility information we required in order to do versioning over the data, and be able to reconstruct the latest or a specific version of the data at any time.

Primarily the schema defines a class for representing profiles called `aat:Profile` and uses the `rdf:Statement` class to represent triples. `rdf:Statement` is used as the basis of triple storage as it makes it possible for non-AAT aware tools to extract the original triples from the profile RDF if needed without awareness of the AAT schema. A number of properties are defined which store meta data about the profile itself such as created date, updated date and source URI. Similarly properties are defined for triples which allow the first and last asserted dates, source URIs and change status of a triple to be indicated. A key distinction in the schema is between the properties `aat:profileSource` and `aat:source` which are used to store the source URI of profiles and triples respectively. Despite storing equivalent data two properties are created since the former expresses the single URI which is the starting point for the profile while the latter expresses all the URIs at which a given triple is asserted.

[3]This schema is available at `http://www.dotnetrdf.org/AllAboutThat/`

While there were alternative schemas and vocabularies available that could have potentially been used to store the required data the motivation behind designing our own schema was to provide a lightweight schema that attached all data to a single subject for ease of processing. Alternatives such as the Provenance Vocabulary (Hartig and Zhao, 2009) are far more expressive, but they potentially require introducing multiple intermediate blank nodes which would significantly complicate the processing needed to implement many of the core features of AAT. Similarly the Open Provenance Model (Moreau et al., 2007) is highly expressive but like Hartig & Zhao's vocabulary the RDF serialization is overly complex for use in AAT. As discussed in Section 8.3.1 there is no reason why the data contained in AAT could not be exposed in other provenance vocabularies in the future, but for AATs processing and storage a lightweight vocabulary is preferable.

5.1.1.2 Use of RDF Reification

The use of reification was chosen, perhaps controversially so, over the use of named graphs primarily due to the need to make annotations at the level of individual triples rather than at the graph level. This choice was motivated by the fact that the mechanism provides a clear and obvious schema for encoding a triple and adding additional annotations to it. While reification may significantly increase the size of the data being stored initially, over time this balances out compared to named graphs where it is necessary to either store many copies of the same graph or store multiple named graphs which represent a series of deltas to the original data. Interestingly by encoding the annotations at the triple level we actually are able to reconstruct the equivalent deltas anyway when necessary, yet still minimise our storage requirements.

The other difficulty inherent in the named graphs approach is that the annotations typically would then be held separately in other named graphs which adds to the complexity of the data processing. Nevertheless named graphs are used within AAT since each profile naturally forms a named graph and AAT generates several related named graphs about each profile, these detail change history and changesets as described in the next section.

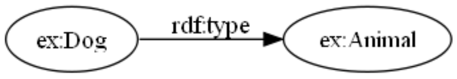


Figure 5.1: Original Triple

5.1.1.3 Data Retrieval

Data retrieval is a core part of the AAT system since it is responsible for retrieving the data the user wants to preserve and then monitoring it over time. The first step in this process is for the user to specify a set of URIs that they are interested in and the system will then create a profile for each URI. To do this we need to obtain RDF data about the URIs using a data retrieval algorithm. In our initial prototypes this algorithm was very simple and only retrieved the RDF from the URI the user wished to preserve i.e. it simply dereferenced each URI of interest. In later versions we replaced this with the use of the expansion algorithm which is detailed in Chapter 6.

Once the RDF data is retrieved it is transformed into an alternative RDF representation using the AAT schema. Each triple in the retrieved data can be transformed into a set of triples which represent it and the relevant provenance and temporality information. For each triple in the original RDF a blank node is created which is used as the subject of the set of triples which represent it in the AAT schema. Figure 5.1 shows an example triple and Figure 5.2 shows it transformed into the AAT form. A URI's profile consists of a set of these transformed triples, and each profile of a URI is stored as named graph in a triple store. Note that the system is entirely store agnostic though as is discussed later in this chapter we found some stores were better suited as the backend for this than others.

After the initial profile creation necessary to replicate the data of interest the system must then monitor the data over time in order to ensure the replicated version is always as up to date as possible. To do this the system uses a process installed as an operating system service which periodically performs a batch update of all the URI profiles. The update frequency can be configured by the user and the chosen setting will generally depend on the data being monitored. For example, if the data changes quickly then it may be desireable to carry out daily updates, however if it changes more slowly a weekly or monthly update frequency will likely be more appropriate. Obviously the more frequently the user chooses to update their replicated data the more computing

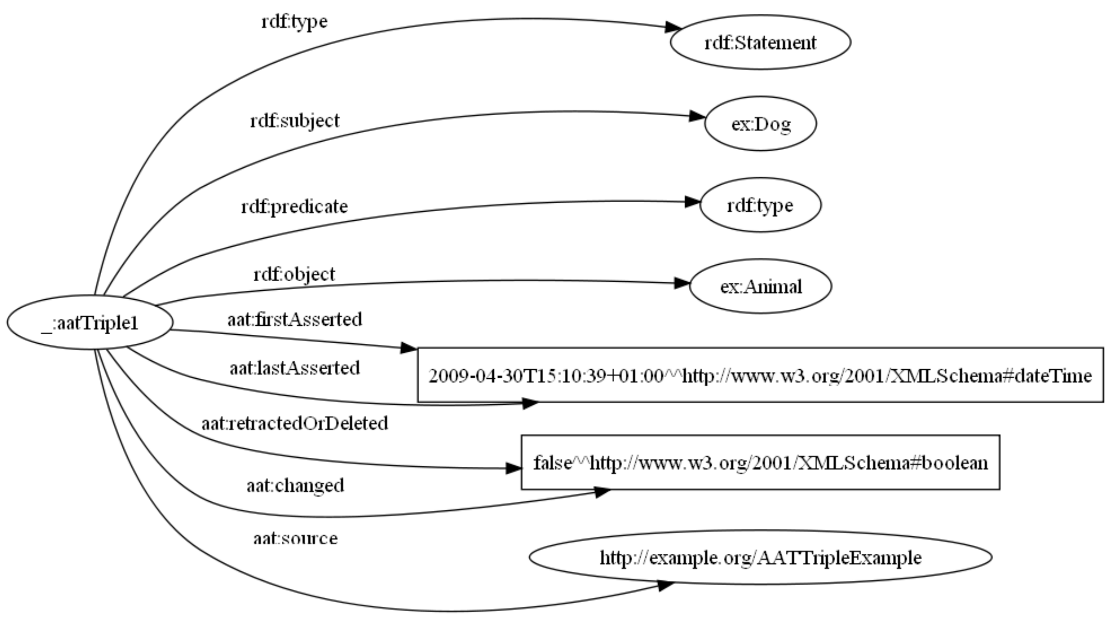


Figure 5.2: Triple transformed to AAT Annotated Form

time and resources will be required i.e. there is a trade off between ensuring data is as up to date as possible against use of resources in processing and storing the data.

The actual update process for a URI's profile is broadly similar to the creation process in that firstly RDF data is retrieved using the data retrieval algorithm. The main difference in carrying out an update is that rather than directly transforming the retrieved triples to the annotated form they are compared with the existing annotated form and wherever applicable the existing annotations are updated rather than new ones being added. For example consider figures 5.1 and 5.2 again, if you were to update such a profile and saw that the triple in figure 5.1 had not changed then you would only need to update the value given for the `aat:lastAsserted` property in figure 5.2.

In comparing the latest version of the data with our stored data we look for four types of changes using simple computations over the data (see Definitions 5.2-5.5). A distinction is made between missing knowledge and retracted or deleted knowledge as it may be possible for triples to be perceived to be temporarily non-present in the RDF, for example in the event of a transient network issue making some or all of the relevant URIs unretrievable. In this case the updated date for the profile will still be updated leaving many of the triples in the profile to appear missing. This is important as it echoes the notion of *intermittance* as defined by Koehler (2002) i.e. we should not assume a URI is broken just because it fails to respond to one attempt to access it.

The length of time we require triples to be missing before we consider them to be deleted is currently set to 7 days by default. This time period is a user configurable parameter that can be adjusted depending on the data that is being monitored. In some scenarios you may wish to have no notion of missing knowledge in which case this parameter can be configured to be zero so as soon as a triple is not seen it will be considered retracted or deleted.

We chose to combine retracted and deleted knowledge into one definition since upon retrieving the data and comparing it with the existing data we have no way of distinguishing between the two things. Without explicit statements from the data publisher there is no way to know whether the triples in question were simply deleted from the data for whatever reason, or if they were retracted in the strict semantic sense i.e. the publisher logically retracts the facts stated in those triples.

Definition 5.2. New knowledge is any triple that is new to the RDF for the profiled URI.

Definition 5.3. Changed knowledge is any triple where the object of the triple has changed. Only triples where the predicate has a cardinality of 1 can be considered to change.

Definition 5.4. Missing knowledge is any triple no longer found in the RDF for the profiled URI but which was recently seen in the RDF.

Definition 5.5. Retracted or deleted knowledge is any triple no longer found in the RDF for the profiled URI which has not been seen for a reasonable length of time.

In regards to the concept of changed knowledge consider some arbitrary predicates `ex:one` and `ex:many` which have cardinalities of 1 and unrestricted respectively. Since `ex:one` has a cardinality of 1 it can be said whenever the object of that triple has changed it is changed knowledge. Yet it cannot be said for `ex:many` triples as the predicate has unrestricted cardinality, therefore each triple using this predicate must be treated as a unique entity i.e. one instance of a triple using this predicate cannot be considered to replace another. In the examples the fact that $< A >$ was related to $< C >$ via the predicate `ex:many` in Example 5.1 and now is instead related to $< E >$ in Example 5.2 doesn't mean they are related to $< E >$ instead of $< C >$, it just means they no longer consider themselves related to $< C >$. The fact that they are related to $< E >$ is new knowledge while the fact they related to $< C >$ is missing/deleted knowledge, but if the value of the `ex:one` relationship had changed then that would be considered changed knowledge.

Example 5.1 Original Graph

```
<A> ex:one <B> .
<A> ex:many <C> .
<A> ex:many <D> .
```

Example 5.2 Modified Graph

```
<A> ex:one <B> .
<A> ex:many <D> .
<A> ex:many <E> .
```

The side effect of updating a profile is that we generate a change report, which states the exact changes we have detected between the most recent replicated version and the latest version we have just retrieved. When a change report is computed it itself is serialized into an additional RDF graph using the Talis Changeset ontology (Davis and Tunnicliffe, 2007), which is stored as another named graph in the triple store. Each changeset generated links back to the previous changeset (if one exists) such that an end user or client application consuming the data can follow the history of changes. To help with this a special URI which retrieves the most recent changeset is provided such that users have a starting point for such navigations. Separate to changesets a further named graph containing a history for each profile is also stored and this links to all the relevant changesets for a profile. Finally at the end of a batch update run we create a change report graph which links to the changesets for all profiles which were detected to have changed during the update run. Again this is stored as its own named graph and links back to the preceding change report (if any) to allow clients to follow the change history.

5.1.1.4 Web Interface

The Web interface of AAT is the user facing part of the system and is designed partly to allow humans to browse and manage the replicated data, but also to allow clients to access the replicated data via a HTTP API. It is implemented as a web application running on a standard web server which allows it to easily provide a traditional Hypertext Markup Language (HTML) based interface for humans and the HTTP API for machines.

From a human point of view the interface allows users to explore the data by first selecting a profile to view and then choosing from various actions related to that profile such as viewing it, exporting it, retrieving a specific version etc. A user may also use the interface to add new URIs they wish to monitor to the system and to initiate unscheduled updates to profiles. A sample screen shot of the human interface can be seen in Figure 5.3 which shows the interface for reviewing a change report on a particular URI. Additionally in the screen shot you'll see that a human user has access to a variety of links which lead to both human readable representations of the replicated data in the system and the machine readable RDF data.

In order to provide arbitrary client access to the replicated data we followed linked data best practices (Bizer et al., 2007) and minted multiple dereferenceable URIs for each profile, which are served to clients via the web application. These allow the retrieval of the profile contents which consists of all the triples ever retrieved from the profile URI in the transformed form, the export of the profile (see Definition 5.6) and various meta graphs about a profile e.g. change history, changesets etc. This means that the profile of a URI has a URI and thus can itself be profiled if it was desired though we doubt that this would ever be a useful or sensible thing to do. By providing relevant linked data through our Web application clients can consume the replicated data with only minimal knowledge of the actual system i.e. as long as they understand how to dereference a URI to retrieve RDF they can access the data.

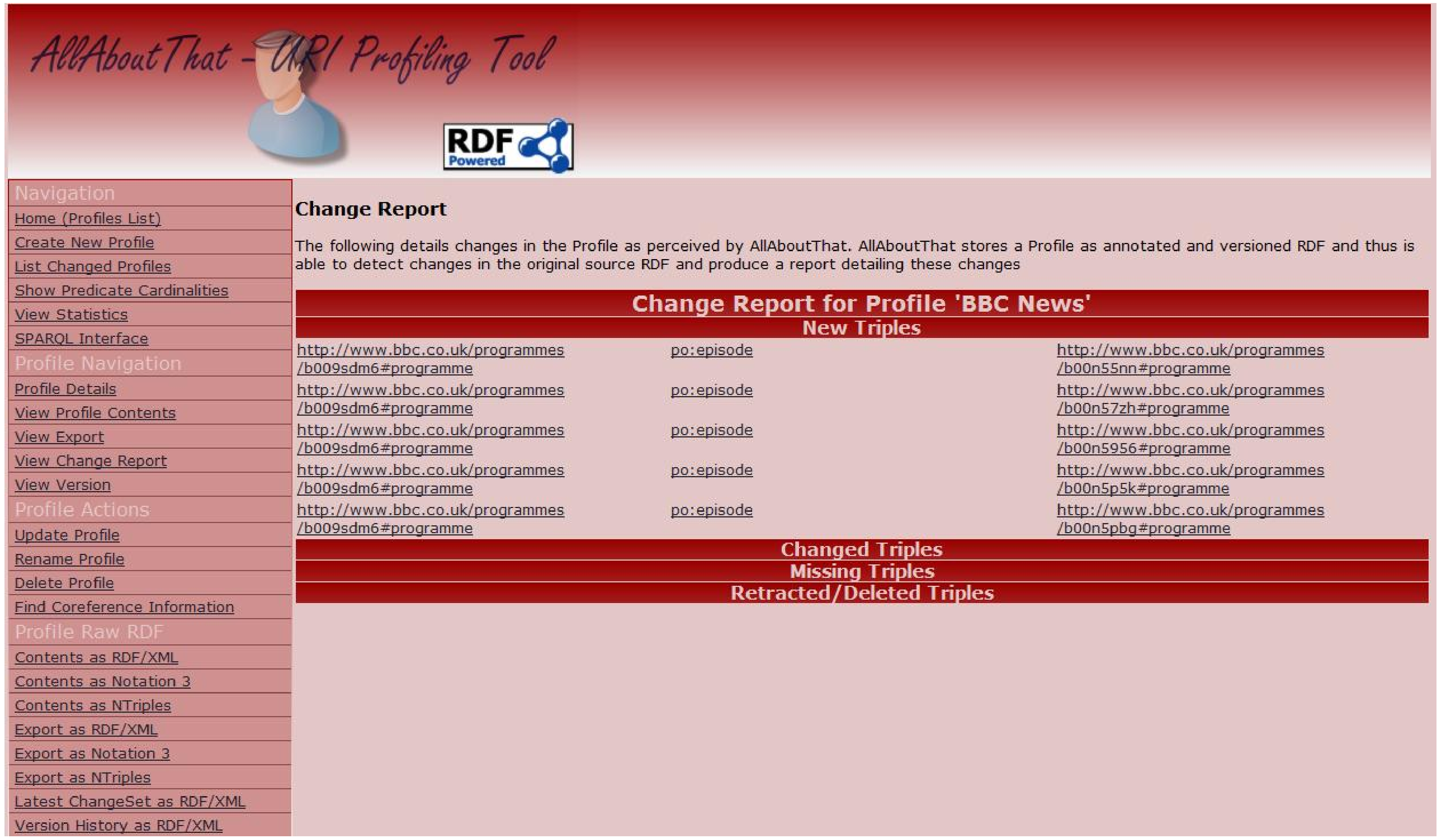
Definition 5.6. The export of a profile is the result of taking the transformed form of the data that AAT stores and converting it back into the original RDF. An export only uses the data that was calculated to be present when the most recent update to the profile was conducted i.e. old data is not exported.

The full HTTP API for arbitrary clients consists of the following set of URIs. Note that all URIs given here are relative to the base URI of an installation of AAT since you may have multiple instances of the system running on a single server each exposed via a different base URI.

- `/profiles/` - Retrieves the RDF which list all available profiles i.e. links to all the replicated data in the system.

- `/profiles/{profileID}` - Retrieves the RDF for a profile. This is the data stored using the AAT schema detailed in Section 5.1.1.1 plus metadata about the replicated data.
- `/profiles/{profileID}/export` - Retrieves the export of a profile (see Definition 5.6).
- `/profiles/{profileID}/history` - Retrieves the version history of a profile. The version history includes links to all changesets for a profile.
- `/profiles/{profileID}/changes/{changesetID}` - Retrieves a specific changeset for a profile. A changeset represents a set of changes to a profile calculated when a update to the profile was made.
- `/profiles/{profileID}/recentchanges` - A special URI which redirects to the most recent changeset for a profile.
- `/profiles/{profileID}/version/{versionDateTime}` - Retrieves a version of the replicated data for a profile transformed back into the original RDF which contains only the data known to have been present at a specific date and time.
- `/changes/{reportID}` - Retrieves a change report which links to all profiles which were calculated to have changed (and thus have changesets) in the most recent batch update.
- `/recentchanges` - A special URI which redirects to the most recent change report for the system.

With this set of URIs it should be apparent that a client can easily access all the data stored within an AAT instance and use standard follow your nose link navigation to discover additional data about any data it retrieves from the system. Again following linked data best practises set down in Bizer et al. (2007) all the above URIs support proper content negotiation so clients can retrieve the data in a RDF serialization they understand.



Navigation
Home (Profiles List)
Create New Profile
List Changed Profiles
Show Predicate Cardinalities
View Statistics
SPARQL Interface
Profile Navigation
Profile Details
View Profile Contents
View Export
View Change Report
View Version
Profile Actions
Update Profile
Rename Profile
Delete Profile
Find Coreference Information
Profile Raw RDF
Contents as RDF/XML
Contents as Notation 3
Contents as NTriples
Export as RDF/XML
Export as Notation 3
Export as NTriples
Latest ChangeSet as RDF/XML
Version History as RDF/XML

Change Report

The following details changes in the Profile as perceived by AllAboutThat. AllAboutThat stores a Profile as annotated and versioned RDF and thus is able to detect changes in the original source RDF and produce a report detailing these changes

Change Report for Profile 'BBC News'		
New Triples		
http://www.bbc.co.uk/programmes/b009sdm6#programme	po:episode	http://www.bbc.co.uk/programmes/b00n55nn#programme
http://www.bbc.co.uk/programmes/b009sdm6#programme	po:episode	http://www.bbc.co.uk/programmes/b00n57zh#programme
http://www.bbc.co.uk/programmes/b009sdm6#programme	po:episode	http://www.bbc.co.uk/programmes/b00n5956#programme
http://www.bbc.co.uk/programmes/b009sdm6#programme	po:episode	http://www.bbc.co.uk/programmes/b00n5p5k#programme
http://www.bbc.co.uk/programmes/b009sdm6#programme	po:episode	http://www.bbc.co.uk/programmes/b00n5pbg#programme
Changed Triples		
Missing Triples		
Retracted/Deleted Triples		

Figure 5.3: Example page from the All About That Web Interface

5.1.2 Architecture

The components of AAT described in the previous sections are combined together as shown in Figure 5.4. As already noted AAT is designed to be agnostic of its underlying storage though in practise differences in implementation between triple stores mean only certain stores were found to be viable for use as the backing store. In the early prototyping stage a simple Relational Database Management System (RDBMS) based store was used which was sufficient for initial prototyping but not scalable for real world testing, so then the usage of production grade triple stores was adopted. Initially it was intended to use the open source release of Virtuoso[4] as the backing store but it was found that Virtuoso didn't correctly preserve boolean typed literals which created issues in the internal processing of data within AAT. 4store[5] was then used briefly but it was found that it was unable to handle the heavy volume of parallel read or writes which AAT uses during its data processing due to 4store's concurrency model. For most of our actual testing we ran it against AllegroGraph[6] since it demonstrated the ability to handle the high volumes of read or writes necessary for using AAT on large datasets.

As can be seen in Figure 5.4 various components of the system are intended to be switchable such as the triple store as detailed in the preceding paragraph. Probably the most influential switchable component is the data retrieval algorithm which is used to retrieve the data for a URI in order to create or update a profile. This is a key component since depending on the algorithm used the amount of data that gets replicated may vary massively. For example, as discussed in the earlier Section 5.1.1.3, in early prototypes we used a very simple algorithm which just dereferences the starting URI which tends to retrieve relatively small amounts of data. In later iterations of the system we replaced this with our expansion algorithm which we discuss in the subsequent chapter, depending on the URI this can produce massive amounts of data, in some cases into the 100,000s of triples.

[4]`http://www.openlinksw.com/virtuoso`
[5]`http://4store.org`
[6]`http://www.franz.com/agraph/allegrograph/`

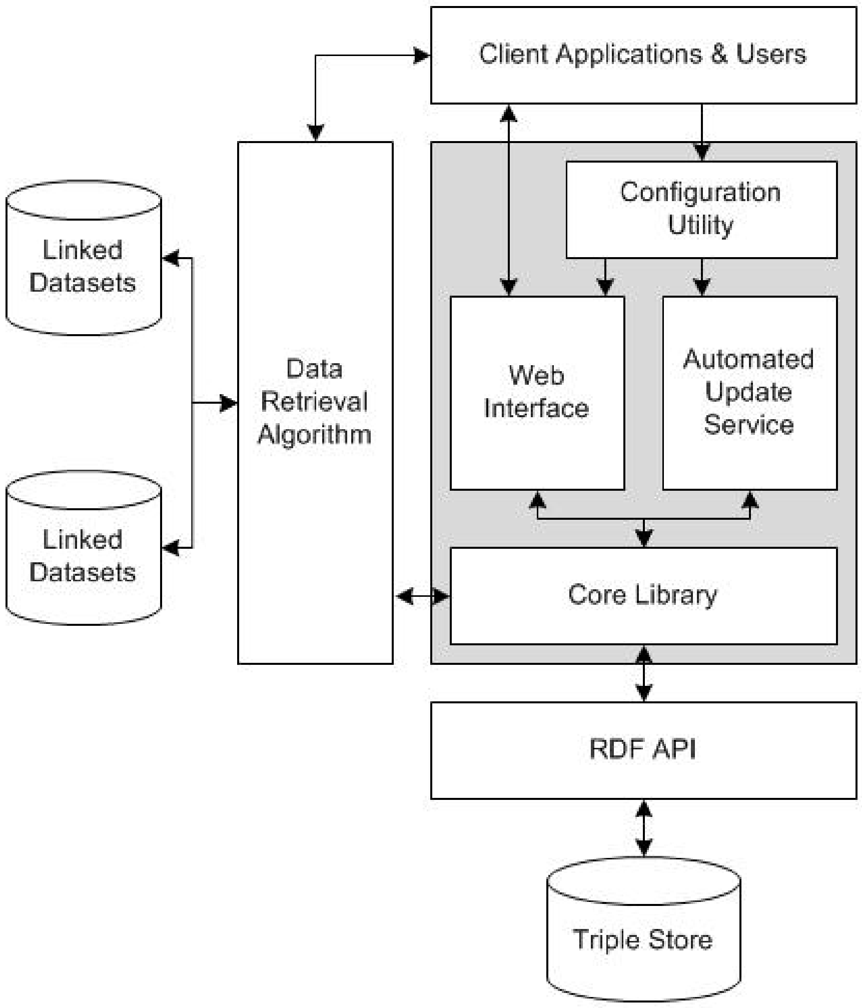


Figure 5.4: All About That Architecture

5.2 Evaluation

Generally speaking one advantage that replication based systems have is that in order to provide link integrity they need only successfully replicate the data and provide access to it. This basic behaviour is quick and easy to test but does not tell us anything about the real world viability and scalability of the system. In order to evaluate whether the system we had developed was viable in these regards we need to test how it would perform under the intended usage scenario i.e. that of monitoring a specific dataset over a period of time.

So to test our system we chose a dataset that we were familiar with and which we knew changed rapidly. This was the British Broadcasting Corporation (BBC) Programmes dataset[7] which is a dataset published by the BBC detailing all the TV and radio programs they broadcast. This includes detailed information such as individual episodes of programs and times of broadcast which are frequently updated in the dataset. Given that the dataset changes continuously we decided to monitor it for a short period of a week with a daily update frequency. The purpose of this observation was to determine whether the system was capable of monitoring the dataset at that frequency and how many changes we would actually observe. As the complete dataset is very large we limited the dataset for our testing to a subset of URIs which represented all the TV programs broadcast on the flagship channel BBC 1. This channel was chosen because this is the channel which broadcasts programs that are likely to be most frequently updated in the dataset such as daily soaps, news broadcasts etc.

For this experiment we used a machine that is equivalent to current consumer grade desktop machines with a quad core processor, 4GB RAM and standard SATA hard disks. We chose to test on a consumer grade machine as we wanted our approach to be accessible to all users of the semantic web and we felt that we could not consider our approach viable if the computing power required was not available to the average user. It is likely that better performance could be obtained by running this system on server grade machines and we envisage that most serious users of any system similar to ours would indeed do so.

The results of this brief experiment are given in Table 5.1. On the first day we see many changes in the profiles since entirely new data has been imported into the system, on subsequent days the average number of changes is seen to be 2. This is due to the fact that the typical update we see the BBC make to their data is that they add a triple describing a newly broadcast episode of a programme, and update the value of the `dc:modified` triple for that programmes RDF. The apparently high figure of 25 for max changes was actually due to the URI of one program consistently failing to resolve on subsequent days resulting in the system correctly reporting the triples to be missing each day. In general the relatively high number of profiles changing each day is due in part to many of the programs being broadcast daily and in part to the BBC automatically generating some of their data so things like modified dates change regularly.

For the actual dataset size we found that after the initial import the data totalled around

[7]http://www.bbc.co.uk/programmes/developers

1 million triples and that after a week of running the system this had risen to around 1.3 million. This implies that for a rapidly changing dataset the size of the data to be stored increases quite quickly even using storage techniques like ours that effectively only store deltas once the initial data replication has taken place.

Table 5.1: BBC Programmes Dataset replication statistics for 1 week

Date and Time	Number of Changed Profiles	Average Changes per Profile	Max. Changes	Min. Changes
12/2/2010	163	43	311	1
13/2/2010	105	2	25	1
14/2/2010	90	2	25	1
15/2/2010	87	2	25	1
16/2/2010	90	2	25	1
17/2/2010	87	2	25	1
18/2/2010	86	2	25	1

Note that this was not the only experimental run that we carried out during our development of this sytem. We carried out other runs of longer durations (1-2 months) during the prototyping stage but are unable to present meaningful results from those since many of them revealed issues in our implementation which made the results invalid. We chose not to conduct further long runs after this short run proved that the system was running correctly since as we discuss in the subsequent section we did not feel the performance was sufficient for a number of reasons.

5.2.1 Performance and Scalability

In terms of scalability, on paper the majority of algorithms in AAT need to run on a single thread for each profile, but it is trivial to process multiple profiles in parallel and this is the approach taken in the system. Since work can be divided over multiple threads it is possible to increase the scalability by dividing the work over a cluster of machines which would allow much larger datasets to be monitored efficiently. Despite this the system will only scale at best linearly as more threads, processors and machines are added. This poses an obvious issue in that the amount of data you can replicate and monitor is limited directly by the amount of computing resources you have. As the results in the previous section have shown the amount of data you can monitor was relatively small given our test system whose specifications are comparable to today's typical consumer level desktop hardware. This poses a problem in that such a solution is never going to scale to web scale because the computing resources required would be staggeringly massive.

This is the key disadvantage of any replication based system including our own, the resources required to replicate and store the data can be surprisingly large even for relatively small datasets. We observed that to just monitor the couple of hundred or so URIs that made up our dataset and run the batch update process could easily take several hours on our machine. Therefore one can extrapolate that monitoring a dataset consisting of thousands or tens of thousands of URIs could realistically take a number of days.

As far as specific bottlenecks go we observed two main issues in our implementation which we would expect to apply to any similar systems one could develop:

1. Data Retrieval overhead.

2. Triple Store performance.

5.2.1.1 Data Retrieval Overhead

Any such system needs to retrieve the original data from the Web using some retrieval algorithm. Whether this is a simply dereferencing a URI or a more complex algorithm like the expansion algorithm we present in Chapter 6 the data has to be retrieved in real time. The amount of time taken is dependent on a number of factors from fundamental physical overheads such as network latency to social overheads such as Web crawling etiquette.

While the typical overhead may only be a few seconds this quickly mounts up as the amount of URIs you are replicating and monitoring increases. While some of this overhead is mitigable by parallelising the algorithms and processes involved, other parts such as the social overheads cannot be avoided. If there are vast numbers of crawlers running in parallel it is inadvisable to have too many accessing one web server simultaneously as that will look like a denial of service (DOS) attack to the server administrators and the crawlers may be blocked from accessing those servers. Once a crawler is blocked it can no longer replicate and monitor the desired data properly. As a consequence the system is rendered useless for maintaining link integrity for that data and is in effect useless to the user.

This social overhead is also the reason that a system can scale at best linearly as we implied earlier. Even if a system has access to an infinite amount of computing resources the overheads of respecting crawling etiquette will render most of the resources unusable if all the data that needs replicating is from one server. It is this overhead which was the main factor in the time taken to update the data in our testing, as all the data originated from one source our system had to be careful to respect crawling etiquette resulting in it taking several hours just to update a couple of hundred profiles.

5.2.1.2 Triple Store Performance

Another major performance bottleneck in our system actually proved to be the Input/Output (IO) with the triple store. Due to the way our system functions each profile must be loaded from the store into memory where the calculations of changes against the retrieved data takes place. Once done the profile is then written back to the store plus a variety of meta graphs are written/updated back to the store. Most triple stores are not designed to cope with this style of usage, many have good bulk import or update capabilities but perform quite poorly at doing lots of small updates. As already mentioned in Section 5.1.2 we had issues in finding a suitable triple store, even our choice AllegroGraph was not without issues. We found that AllegroGraph 3.2 which we used was very RAM hungy and in a single batch update run would consume the majority of our machines free RAM and requiring us to reboot it daily after each run. Also it did not automatically index new triples meaning that after each reboot we would then have to manually invoke its indexer.

These issues meant that the system was not as easy to run as we had hoped since it required plenty of manual maintenance which is clearly not ideal. One of the aims of any link integrity system must be ease of use as otherwise everyday users cannot make use of your system and it will not be a viable solution.

5.3 Conclusion

In this chapter we have presented a replication based approach to link integrity for the Semantic Web and demonstrated that our system (AAT) is capable of providing link integrity. Unfortunately as alluded to in the preceding section on performance and scalability (see Section 5.2.1) our testing also served to demonstrate that AAT was not sufficiently viable or scalable in terms of the evaluation criteria in relation to part 2 of our hypothesis (see Section 1.1).

Though AAT shows that replication and preservation based approaches can provide link integrity for the Semantic Web, and therefore proving part 1 of our hypothesis, it is clearly not a viable approach. To start with the amount of computing resources required to monitor small datasets of just a few hundred URIs was surprisingly large. Even set to run overnight when the machine was not otherwise used our software would monopolise available system resources and leave the machine sluggish and unusable in the morning until manual reboots of the backing triple store were carried out. While this is arguably in part due to the choice of triple store as we detailed in the previous section the true overheads in our system are mostly around data retrieval.

Data retrieval whether simply dereferencing a URI as in early prototypes of AAT or using more complex crawling style behaviour as in later versions has to adhere to social etiquettes. It is considered bad behaviour to make requests too frequently, or to make many requests simultaneously to a server and data retrieval algorithms must adhere to these restrictions. As we noted earlier if a system fails to respect this etiquette it may be blocked from accessing a server completely, this would make the system incapable of actually replicating data and thus providing link integrity. Therefore a system cannot ignore this overhead and in some cases this results in adding additional overhead into the system in order to respect these etiquettes.

Due to this the scalability of such approaches is rather limited, however many threads, processes and machines you use, you can typically scale such a system linearly at best. The only exception to this would be the case where you were monitoring data from a sufficiently varied number of servers that the social overhead was minimalised or even eliminated. Realistically though this is not a scenario such a system is likely to encounter as the typical usage scenario for any such system is that there is a particular dataset of interest which a user wishes to replicate. Thus in most cases all the URIs will come from one server and these overheads will be unavoidable.

Returning to the notion of viability (stated in part 2 of our hypothesis) that in order for us to consider a system viable it must require minimal user involvement. Clearly we have managed this in terms of the initial set up of the system and in allowing the user to access the replicated data through the web interface with both requiring only a few clicks. Unfortunately we did not manage this in the maintainability of our system, as we noted we had to regularly reboot the backing triple store and run manual indexing jobs on the store. These issues could probably be alleviated by switching to a different triple store but at the time of our experiments our chosen store AllegroGraph was actually the most usable as the back end of our system. We could argue that this is somewhat indicative of the relative immaturity of triple stores compared with traditional RDBMS systems but the problem is more that our processes use the triple stores in ways they were not really optimised for. In this assertion we are referring to the fact that our batch update process typically reads large portions of the existing data into memory - in fact the majority of the store - before overwriting or updating it in the store. While vendors put plenty of effort into IO performance it is often aimed at bulk IO operations or query operations, not the vast quantity of small IO operations that AAT uses.

For the reasons outlined here we must conclude that such systems are not truly viable for providing link integrity at scale. They clearly work and are a relatively low cost and complexity solution to the problem but they just don't have the scalability needed to provide link integrity to broad swathes of the Semantic Web. The viability issues we have outlined are all solvable given time and further research but we don't believe the scalability issues are easily addressed. There are a few cases where it would be less of an issue e.g. replicating data from many disparate sources but this doesn't really fit with the typical use case we envisage for such systems. Our results are very much inline with those from hypermedia research such as Moreau and Gray (1998) or Veiga and Ferreira (2003, 2004). These showed that other similar systems worked for hypermedia but that they were similarly limited in their viability by the need for user involvement, and their scalability by the resources required and data retrieval overheads that we have encountered with our own system.

As a result of this lack of scalability in the next Chapter we outline an alternative recovery based approach to link integrity for the Semantic Web which should be much more scalable.

5.3.1 Alternatives to Replication and Preservation

As we stated in our summary of approaches to link integrity back in Section 4.3 replication is not the only approach that we could have taken. As we have seen in this chapter the replication approach has a number of problems that don't make it a truly viable or scalable solution to link integrity. In the next chapter we look at the recovery approach which is a promising alternative to replication.

Chapter 6

Recovery

Recovery based approaches are an alternative approach to providing link integrity which focus on fixing broken links Just-in-Time (JIT) when they actually fail. The motivation behind this is that it requires minimal preparation on the part of the user or client ahead of the actual failure occuring. Typically a client need only know that there is a recovery service available to use and fall back to using that in the event of a link failure. This approach originates from the work of Phelps and Wilensky (2004) which demonstrated that you could compute a lexical signature of a page and use this to relocate it in the event that it moved. The disadvantage of their approach was it required you to precompute these and alter existing links to include them. Obviously this was not a practical approach that would scale on the real web since you had to convince everyone to change their links.

Harrison and Nelson (2006) later provided an important extension to Phelps and Wilenksy's work by demonstrating that the signatures could be computed JIT using secondary data sources like search engine caches. This is a workable system since it doesn't require link publishers to include any additional information in their links and exploits existing data already available on the Web. As seen in the introduction with the linked data cloud (see Figure 2.5) there are already a multitude of data sources on the Semantic Web so it should be possible to exploit these in a Semantic Web recovery approach.

6.1 Expansion Algorithm

Based on the approach taken by Harrison and Nelson (2006) we designed an algorithm which we call the expansion algorithm. It exploits the interlinked nature of the Semantic Web and the multitude of datasets available to effect Resource Description Framework (RDF) data retrieval and recovery. This is designed to be embedded into existing clients and also easily exposed via a web service if desired. This allows clients that need to guarantee data retrieval to do so and provide the algorithm to clients that don't want to embed it directly.

6.1.1 Design

We call our algorithm the Expansion Algorithm since its aim is to expand upon a given URI using as many data sources as it can discover to return RDF data it considers relevant. Our approach is based upon exploiting both the interlinking between data and arbitrary query services. By this we mean that we both follow links in the data we retrieve and make use of the query services associated with various datasets to augment our results.

The design of the algorithm was inspired in part by the linked open data cloud diagram presented earlier in Figure 2.5[1]. When designing this algorithm we were thinking about what would you do if a major hub of the linked data movement like DBPedia[2] were to dissappear? Figure 6.1 shows the section of the cloud diagram that surrounds DBPedia. Note that there are a large number of data sources that link back into it. We postulated that if we could exploit these by discovering the data from the other sources automatically that we could provide effective data recovery in the event of a link failure. Since most data sources provide some form of query service our basic idea was to combine the use of these services with services that allowed us to automatically discover these interlinkings.

Firstly to exploit the explicit interlinking between datasets we start by using a simple web crawling approach, as its most basic the algorithm simply attempts to retrieve RDF data starting with the initial Uniform Resource Identifier (URI) and then follows links

[1]Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

[2]http://dbpedia.org

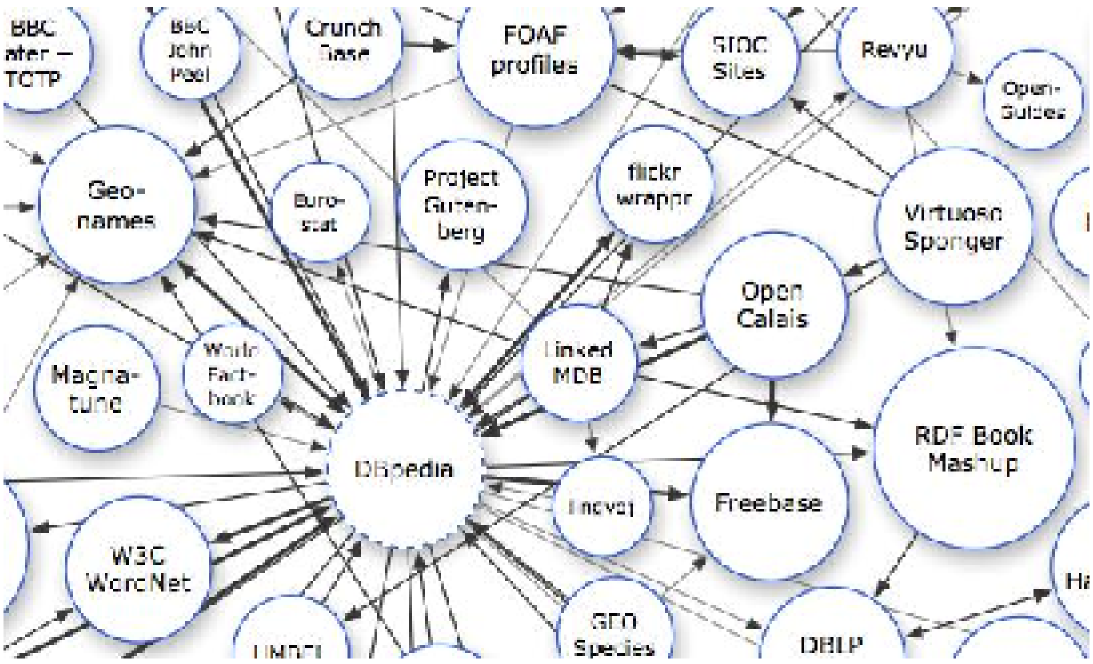


Figure 6.1: What if DBPedia were to dissappear?

in that data. Our crawling strategy is breadth-first and depth-limited, the default depth limit is 1 but may be configured to higher values when necessary. A pseuo-code style outline of the processing model of the algorithm can be seen in Algorithm 1.

The types of links (i.e. predicates) followed are fully configurable but by default it just follows `rdfs:seeAlso` and `owl:sameAs` links. These defaults were chosen because they are typically used in RDF to express additional sources of equivalent or related data.

Since predicates in RDF statements may indicate links in either or both directions (i.e. Subject → Object, Object → Subject and Subject ↔ Object) the algorithm is fully capable of differentiating between these based on its configuration. Any link type can be configured so that it is followed only in the desired directions. In the aforementioned default configuration `owl:sameAs` links are followed in either direction while `rdfs:seeAlso` are followed from Subject → Object only.

Our link following behaviour is designed to be more intelligent than the average RDF crawler in that we further restrict which links we actually follow. The algorithm only follows links where the origin of the link - either the Subject or Object depending on the direction of the link - is a URI equal to the current RDF documents origin URI. By doing this we avoid following irrelevant links in the case where we get a RDF document which makes statements about multiple things. To understand this restriction consider Listing

Algorithm 1 Expansion Algorithm

```
Require: URI, Expansion Profile
 1: ToExpand as a set of pairs of URIs and Depths
 2: while ToExpand $\neq \emptyset$ do
 3:    Remove first pair from ToExpand
 4:    if Graph with URI is already in the Dataset then
 5:       Continue
 6:    end if
 7:    if Depth > Max Depth then
 8:       Continue
 9:    end if
10:    Retrieve the Graph at the URI
11:    Add the Graph to the Dataset
12:    for all Triples in Graph do
13:       if Triple is a Link then
14:          Add a new pair to ToExpand
15:       end if
16:    end for
17:    for all Datasets in Expansion Profile do
18:       if Dataset has a SPARQL Endpoint then
19:          Issue a DESCRIBE for the URI against the Endpoint
20:          Add resulting Graph to the Dataset
21:          Process the Graph for additional Links
22:       end if
23:       if Dataset has a Lookup Endpoint then
24:          Issue a Lookup for the URI against the Endpoint
25:          Add resulting Graph to the Dataset
26:          Process the Graph for additional Links
27:       end if
28:       if Dataset has a Discovery Endpoint then
29:          Issue a Discovery for the URI against the Endpoint
30:          for all Equivalent URIs do
31:             Add a new pair to ToExpand
32:          end for
33:       end if
34:    end for
35: end while
36: return Dataset
```

```
@base <http://example.org/resource/Something> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex: <http://example.org/resource/> .
@prefix else: <http://elsewhere.org/resource/> .

ex:Something rdfs:seeAlso else:Something .
ex:SomethingElse rdfs:seeAlso else:SomethingElse .
```

Listing 6.1: Link Following Restriction Example

6.1, in the example the document has the URI `http://example.org/resource/Southampton` and contains two RDF triples. The crawler would only follow the link to `else:Something` since the Subject of that link is equal to the URI of the document and would ignore the link to `else:SomethingElse`.

Secondly we augment our link following behaviour by adding the ability to make requests to three forms of query service. The aim of these requests is to discover the implicit and externally asserted interlinkings which are not directly in the data retrieved from each URI itself. The forms of query services used are as follows:

1. **SPARQL Protocol and RDF Query Language (SPARQL) Endpoints:** These are endpoints to which we submit a query of the form `DESCRIBE <uri>` where `uri` is the current URI we are retrieving data for. This gets us whatever RDF data the endpoint considers relevant to the given URI.
2. **Lookup Endpoints:** These are endpoints that given a URI returns some RDF data about the URI. These may represent simple lookup services or they may be used to point to other services which may themselves do some data retrieval or crawling.
3. **Discovery Endpoints:** These are endpoints that given a URI returns some RDF data describing equivalent URIs.

While lookup and discovery endpoints may seem quite similar they are treated quite differently within the algorithm. Data retrieved from a lookup endpoint is fully processed and retained while the data retrieved from a discovery endpoint is used only to discover further URIs to be processed. There is also an important distinction with regards to the calculated depth of links discovered from the two endpoints. Links discovered using data from lookup endpoints are considered to be at $depth + 1$ relative to the depth of the current URI being looked up. Whereas links discovered using data from a discovery endpoint are considered to be at the same depth as the current URI. The distinction is quite important since as mentioned earlier the algorithm is depth-limited. We chose to make this distinction because if, as we intended, a discovery service returns equivalent URIs then it is logical that they should be considered to be at the same depth as the current URI in the crawl.

6.1.1.1 Configuring the Algorithm

Drawing on ideas described in Vocabulary of Interlinked Datasets (VoID) (Alexander et al., 2009) about the way it can be used to direct crawlers, we chose to use VoID as the primary means of expressing the algorithms configuration. We refer to our configuration files as expansion profiles, any VoID document is automatically a valid expansion profile. We introduce a couple of additional predicates in our own namespace since we require the means to allow end users to configure how the algorithm should behave.

VoID uses the concepts of datasets and linksets, the former represent a set of data which may have SPARQL endpoint(s) and/or URI lookup endpoint(s), while the latter represent the types of interlinkings between datasets. What VoID does not have a means to express is the location of a service provided by a dataset which allows an application to retrieve URIs which are considered equivalent to a given URI - these are the discovery endpoint we described in the preceding section. Examples of existing discovery endpoints on the Semantic Web include RKBExplorer's CRSes (Jaffri et al., 2008) and sameAs.org[3].

Our other extensions to VoID allow individual datasets or linksets to be marked as ignored (the algorithm will not use them) and for the user to define the depth the algorithm should crawl to (defaults to 1). These extensions are defined in our own namespace as part of the All About That schema[4] (Vesse et al., 2010).

The importance of having an algorithm that can be configured to behave differently depending on the domain of interest should not be underestimated. In Chapter 4 we discussed a number of systems including DSNotify, Okkam and coreference resolution services (CRSs) all of which rely on domain specific knowledge. As we noted there is an issue that this need can harm the scalability of a solution since it typically requires

[3]http://www.sameas.org
[4]http://www.dotnetrdf.org/AllAboutThat/

more user involvement to configure a system appropriately. Yes as you will see in the subsequent chapter having the algorithm configured appropriately for a domain can make a significant difference in its performance.

For the experiment described in the subsequent chapter we primarily used the default expansion profile[5] for our algorithm. This profile configures the algorithm to use DBPedia[6] as a SPARQL endpoint and use sameAs.org as a discovery endpoint while keeping the default depth limit set to 1.

6.1.2 Experimental Design

To evaluate our approach we designed an experiment whereby we compared our algorithm against standard RDF crawlers, since they use the most similar data retrieval behaviour to our own algorithm. This evaluation looks at the performance of our algorithm in two different retrieval scenarios, both these scenarios assume that the user has some URIs for which they want to retrieve relevant RDF. We describe the design of the experiment in this section and then present our evaluations based on this experiment in the subsequent chapter.

1. **Retrieval Scenario** - This scenario represents optimal retrieval conditions where the datasets from which the URIs originate are functioning normally and fully available on the Web.
2. **Recovery Scenario** - This scenario represents poor retrieval conditions where the datasets from which the URIs originate are unavailable on the Web and only secondary data sources are available.

6.1.2.1 Other Algorithms

To evaluate the expansion algorithm we compare its performance on a number of datasets against several other crawlers. We use two crawlers which are run locally and a third which is run via a Hypertext Transfer Protocol (HTTP) Application Programmers Interface (API). In this section we describe the behaviours of the three crawlers and how we configured them for our experiments.

[5] Available at http://www.dotnetrdf.org/expander/defaultProfile
[6] http://dbpedia.org

LDSpider LDSpider[7] can be characterised as as a truly dumb crawler. Given a starting URI it attempts to retrieve all RDF it can within a certain depth of the starting URI. It resolves all URIs it encounters and adds the RDF retrieved to its output. While LDSpider can be configured to be a more selective crawler this is only supported by writing your own Java program which uses the crawlers API directly. For ease of experimental set-up, and so we had a dumb crawler to make a comparison with, we chose not to configure it for our experiment.

Slug Slug[8] is a crawler designed to be highly configurable in its behaviour. Unlike LDSpider it can be easily configured to behave as desired just by changing settings in an RDF configuration file. Due to this ease of configuration we chose to configure Slug to behave as closely as possible to our expansion algorithm i.e. limit it to follow `owl:sameAs` and `rdfs:seeAlso`. This gave us an intelligent crawler to compare our crawler against in addition to comparing it against the dumb crawler provided by LDSpider in its default state.

URIBurner URIBurner[9] is a online service provided by OpenLink Software which uses their Virtuoso sponger technology OPL (2009) to retrieve RDF they consider relevant about a submitted URI. It can use their local caches of RDF data as well as arbitrary lookup services which may return non-RDF data which they then transform into RDF. We use this as one of the crawlers since it is potentially fault tolerant as it can use local caches and because it can use arbitrary lookup services in a manner similar to our own algorithm.

6.1.2.2 Test Harness

In order to conduct our experiment we constructed a test harness which would take in a text file containing one URI per line and the algorithm to run. The harness runs the algorithm against each URI and provides a results file for that URI containing all the triples the algorithm retrieved. We configured each algorithm to provide either NTriples

[7] `http://code.google.com/p/ldspider/`
[8] `https://github.com/ldodds/slug`
[9] `http://www.uriburner.com`

or NQuads output since they are RDF serializations which are easy and fast to process and conduct analysis over.

Firstly each algorithm was run against each dataset under normal conditions (our retrieval scenario). We then ran each algorithm again but this time used a proxy server to intercept and block all attempts to access the location of each primary data source e.g. dbpedia.org blocked for running the DBPedia dataset. We used the open source Privoxy[10] for this since it was freely available and easy to configure the required blocking. This second run allowed us to simulate our recovery scenario i.e. the situation where the main dataset is unavailable so that we can compare the results of the algorithms performance in the two scenarios.

6.1.2.3 Evaluating Performance

To compare the performance of the algorithms we used the standard information retrieval metrics of Precision, Recall and F_β number. We used this analysis to determine how accurate each algorithm was in terms of the triples it retrieved for each URI.

One issue we had in designing this experiment was selecting an appropriate benchmark. The usual approach for such an experiment is to have humans either generate the expected relevant results or evaluate the retrieved results to determine which results are relevant. Our problem is that even with our smallest dataset (British Broadcasting Corporation (BBC) Programmes) we generate 302MB of data over the 4 algorithms which is around 1.6 million triples. This climbs to 11.8GB of data with 57 million triples for the dataset that produced the most data (DBPedia). Human analysis of this volume of data would be so costly and time consuming that we decided to take a different approach. We are fully aware that our use of NTriples or NQuads as the output format for the algorithms inflates the size of the data on disk due to the verbosity of the serialization but the quantity of triples is still well beyond the scale at which it can be manually evaluated by humans.

As we have four algorithms all of which we know should produce relevant data we decided to create a synthetic benchmark based on the data they produced. Essentially we analyse all the data from the algorithms and count for each distinct triple the number of

[10]http://www.privoxy.org

algorithms that return that triple. To correctly count triples containing blank nodes we generate the minimal spanning graphs for all blank nodes in each algorithms data, and then use graph isomorphism to determine which are equivalent. With this analysis done our benchmark is then the majority consensus of the algorithms on what is relevant. Since we were uncertain about how effective a benchmark this would result in, we generated multiple benchmarks for each dataset (50, 75 and 100% majorities respectively) so that in the next phase of our analysis we could use whichever seemed most appropriate i.e. realistic.

Generally we found that the 75% benchmark appeared to be the most realistic. A 50% benchmark resulted in the majority of retrieved triples being deemed relevant and algorithms were all ranked very high by the analysis. On the other hand a 100% benchmark resulted in almost all retrieved triples being deemed irrelevant due to differing crawling and retrieval strategies between the algorithms, this resulted in lots of variation in the exact triples retrieved. Therefore the 100% benchmark made all the algorithms appear to perform very poorly, as we knew all the algorithms to be functional crawlers this was an unrealistic and unhelpful assesment of them. The only exception to this was for our largest dataset (ECS People) where one of the crawlers, LDSpider, failed to retrieve any data and so we were forced to use the 50% benchmark in that case. We explain the cause of this in our discussion of the ECS People dataset results in Section 7.1.4.

Once our benchmarks were generated for each dataset then we calculated the precision and recall of each algorithm against each dataset. We chose to use the F_2 number for the purpose of ranking the algorithms. This was chosen as we were interested primarily in the ability of the algorithms to retrieve the relevant data and therefore weighted recall twice as highly as precision.

In the next Chapter we present the results of this experiment and discuss what this means for the effectiveness of our recovery approach.

Chapter 7

Evaluating the Recovery Approach

In the preceding chapter we presented our recovery based approach to link integrity for the Semantic Web and described the design of an experiment to evaluate its effectiveness. In this chapter we present the results of this experiment and provide some conclusions about the effectiveness, scalability and viability of this technique.

7.1 Precision and Recall Experiment

For our initial experiments to evaluate performance we used three different datasets to evaluate the algorithms against. The datasets were chosen because each exhibited different properties and would help to evaluate how our approach performs against different kinds of dataset. For each dataset we take a subset of Uniform Resource Identifiers (URIs) used in the overall dataset, the number used for each dataset is given in the descriptions below. The subset of URIs is generated using a SPARQL Protocol and RDF Query Language (SPARQL) query against the appropriate SPARQL endpoint for each dataset.

1. **British Broadcasting Corporation (BBC) Programmes:** This dataset is provided by the BBC and contains information about TV programmes broadcast on the BBC. We use a subset of this dataset limited to programmes on BBC 1

(their flagship channel), which includes many programmes that have large amounts of data about them e.g. soaps, news broadcasts and many short running one-off programmes. This results in a dataset containing 233 URIs. This dataset is interesting because it has few (if any) links to external datasets and the concepts contained in it are not widely reused elsewhere on the Semantic Web. As a small and insular dataset it provides data which is relatively quick to obtain and process unlike some of our larger datasets, but which should be quite challenging for our algorithms to perform well on.

2. **DBPedia Countries:** This is a subset of the DBPedia dataset limited to just the URIs which are defined to be of `rdf:type` country in the DBPedia ontology[1]. This gives us a dataset containing 1508 URIs representing both existing countries who we expect there to be lots of data about plus many defunct countries with minimal data about them e.g. historical countries such as the California Bear Republic. This dataset was chosen because it contains widely agreed upon and reused URIs which can be found in many datasets on the Semantic Web.

3. **Drugbank:** This dataset contains all the URIs from Drugbank[2,3] which represent drugs, this gives a total of 4,772 URIs. This dataset is similar to the DBPedia Countries dataset in that it forms a hub of a portion of the linked data space that revolves around biomedical and life sciences data and therefore has a high degree of interlinking.

4. **Electronics and Computer Science (ECS) People:** This is a subset of the Southampton resilient knowledge base Explorer[4] dataset limited to the URIs which are defined to be of `rdf:type` person in the Advanced Knowledge Technologies ontology[5]. This gives a dataset containing 11,108 URIs. We chose to use this dataset because it is similar to the DBPedia Countries dataset in that the URIs in it are frequently reused across other datasets on the Semantic Web. It is also a large dataset as per our definition of large in our hypothesis so allows us to see how well our approach works on datasets of such size.

[1]Specifically those with the type `http://dbpedia.org/ontology/Country`
[2]`http://www4.wiwiss.fu-berlin.de/drugbank/`
[3]`http://www.drugbank.ca`
[4]`http://southampton.rkbexplorer.com`
[5]Specifically those with the type `http://www.aktors.org/ontology/portal#Person`

These datasets were selected partly because of our familiarity with each of these but primarily because we consider them to be representative samples of the types of datasets available on the Semantic Web. The BBC programmes dataset is a typical example of an insular dataset, by this we mean a dataset published by a single entity which links only within itself and does not provide links to external data sources. Insular datasets are challenging for link integrity solutions because their data is unlikely to be as widely available, if the publisher has not made the effort to provide external links then other data publishers are unlikely to provide links back to the dataset. DBPedia and Drugbank are excellent examples of hub datasets, they represent large amounts of data which act as hubs for some domain and provide many internal and external links. These datasets are often used as the first choice for obtaining data in a particular area and thus other data publishers are motivated to link to them and replicate their data because of its obvious utility. We expect hub datasets to be the easiest to apply link integrity solutions to as there are large numbers of secondary data sources associated with them. The ECS people dataset represents a middle ground between insular and hub datasets, it represents a type of data that can be considered a hub of an area (bibliographic data) but limited to the the data from a single publisher so also partially insular. However this dataset comes from a domain where publishers have traditionally collaborated and so we still expect to see reuse of identifiers in secondary data sources as with hub datasets. As a result we would expect that this dataset will be easier to provide link integrity for than a insular dataset it will not be as easy to provide as with a true hub dataset.

In our results we expected to see that our algorithm would perform similarly to the other algorithms in the normal retrieval scenario (all data sources available) and outperform the other algorithms in the recovery scenario (primary data source unavailable). Generally we expected to see that our algorithm would exhibit lower precision than the others because it would retrieve data from sources that the others were unable to locate by simple crawling but that its recall would be as good or better. In the experiments we expect Slug to be the algorithm most likely to outperform our own in the retrieval scenario since we specifically configured it to function as closely to our own as possible for comparison purposes. Conversely in the recovery scenario we consider that URIBurner is most likely to outperform our algorithm since we have no control over what data it can access and so have to make the assumption[6] that it would have access to cached

[6]This assumption being based on OpenLink's description of the algorithm in OPL (2009)

data even in the event of dataset failure.

Another issue we encountered is that the synthetic benchmarks we generate tend to bias the results towards the other algorithms in the retrieval scenario. This is because our algorithm is usually capable of retrieving data even when the starting URI is not accessible. This leads to many cases where our algorithm retrieves some data for a URI but the other algorithms are unable to retrieve anything resulting in the generated benchmark for that URI often being no triples. Therefore when precision is calculated for our algorithm it scores 0.0 since it retrieved some data when none was expected, yet when calculated for the other algorithms they score a perfect 1.0 because they didn't retrieve anything which was precisely what the benchmark expected. Essentially this leads to the other algorithms having slightly inflated precision scores for the retrieval scenario.

7.1.1 BBC Programmes Dataset

The results for the BBC Programmes dataset can be seen in Table 7.1. For the retrieval scenario our algorithm performs very well despite being marginally outperformed by Slug - a difference of only 0.008 in F_2 number - since as expected we show worse precision but better recall than Slug. The other two algorithms look decidedly poor against Slug and our own on this dataset, in particular LDSpider's dumb crawling behaviour causes it to retrieve large quantities of irrelevant data hence its poor precision.

In the recovery scenario things are more interesting, only URIBurner actually retrieves anything at all. This highlights the fact that our algorithms ability to recover data in the event of a data source failure is reliant upon two main factors:

1. The existence of relevant external data sources.
2. Our algorithm being configured appropriately.

Without any secondary data sources that are aware of the URIs we are interested in, our algorithm cannot retrieve anything and so cannot perform any better than the other crawlers.

Yet if you have some domain knowledge about the data you are interested in it should be possible to achieve much better results using our algorithm. To demonstrate this we ran our algorithm against this dataset again but this time using an expansion profile that was configured to use relevant secondary data sources. These sources were datasets we were aware of that had copies of the relevant data in them. As can be seen from Table 7.2 with a domain specific expansion profile our algorithm is able to achieve much better performance outperforming all the other algorithms for the scenario. Additionally when compared with results from Table 7.1 it can be seen that our algorithm in the abnormal recovery scenario outperforms two of the algorithms results under the normal retrieval scenario.

Algorithm	F_2 Number	Precision	Recall
Retrieval Scenario			
Slug	0.999938356	0.99969436	1.0
Expansion Algorithm	0.991505517	0.959508323	0.999975334
URIBurner	0.516346236	0.267646815	0.94693825
LDSpider	0.175031664	0.040992061	0.99453169
Recovery Scenario			
URIBurner	0.516346236	0.267646815	0.94693825
Expansion Algorithm	0.0	0.0	0.0
LDSpider	0.0	0.0	0.0
Slug	0.0	0.0	0.0

Table 7.1: BBC Programmes Dataset

Algorithm	F_2 Number	Precision	Recall
Expansion Algorithm	0.60977794	0.66909694	0.60977794
URIBurner	0.516346236	0.267646815	0.94693825
LDSpider	0.0	0.0	0.0
Slug	0.0	0.0	0.0

Table 7.2: BBC Programmes Dataset (Recovery Scenario using Domain specific Profile)

7.1.2 DBPedia Countries Dataset

The results for the DBPedia Countries dataset can be seen in Table 7.3. For this dataset our algorithm is outperformed by all the other algorithms despite no algorithm significantly outperforming any other. As can be seen in Table 7.3 our algorithm exhibits excellent recall but is hampered by its apparent poor precision. As discussed at the start of the results section this is due to both its ability to access data sources that the other algorithms are unaware of, and the fact that it is able to retrieve data for many URIs which cause the other algorithms to retrieve no data.

As a result of this we are also beaten in the recovery scenario which went against our expectations. We still show good recall of above 0.5 which is comparable with all other algorithms in this scenario. Unfortunately our already low precision scores are worsened due to the inability to access the triples from the primary data source, which would have made up a large part of the benchmark. Somewhat paradoxically Slug and LDSpider show an improvement in their figures, which is due to there being a portion of the dataset for which the benchmark is calculated to be empty. Under normal retrieval conditions the algorithms retrieve data successfully and so achieve poor precision scores for that portion of the dataset. Therefore under recovery conditions the algorithms fail to retrieve anything which actually gains them perfect precision scores for that part of the dataset, thus boosting their scores.

These results show that clearly we may need to work on the precision of our algorithm, though it is difficult to tell how much of the apparent lack of precision is genuine and how much is due to our benchmarking technique.

Algorithm	F_2 Number	Precision	Recall
Retrieval Scenario			
URIBurner	0.508071667	0.493644092	0.523415903
Slug	0.357593034	0.192212947	0.997593629
LDSpider	0.330951191	0.271078006	0.966237521
Expansion Algorithm	0.235423546	0.104985202	0.997342193
Recovery Scenario			
URIBurner	0.508071667	0.493644092	0.523415903
LD Spider	0.474801061	0.474801061	0.47481061
Slug	0.474801061	0.474801061	0.47481061
Expansion Algorithm	0.018366973	0.023073002	0.502068649

Table 7.3: DBPedia Countries Dataset

7.1.3 Drugbank Dataset

The results for the Drugbank dataset can be seen in Table 7.4. Firstly in terms of the retrieval scenario the results are in line with what we'd expect based on the previous experiments and our own performance expectations. We are again narrowly outperformed by Slug, but only by 0.03 in terms of F_2 number and we perform comparably in terms of recall. Interestingly, and against our expectations, we actually very narrowly outperform Slug in terms of precision which was not something we had not seen in any of our previous experiments. As with the previous experiments URIBurner and LDSpider performed significantly worse than our algorithm and Slug by a factor of almost 2.

One thing to note is that URIBurner actually performs relatively well on this dataset compared to its performance against other datasets e.g. ECS People. This would seem to support our earlier assertion that URIBurner design and operation appears be suited to certain kinds of datasets.

In line with our expectations the results from the recovery scenario show apparent poor performance of our algorithm primarily hampered by extremely poor precision. As we have stated in our analysis of previous results this is usually due to the fact that in the recovery scenario we will succeed in obtaining data from secondary sources which the other algorithms cannot reach. Therefore this data is not in the synthetic benchmarks and as a result our calculated precision is very low. Admittedly both Slug and LDSpider also perform very poorly but they benefit from not scoring as poorly on precision as they simply return nothing for most URIs under the recovery scenario.

Algorithm	F_2 Number	Precision	Recall
Retrieval Scenario			
Slug	0.743500701	0.627924945	0.974700124
Expansion Algorithm	0.713347737	0.6287085	0.909308884
URIBurner	0.435532831	0.0256177288	0.653407618
LDSpider	0.389445542	0.192230147	0.868189438
Recovery Scenario			
URIBurner	0.435532831	0.256177288	0.653407618
LDSpider	0.0158843252	0.0158843252	0.0158843252
Slug	0.0158843252	0.0158843252	0.0158843252
Expansion Algorithm	0.06500628	0.066706419	1.60447616

Table 7.4: Drugbank Dataset

To investigate whether our poor performance was due to the need for domain specific knowledge as we saw with the BBC dataset in Section 7.1.1, and as we have seen with other link integrity systems e.g. DSNotify, we again re-ran our algorithm using a different expansion profile. For this dataset we replaced our usual DBPedia[7] SPARQL endpoint with the Sindice[8] SPARQL endpoint since it contains a copy of the Drugbank data. While we thought that this would improve results it actually worsened them since although Sindice gave us some relevant data, it also gave us additional irrelevant data which worsened our precision score and therefore our F_2 number.

[7]http://dbpedia.org
[8]http://sparql.sindice.com

7.1.4 ECS People Dataset

The results for the ECS People dataset can be seen in Tables 7.5. Note that unlike the previous three datasets we were forced to use a 50% benchmark rather than a 75% benchmark for assessing the algorithms performance. This was due to the fact that LD-Spider would not retrieve any data for this dataset, even under normal retrieval scenario conditions due to its strict adherence to the restrictions emplaced by the `robots.txt` file on the Southampton RKB explorer website[9]. As can be seen in the `robots.txt` file the disallow rule for /id/, which is the URI under which all the URIs we wished to retrieve resided, meant that LDSpider refused to perform a crawl and retrieve any data.

With no results from LDSpider in order for a triple to be deemed relevant in the 75% benchmark it must have been retrieved by all three of the other algorithms i.e. in effect it became a 100% benchmark. Unfortunately due to the different crawling strategies we observed that only 6% of the dataset had a non-empty benchmark, this meant any assesment of the algorithms against it would not properly reflect the performance of the different algorithms. So we used the 50% benchmark instead to assess this dataset since it had a non-empty benchmark for 84% of the dataset which represented a realistic benchmark to use for this dataset.

Similarly to the BBC dataset we see that Slug beats our own algorithm overall on F_2 number but for this dataset we outperform Slug in terms of recall. As mentioned previously we score lower on precision but this is primarily due to our ability to use data sources that the other algorithms cannot reach using their crawling strategies. Surprisingly URIBurner shows unexpectedly poor performance on this dataset implying that its data retrieval behaviour may be suited only to certain kinds of dataset.

As with the DBPedia dataset we show very poor performance in the recovery scenario which was very much against our expectations. As we have already discussed some of this may be due to deficiencies in our benchmarking technique.

[9] `http://southampton.rkbexplorer.com/robots.txt`

Algorithm	F_2 Number	Precision	Recall
Retrieval Scenario			
Slug	0.932664469	0.090901398	0.983630019
Expansion Algorithm	0.700407427	0.444692845	0.997399084
URIBurner	0.066872402	0.046400893	0.094303709
LDSpider	0.029650437	0.029650437	0.029650437
Recovery Scenario			
URIBurner	0.273270014	0.169555159	0.483843966
LDSpider	0.14516129	0.14516129	0.14516129
Slug	0.14516129	0.14516129	0.14516129
Expansion Algorithm	0.04744997	0.031523062	0.22056149

Table 7.5: ECS People Dataset

7.2 Conclusions on the Precision and Recall experiment

Clearly as can be seen in the preceding section our results were very mixed and not nearly as good as we had predicated. Our initial conclusion based upon these results is that it appears that some types of datasets are more suited to the application of our technique than others. Ironically the insular dataset (BBC Programmes) which we thought we would perform the worst on actually yielded us the best results, although in the recovery scenario only when we used domain specific knowledge. Conversely for the more interlinked datasets where we expected our algorithm to perform well we actually appeared to perform quite badly. As discussed in the opening part of this section we believe some of this is in part due to our synthetic benchmark but it also indicates that the algorithm is better suited to some kinds of data than others.

This highlights the issue we have raised in relation to several systems we discussed in Chapter 4 (e.g. DSNotify) of the need for domain specific knowledge in order for a system to be effective. As our results show the presence of this knowledge can make a major difference to performance which causes us a problem in terms of scalability and viability since requiring this additional knowledge has two effects:

1. It makes it harder to scale the solution since each new dataset you wish to apply it to potentially needs new domain specific knowledge.

2. It makes the system less viable (according to our definition from part 2 of our hypothesis, see Section 1.1) as humans potentially need to be in the loop more to provide the necessary domain knowledge.

While we have seen that given adequate domain knowledge it can provide excellent recovery of data, as with the BBC programmes dataset (see Table 7.2), in other cases such as the Drugbank dataset it has provided very poor results even given some domain knowledge. This raises the question of whether our approach is universally applicable or only applies to certain kinds of dataset. This issue of the need for domain knowledge and whether it can be automatically discovered is suggested as one potential avenue of future research in Section 8.3.3.

As a result of these mixed results we decided to design another experiment that would attempt to evaluate the algorithms in terms of a specific use case. This was intended to see if this would give us more consistent results and rule out the apparent bias of our synthetic benchmark from this experiment, the results of this experiment are presented in the subsequent sections.

7.3 Query Use Case Evaluation

Given that the pure precision and recall based evaluation had shown very mixed results and made it difficult for us to draw any firm conclusions about the effectiveness of our approach we decided to try a different tact. Our idea was that rather than evaluating against the synthetic benchmarks we'd test against a real world use case in which we envisaged our solution to link integrity playing a part. The use case is that a user wants to make some SPARQL queries against a dataset, in the event that the dataset itself is not available could the user still get accurate answers for his queries using data recovered by our expansion algorithm? If they could get accurate answers would these be any more accurate than those obtained by evaluating the queries over data obtained by the other pure crawler algorithms?

In some ways this is similar to our previous experiment in that we evaluate performance based upon precision and recall. The difference from our preceding experiment is that we can obtain a genuine benchmark since we can use the actual SPARQL endpoints of the datasets we are testing against to check what the correct result should be under normal conditions.

Another advantage of this experiment is that we can simply reuse the data we have already gathered to check against the benchmarks we obtain from the real endpoints, which means we don't have to repeat the data gathering. As we already had all the data it was relatively easy for us to construct a new test harness which would run this alternate evaluation over our existing data. The main difficulty in designing this experiment was choosing queries that were both realistic and tractable i.e. those that we thought one or more of the algorithms could potentially answer with their partial data in the recovery scenario. For each dataset we chose to evaluate the following queries:

1. **Label Query**

   This is a simple query designed to check that the retrieved data includes the very basic `rdfs:label` property. This is effectively the human readable label associated with each of our URIs in the dataset, see Listing 7.1 for a sample query.

2. **Description or Type Query**

   The description query is a variation on the label query but it asks for the description of the URI, depending on the dataset this may be specified differently e.g. `dbp:abstract` in DBPedia, see Listing 7.2 for a sample query. This query is more difficult than the first since we expect that while algorithms may find the correct label in their secondary data sources, it is less likely that they find the exact description from such sources since it is a larger block of text.

   The type query is an alternative to the description query and is used for datasets that do not include a longer textual description of the URIs, it asks for the `rdf:type` of a URI, see Listing 7.3 for a sample query. Again this is harder than the label query since although secondary data sources may also assert types for URIs, they will typically use types from their own ontologies rather than those from the primary data source.

3. **Relationship Query**

   This query is designed to see whether relationships expressed in the main data source can be recovered by the algorithms. As with the description query this is tailored to each dataset to find a specific type of relationship e.g. `po:episode` in the BBC Programmes dataset, see Listing 7.4 for a sample query. We expect this to be the most difficult query since while simple facts like labels and descriptions may be easily retrieved from secondary sources they may not contain more complex facts like relationships.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>
SELECT ?label
WHERE
{
        ?uri rdfs:label ?label
}
```

Listing 7.1: Label Query

```
PREFIX dbp: <http://dbpedia.org/property/>
SELECT ?description
WHERE
{
        ?uri dbp:abstract ?description
}
```

Listing 7.2: Description Query

```
SELECT ?type
WHERE
{
        ?uri a ?type .
}
```

Listing 7.3: Type Query

```
PREFIX po: <http://purl.org/ontology/po/>
SELECT ?episode
WHERE
{
        ?uri po:episode ?episode
}
```

Listing 7.4: Relationship Query

Note that these queries are intentionally simple, while there is an argument that a real world use case would involve much more complex queries we felt that we must first show that our technique can answer simple queries. If we can successfully demonstrate this then we can have some degree of confidence in making statements about whether our approach would provide sufficient data to answer more complex queries. Also real world queries would typically tend to use data spread across the dataset, whereas in our experiment we have individual sets of data for each URI that we were evaluating our approach for. Therefore even if we were to load all our data together and query over it we would still only be representing a small portion of the overall dataset. Thus the more complex and arguably real world queries we could potentially perform would still be limited by this.

Query Variations Since we expect and know that the data retrieved by our algorithms and those we are comparing ourselves against will be incomplete we evaluate two variations of each query:

1. **Basic Variation:** This variation is the basic query as shown in the example queries in Listings 7.1-7.4 tailored to each dataset where applicable.
2. **Advanced Variation:** This variation is the query rewritten to incorporate some explicit reasoning into the query so that where the algorithm has partial data it may still be able to obtain an answer for this variation even when the basic variation fails.

   For an example of this see Listing 7.5 in which the query is rewritten to allow for extracting values obtained from related secondary data sources.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
SELECT ?label
WHERE
{
        { ?uri rdfs:label ?label }
        UNION
        { ?uri owl:sameAs ?alias . ?alias rdfs:label ?label }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias rdfs:label ?label }
}
```

Listing 7.5: Advanced Variation of Label Query

The exact queries for each dataset and the SPARQL endpoints against which we ran these can be found in Appendix A.

Expected Results What we expect to see in the results from this experiment is that all of the algorithms perform well on the queries under the normal retrieval scenario conditions, while only our algorithm and URIBurner will return reasonable results under the recovery scenario. Typically in the retrieval scenario we'd expect results around the

0.8 mark for F_2 number as all algorithms will likely retrieve some secondary data that is not present in the SPARQL endpoint used to determine the benchmark. We think that we will see Slug and LDSpider fall below this mark and perform worse than our algorithm as they will tend to retrieve more irrelevant data and thus score lower on precision.

In the recovery scenario all the results should be worse than in the retrieval scenario, and for Slug and LDSpider we should see very low scores as we know they are incapable of retrieving any data in that scenario so should not be able to answer queries successfully. On the more insular datasets, like the BBC programmes one, we may see very low scores across the board as it is unlikely that even our algorithm will be able to retrieve usable data. Additionally we should see that under the retrieval scenario the basic variations of the queries perform better than the advanced and in the recovery scenario the opposite will most likely be true.

The aim of this experiment is to demonstrate the ability of our algorithm to answer queries even with partial incomplete data returned in the recovery scenario, and by doing so further prove part 1 of our hypothesis. That is we hoped that this experiment would provide better confirmation that our adaptation of Harrison and Nelson's Just-in-Time (JIT) style approach to link integrity (Harrison and Nelson, 2006) for the Semantic Web does provide effective link integrity.

Note on Presentation of Results In the subsequent sections for simplicity and readability we have shown only the F_2 numbers for each algorithm against each query in the result tables (see Tables 7.6-7.10). Please note that these tables are not ordered from best to worst performing algorithm like earlier tables since different algorithms performed best on different queries so such ordering is not applicable, instead the tables are simply ordered alphabetically.

7.3.1 BBC Programmes Dataset

The results for the queries evaluated against the BBC programmes dataset can be found in Table 7.6, the exact queries evaluated can be found in Appendix A.1.

The first thing to notice is that for this dataset there was no difference in performance for any of the algorithms between the basic and advanced variants of the queries. This is likely due to the fact that this dataset is very insular and doesn't really link out to external datasets. Looking at the results for specific queries we see that all the algorithms exhibit near perfect scores for the label query with differences in performance small enough to be considered indistinguishable. Similarly algorithms all scored highly on the description query (though not as near perfect as on the label query) with again the differences in performance being indistinguishable.

Interestingly the algorithms all performed a lot worse on the relationship query which for this dataset was asking for all episodes of programmes. The likely explanation of this is that because the BBC do not provide a public SPARQL endpoint we had to use an endpoint which has a mirror of the data. Unfortunately that mirror is not entirely up to date as far as we are aware, which means that the algorithms that retrieved their data directly from the BBC, obtained data pertaining to episodes of the programmes broadcast since the mirror was last updated. Thus causing them to exhibit low precision scores resulting in their lower performance overall.

In the recovery scenario we again see that none of the algorithms, bar URIBurner, were able to answer any of the queries accurately at all. This shows clearly that pure crawler based algorithms are completely ineffective for data recovery usage i.e. in actually providing link integrity. What it also shows though is that even the augmented approach we have used can be useless if our algorithm cannot locate relevant secondary data sources through the services it uses. This confirms our expectation that because the dataset is insular it is very difficult to retrieve usable data from secondary sources because it just does not exist.

In evaluating our results for this dataset from the precision and recall experiment (see Section 7.1.1) we looked at a variation of this dataset where we re-ran our algorithm using a domain specific expansion profile which allowed us to improve our results significantly. As we had this data from our preceding experiments we ran this query based evaluation on it as well to see how domain knowledge would affect the results, these results can be seen in Table 7.7. With the domain specific profile our algorithm exhibited perfect scores across all queries which shows just how valuable domain specific knowledge can be, particularly for an insular dataset like this one.

The reader must be aware of the fact that our domain specific profile used the same

SPARQL endpoint used as the reference endpoint for this dataset so these perfect scores are not an entirely unexpected result. Despite this somewhat artificial result it should be apparent that a little domain knowledge can go a long way, if you know of good secondary data sources then our algorithm can perform as good or better than its performance in the normal retrieval scenario.

	Label Query		Description Query		Relationship Query	
Algorithm	Basic	Advanced	Basic	Advanced	Basic	Advanced
Retrieval Scenario						
Expansion Algorithm	0.982823618	0.982823618	0.884120172	0.884120172	0.476536642	0.476536642
LDSpider	0.982832618	0.982832618	0.888412017	0.888412017	0.477998963	0.477998963
Slug	0.982832618	0.982832618	0.884120172	0.884120172	0.47706644	0.47706644
URIBurner	0.935622318	0.935622318	0.836909871	0.836909871	0.453248107	0.453248107
Recovery Scenario						
Expansion Algorithm	0.0	0.0	0.0	0.0	0.0	0.0
LDSpider	0.0	0.0	0.0	0.0	0.0	0.0
Slug	0.0	0.0	0.0	0.0	0.0	0.0
URIBurner	0.935622318	0.935622318	0.836909871	0.836909871	0.453248107	0.453248107

Table 7.6: Query Use Case results for BBC Programmes Dataset

	Label Query		Description Query		Relationship Query	
Algorithm	Basic	Advanced	Basic	Advanced	Basic	Advanced
Expansion Algorithm	1.0	1.0	1.0	1.0	1.0	1.0
LDSpider	0.0	0.0	0.0	0.0	0.0	0.0
Slug	0.0	0.0	0.0	0.0	0.0	0.0
URIBurner	0.933628319	0.933628319	0.836283186	0.836283186	0.467220526	0.467220526

Table 7.7: Query Use Case results for BBC Programmes Dataset (Recovery Scenario using Domain specific profile)

7.3.2 DBPedia Countries Dataset

The results for the queries evaluated against the DBPedia[10] countries dataset can be found in Table 7.8, the exact queries evaluated can be found in Appendix A.2. As you will see from the table these results are much more varied and have some unexpected results, such as all the algorithms performing well on the hardest query (the relationship one) under the recovery scenario.

First lets look at the normal retrieval scenario where we see a marked variation in the scores obtained by the different algorithms, our algorithm scores very well across the board outperforming all the other algorithms substantially. This is an encouraging result for us as it gives a clear indication that our algorithm is returning relevant data which is something that our pure precision and recall experiment made it hard to determine.

One interesting oddity is how poorly URIBurner performs in comparison to the other algorithms and we have no real explanation for this, especially since the authors of that algorithm are involved in the DBPedia project and we would have expected their algorithm to utiltise DBPedia as a data source. URIBurner still manages to score in the mid range for the hardest query though we expect this is mostly due to the actual data as we explain in our analysis of the recovery scenario later in this section.

LDSpider and Slug both score very much in the mid range showing reasonable but not great performance. Again it is strange to see that Slug performs quite poorly on the description query yet performs almost as well as our own on the harder relationship query. We suspect that this is due to the differences in our link following behaviour versus Slug, as we explained in Section 6.1.1 we are highly selective in which links we follow whereas Slug follows any link of the relevant type. This means that Slug will obtain more potentially irrelevant data, thus harming its precision scores and therefore reducing its final F_2 number.

In the recovery scenario the results are more consistent and we see very poor performances from all the algorithms. No algorithm has a clear edge over any other in this scenario with all performing within 0.01 or so of each other. One very unexpected result for this scenario is that all the algorithms score in the mid-range for the hardest relationship query and most of them score identically. Having obtained this result we examined

[10]http://dbpedia.org

the data in more detail and found that this is a result of a significant portion of the URIs in the dataset actually having no values for the relationship we were querying for. This meant that there are many URIs in the dataset where the correct answer for the query is no results so for all of these where the algorithms also returned minimal or zero data they score perfectly since the benchmark was zero results.

These results highlight the fact that an approach such as ours which looks to use hubs like DBPedia as a source of secondary data are reliant on the availability of such hubs. In the event of a major hub like this becoming unavailable our performance can be severely impacted since we lose the ability to use its SPARQL endpoint to augment our crawling results.

Another problem these results highlight are the variety of Resource Description Framework (RDF) vocabularies used across different data sources. If we look at the data retrieved by our algorithm in the two scenarios in terms of raw quantity we are still retrieving a substantial amount of it. In the retrieval scenario we retrieved 6.52 GB of data versus 3.71 GB in the recovery scenario i.e. approximately 56% of the retrieved data can be obtained even in the recovery scenario. Despite this large amount of data available to us we see up to a factor of 10 reduction in our accuracy on some of the queries. Yet when we inspect some random samples of the data we can see that the values we were querying for are present in the data, but they are expressed using properties from different vocabularies so our queries as written did not retrieve those results. What we are talking about here is again a lack of domain knowledge, if we knew in advance the alternate properties that secondary data sources used to express the properties we are interested in we could write advanced variations of the queries that could potentially substantially improve our results.

Vocabulary Mapped Query To get an idea of how the algorithms would perform if we took the vocabulary mapping into account we evaluated a mapped version of the label query which uses additional properties like `fb:type.object.name` which we know that at least one major secondary data source - Freebase[11] - makes use of. These results can be found in Table 7.9 and show that some algorithms improve their performance when the query uses additional domain knowledge while others actually perform worse.

[11] http://www.freebase.com

Firstly for the retrieval scenario we see that our algorithm, which performed best for the equivalent query (the advanced label query), performs slightly worse though still exhibits good performance and remains the best performer overall. All the other algorithms show minor performance improvements so our algorithm actually stands out as being unusual in performing worse with the more domain specific query. Having looked at the more detailed breakdown of results we see that our algorithm loses a small amount of precision thus worsening its F_2 number. This is due to the fact that although the secondary data we now return values for in answer to the queries contains additional correct results it also contains additional incorrect results.

However in the recovery scenario it is a different story with Slug, LDSpider and URIBurner showing no change in performance as opposed to our algorithm which exhibits a close to five fold increase in performance. This nicely illustrates our point that the data is clearly there to answer queries correctly. However if you want to successfully answer queries in abnormal conditions having some domain knowledge about how secondary data sources express their data in RDF can dramatically increase the chances of correctly answering them. While the performance we have seen from our algorithm is still firmly in the mid-range, so not excellent, we would expect that most clients would rather be able to have accurate answers to a good portion of their queries as opposed to answers to none of them. This however comes with a proviso, which we shall discuss properly in Section 8.2.1, that if a user is going to use a technique like ours to answer queries then they must be made aware of the risks of using partial data and that the answers they are getting are not necessarily complete or correct.

	Label Query		Description Query		Relationship Query	
Algorithm	Basic	Advanced	Basic	Advanced	Basic	Advanced
Retrieval Scenario						
Expansion Algorithm	0.893688584	0.777958305	0.847795563	0.0826552957	0.907146833	0.889841203
LDSpider	0.469462045	0.465929977	0.357704768	0.359465093	0.437002653	0.440981432
Slug	0.593625848	0.587524734	0.259153915	0.260379815	0.87458979	0.869000552
URIBurner	0.118942053	0.13062131	0.150834764	0.156802934	0.437665782	0.441644562
Recovery Scenario						
Expansion Algorithm	0.091155361	0.100453133	0.127320955	0.133952255	0.437002653	0.440981432
LDSpider	0.089522546	0.097480106	0.133289125	0.139257294	0.437002652	0.440981432
Slug	0.089522546	0.097480106	0.132289125	0.13957294	0.437002653	0.440981432
URIBurner	0.118942053	0.13062131	0.150834764	0.156802934	0.437665782	0.441644562

Table 7.8: Query Use Case results for DBPedia Countries Dataset

Algorithm	Mapped Label Query
Retrieval Scenario	
Expansion Algorithm	0.732452582
LDSpider	0.469137674
Slug	0.5928561
URIBurner	0.1234000496
Recovery Scenario	
Expansion Algorithm	0.47927289
LDSpider	0.089522546
Slug	0.089522546
URIBurner	0.1234000496

Table 7.9: Query Use Case results for DBPedia Countries Dataset with Vocabulary Mapped Label Query

7.3.3 Drugbank Dataset

The results for the queries evaluated against the Drugbank[12] dataset can be found in Table 7.10, the exact queries evaluated can be found in Appendix A.4.

As we have seen with the other datasets the results are quite varied though relatively consistent for this dataset at least. For all the queries we see good performance in the retrieval scenario for the three algorithms - ours, LDSpider and Slug - which we expected to perform well though Slug has a clear edge over the other two for this dataset. Again URIBurner shows stubbornly mid range performance which would appear to confirm our earlier assertions about it being better suited to some datasets than other. Another thing to note is that as with other datasets for both the label and type query all the algorithms except URIBurner do worse on the advanced variations of the queries as opposed to the basic variations. This is due to the amount of extraneous information that the algorithms retrieve and thus produce in their query results which is not contained in the official endpoints from which we created our benchmarks.

In the recovery scenario the results are very poor, our algorithm, Slug and LDSpider all report identical scores across the board and are equally poor. The fact that the scores are all identical implies that none of the algorithms recovered data sufficient to answer any of the queries correctly in this scenario. That they score anything at all is only due to their being a limited number of URIs in the dataset for which the correct answer to the queries is no results. In the context of these results URIBurner's mid range performance from the retrieval scenario looks a lot more palatable. Despite the

[12]http://www4.wiwiss.fu-berlin.de/drugbank/

fact that it is only accurate in answering queries for around 50% of the URIs you would have to say that given the choice between no answers and half the answers a user would likely prefer the latter.

To some extent we always expected the recovery scenario results for this dataset to be among the poorest because the queries, especially the relationship one, are among the hardest we have used for any dataset. Even if we had better domain knowledge it is dubious whether we could improve on these results primarily because this dataset is so specialised. The secondary data sources that we can locate tend to contain related rather than overlapping data as in the case of datasets pertaining to more common things like countries in the DBPedia dataset. Due to this lack of overlap in the data answering anything but the most trivial of queries in the recovery scenario is very difficult and further complicated by variations in RDF vocabularies used.

In addition to the problem of use of differing RDF vocabularies across secondary data sources another RDF related issue we have observed in browsing random samples of this dataset is related to the different ways of expressing literals. In Drugbank itself the literals for the labels do not have an associated language specifier while in secondary data sources they typically do. According to the RDF and SPARQL semantics a literal such as `"Mestranol"` is not considered to be equal to `"Mestranol"@en` so even though we have effectively retrieved data that answers our queries it appears that we have not. This implies, as with the DBPedia dataset, that we need better domain knowledge in order to be able to answer queries effectively in the recovery scenario. In terms of raw data quantity we see that we retrieve about a third of the data in the recovery scenario versus the normal retrieval scenario so we should not really expect to see performance drops of factors of 200 between the two scenarios.

Literal Mapped Query In order to see how much difference to our results it would make if we rewrote the queries so that literals would match more often we re-ran our analysis for the two variations of the label query. We altered the queries so that any literal found with a language tag would have that language tag removed and therefore increase the likelihood that it would match the benchmark results from the Drugbank SPARQL endpoint.

The results of this can be seen in Table 7.11 and suprisingly show that adding in this domain knowledge makes minimal difference to the results. In the retrieval scenario our algorithm and Slug gain small boosts to performance on the advanced variation on the query while the others are unaffected. For the recovery scenario only our algorithm shows any change in performance and the improvement though by a factor of 2 still gives us a score so low that it is fair to say that our technique is entirely ineffective at recovering data for this dataset.

What this clearly demonstrates is that certain datasets appear to be much better suited to recovery using our approach than others, as we've seen in this experiment and the previous experiment adding some domain knowledge can lead to major performance improvements for some datasets. Yet for a dataset like this where we'd typically expect some domain knowledge to make a major difference we find that it makes so little difference as to be irrelevant. It is hard to characterise why this dataset has this problem as on the surface it has similar properties to the DBPedia Countries dataset in that it contains well defined concepts and is well interlinked with other relevant datasets, we discuss what could be done to address this problem in Section 8.3.3.

	Label Query		Type Query		Relationship Query	
Algorithm	Basic	Advanced	Basic	Advanced	Basic	Advanced
Retrieval Scenario						
Expansion Algorithm	0.781433361	0.702876966	0.781852473	0.738671501	0.823134954	0.823134954
LDSpider	0.760477787	0.760058676	0.760477787	0.760687343	0.771374686	0.771793797
Slug	0.898784577	0.790308031	0.898365465	0.836734476	0.905699916	0.905699916
URIBurner	0.542958927	0.542539816	0.542958927	0.543168483	0.556789606	0.556789606
Recovery Scenario						
Expansion Algorithm	0.003562448	0.003772003	0.003772003	0.003352892	0.079631182	0.079631182
LDSpider	0.003562448	0.003981559	0.003772003	0.003562448	0.079631182	0.079631182
Slug	0.003562448	0.003981559	0.003772003	0.003562448	0.079631182	0.079631182
URIBurner	0.542958927	0.542539816	0.542958927	0.543168483	0.556789606	0.556789606

Table 7.10: Query Use Case results for Drugbank Dataset

	Label Query	
Algorithm	Basic	Advanced
Retrieval Scenario		
Expansion Algorithm	0.781433361	0.733302675
LDSpider	0.760477787	0.760058676
Slug	0.898784577	0.8285619
URIBurner	0.542958927	0.543539816
Recovery Scenario		
Expansion Algorithm	0.003562448	0.069288907
LDSpider	0.003562448	0.003981559
Slug	0.003562448	0.003981559
URIBurner	0.542958927	0.542539816

Table 7.11: Query Use Case results for Drugbank Dataset with Literal Mapped Label Queries

7.3.4 ECS People Dataset

The results for the queries evaluated against the ECS People countries dataset can be found in Table 7.12, the exact queries evaluated can be found in Appendix A.3.

This dataset provides some of the most varied results in the normal retrieval scenario with our algorithm and Slug performing well, whilst LDSpider and URIBurner perform very poorly. For the label query our algorithm performs the best though Slug also performs comparably to ours, URIBurner again performs unexpectedly poorly which is hard to explain. We know from the technical report (OPL, 2009) describing the technology behind the URIBurner service that it dynamically retrieves data from the Web so there is no obvious reason why it should perform so poorly on a well interlinked dataset like this, yet performs very well on insular datasets like the BBC programmes dataset.

For the type query both our algorithm and Slug perform more in the mid range which seems strange given the high scores for the label query. Having looked at the more detailed figures we found that both algorithms score close to perfect recall but score very low on precision. What this implies is that the secondary data sources are expressing a lot of types themselves, most likely in their own vocabularies, which aren't present in the expected data leading to a lot of extraneous results and thus poor precision scores. In terms of the performance of LDSpider this is due to the same problems we discussed in relation to the original precision and recall experiments that LDSpider refuses to retrieve any data for this dataset due to its strict adherence to the robots.txt protocol.

That LDSpider's scores in this scenario are not zero represents the fact that there is a small portion of the dataset for which the correct answer to the queries is no results.

In the recovery scenario we see poor performances for all the algorithms though we see that our algorithm does correctly answer a portion of the advanced variations of the label and type queries. This demonstrates the fact that if we are going to use secondary data sources we do need to use queries that take into account aliases of the URIs which we were asking the queries about. As can be seen in the table for the basic variations of the label and description query our algorithm is effectively scoring zero (since it scores match those of LDSpider) but with the advanced variations it is able to successfully answer a portion of the queries correctly. This is not a large proportion but again we would make the point that in most cases a client would prefer to be able to get some correct answers than none at all.

	Label Query		Type Query		Relationship Query	
Algorithm	Basic	Advanced	Basic	Advanced	Basic	Advanced
Retrieval Scenario						
Expansion Algorithm	0.871192011	0.856604516	0.662566571	0.618500568	0.823808588	0.823808588
LDSpider	0.083573228	0.083573228	0.083573228	0.083573228	0.238035264	0.238035264
Slug	0.821983454	0.814145035	0.624875013	0.610522387	0.831680707	0.831680707
URIBurner	0.13899184	0.13899184	0.133839887	0.1338939887	0.269546534	0.269546534
Recovery Scenario						
Expansion Algorithm	0.083573228	0.476051777	0.083573228	0.334738715	0.238035264	0.238035264
LDSpider	0.083573228	0.083573228	0.083573228	0.083573228	0.238035264	0.238035264
Slug	0.083573228	0.083573228	0.083573228	0.083573228	0.238035264	0.238035264
URIBurner	0.13899184	0.13899184	0.133839887	0.1338939887	0.269546534	0.269546534

Table 7.12: Query Use Case results for ECS People Dataset

7.4 Conclusions

As we have seen in this chapter our recovery approach has produced rather mixed results with a large degree of variation over different datasets. Despite these variations the results are promising and clearly show that given the right conditions an augmented crawling approach as a JIT recovery solution to link integrity can work for linked data on the Semantic Web. Therefore we can say that we have successfully fulfilled part 1 of our hypothesis (see Section 1.1) in that we have demonstrated that an existing link integrity approach from hypermedia can be applied to the Semantic Web and provide link integrity successfully.

In terms of our results some of the variations we saw went against our expectations with the prime example being the BBC programmes dataset (see Sections 7.1.1 and 7.3.1). As an insular dataset we expected to score well on the precision and recall evaluation but actually performed quite poorly. What this served to demonstrate is that our approach in reality does not work at all well for insular datasets where there are minimal secondary data sources available, particularly where these sources are not linked via major hubs like DBPedia. Yet at the same time with this dataset we also showed in both our experiments that applying relevant domain knowledge can lead to significant performance improvements and make the difference between recovered data being useless or usable.

This brings us to an important conclusion, while our approach would appear to be sound and workable it often needs domain knowledge to make it truly effective. If we return to our definition of viability from part 2 of our hypotheses we stated that viability meant minimal user involvement. As it stands our approach will typically need a high degree of user involvement and domain expertise in order to be effective, though this appears to vary depending on the dataset, which in our definition does not yet make it viable. Arguably the need for user involvement is not that high and once the algorithm is configured for a dataset there should be no further need for user involvement but we think this is still unviable. The reason for this is that we are not talking about just needing a user to enter a few bits of configuration data, we are talking about them being sufficiently knowledgeable about the domain of the dataset that they know of relevant secondary sources. This implies that you need expert users to configure the systems and that users may only be able to configure it for a limited number of datasets for which they have expert knowledge of. As a result we cannot say that we have properly fulfilled part 2 of our hypothesis since our technique on its own is not a fully viable approach to link integrity for the Semantic Web.

Another interesting point that has been raised by these experiments, particulary the query use case evaluation, is that the variation in vocabularies and encoding in data makes recovery of data all the harder. For example as we saw with the DBPedia countries dataset (see Section 7.3.2) differences in the vocabularies used by secondary data sources led to unexpectedly poor performance. They meant that even though we were able to obtain gigabytes of data using our algorithm in the recovery scenario most of this data appeared useless since our queries did not take into account the different properties used to express the same pieces of data. Yet again when we evaluated a variation of one of the queries that used vocabularies present in secondary data sources we saw a major boost in performance, while performance was still nowhere near performance under ideal conditions (the retrieval scenario) it made a much larger part of the data usable. This further reinforces our conclusion from the previous paragaph that domain knowledge has a major role to play in providing effective link integrity. Thus the problem becomes not whether we can develop techniques for providing link integrity but how do we obtain the necessary domain knowledge without having to rely on user expertise and we discuss this as a possible avenue of future work in Section 8.3.3.

With regards to part 2 of our hypothesis we need to talk a little about scalability as well as viability which we have already covered. In our hypotheses we defined scalability to be that the technique is still effective on datasets larger that 10,000 URIs and we have evaluated one dataset - ECS people - which was above this size. With the experiments we have conducted it is actually difficult to make any real assertions about the real world scalability of our approach since we did not factor this into the experiments in any way. We did observe that the retrieval of data for this dataset took the longest of all the datasets - on the order of a couple of days per algorithm - though this was expected since larger numbers of URIs should typically require more lookups and more processing time.

One final point that we must highlight in relation to scalability is that our experiments were what we'd consider an atypical usage of our approach. The typical expected usage is that a client would use this approach to recover data about a few specific URIs that they were interested in rather than trying to use it to recover an entire dataset. If the client only needs to recover a few URIs this should always be relatively quick and efficient. Scalability only starts to become an issue when you wish to retrieve large number of URIs and if they all originate from the same server. Though we cannot make any conclusive statements about the scalability of this approach based on these experiments, we do discuss some general observations and problems with scalability related primarily to data retrieval later in Section 8.2.3.1.

Chapter 8

Conclusions and Future Work

In this thesis we have introduced the concept of link integrity as it applies to hypermedia and the Semantic Web and discussed the variety of attempts to provide a solution to the problem. In this vein we have proposed our own solutions for the Semantic Web based upon the adaption of existing solutions proposed for hypermedia. Throughout our work none of the solutions we have discussed or developed ourselves are perfect for many different reasons. In this chapter we present our conclusions on our own work and discuss some of the wider issues around the problem. Finally we present a number of ideas for future work that could be undertaken to extend our work and further our aim of providing scalable and viable link integrity to the Semantic Web.

8.1 Our Research

In our research we have presented our adaptations of two link integrity techniques from hypermedia to the Semantic Web. In doing so we were looking to prove the two parts of our hypothesis, see Section 1.1 for more detailed definitions of these but we summarise them again as follows:

1. Existing approaches from hypermedia research can be adapted to the Semantic Web and shown to provide some level of link integrity.
2. That approaches can be demonstrated to be sufficiently viable and scalable that they could be deployed in real production scenario.

It is our view that we have been successful in proving part 1 of our hypothesis since in Chapters 5-7 we have shown these systems working on the Semantic Web with varying degrees of success. While our experiments have shown that our approaches are not able to provide link integrity effectively for every dataset they have clearly demonstrated that with the right conditions providing link integrity for the Semantic Web is possible. The fact that we could not provide link integrity for every dataset is not just due to our approach but rather to the nature of the Web as a whole. There will always be some data that is poorly linked and that exists only in isolation that will be difficult for any approach to provide link integrity for. Ultimately link integrity is about links and many of the possible solutions involve exploiting the links that do exist. If there are very few links then this task becomes significantly more difficult. This is not simply us trying to justify our failure in creating a universally applicable solution, rather it is an argument that has been advanced and supported by other research in this field, such as Phelps and Wilensky (2004) and Harrison and Nelson (2006), upon which our own research is based.

In terms of part 2 of our hypothesis we have only been partially successful in demonstrating the scalability and viability of such systems. We saw that its replication approach of All About That (AAT) did allow us to successfully and effectively preserve content so that it could be used in place of the original content in the event of link failure, but we found that it has serious scalability problems. As we discussed in Section 5.3 this was caused by two main factors. Firstly there was the fact that the triple stores we used exhibited suprisingly poor performance mainly due to the way that our system used them, and secondly the overheads involved in regularly retrieving and storing large quantities of data from the Semantic Web. Though as we outlined these can be mitigated in various ways (e.g. better triple stores and more parallelisation) these workarounds can only avoid the problems to a limited degree and it was clear that as designed our system simply wasn't scalable enough. To some degree we were setting ourselves a very high bar for scalability and there may perhaps be many small users who would be quite content with the minimal level of scalability we were able to demonstrate in our experiments. However our goal was to provide link integrity for the Semantic Web on as large a scale as possible and it became apparent that large scale replication was not a realistic approach for achieving this, hence why we did not undertake further research in this direction.

It should be noted that replication is a workable way to provide link integrity but only if the replications can happen at fairly large intervals. Especially since large intervals between replications would reduce the resource costs needed to run them down to the point where they could be run on a single machine (or a small cluster of machines). In the scenario where the freshness of data is not the driving factor in using such a system and it is acceptable to the user that the replicated versions of the data may be weeks or months old, this will be the more sensible strategy to employ. This has already been shown to work on the Web where projects like the Internet Archive[1] have employed it with great success. Although they are attempting to replicate a vast swathe of the Web they are only concerned with having one or two versions of the content per year. For some consumers of linked data that level of replication may be acceptable but for many data accuracy is a major concern and replicated copies that were months old would be completely unacceptable.

The expansion algorithm exhibited much better scalability primarily because it does not need to regularly retrieve data as AAT does, and need only perform data retrieval at the time when a link fails. This allows our technique to scale to reasonably large datasets (10,000+ Uniform Resource Identifiers (URIs)) on a single machine, though we must note when it is applied to the entirety of a large dataset in one go it still takes appreciable time (i.e. hours to days). As stated earlier, this should not typically be as much of a problem for this approach as the expected use case is that a user has a few specific URIs for which they need to retrieve data rather than them trying to recover an entire dataset. There is one rather interesting approach that we could use to improve the scalability of our algorithm that we discuss in Section 8.3.2.

Unfortunately as we have already mentioned in the conclusions of the Chapter 7. it is apparent that the expansion algorithm is not a viable approach to link integrity for the Semantic Web in its current form. This is due to the fact that we have seen very mixed results in our experiment which have particularly highlighted the fact that domain knowledge can make the difference between our approach being useful or useless. Our main problem is that a client often needs domain knowledge for the algorithm to perform effectively. This reduces its viability since it then relies upon a user providing that knowledge. As we have already stated several times this limits viability since a user will typically only be an expert in a few datasets and so can only configure the algorithm

[1] `http://www.archive.org`

to work with those. Thus to apply the technique to a variety of datasets you require a variety of domain experts. In Section 8.3.3 we discuss whether it would be possible to automatically discover the required domain knowledge and what the challenges in doing this would be.

The secondary problem that we have encountered is that some datasets seem to be poorly suited to data recovery regardless of whether we employ relevant domain knowledge or not. Interestingly this is similar to an issue raised by Harrison and Nelson (2006) and Phelps and Wilensky (2004) that certain types of resources on the Web are poorly suited to recovery and that this is an artifact of the specific approach. So for example in both of their papers it is noted that both non-textual content and content where the textual content changed frequently could not be recovered using their approaches. This implies that however much we refine and improve our technique there may always be datasets for which our approach will not work. In Section 8.3.3 we discuss whether it would be possible to automatically determine whether a dataset is suited to our technique or not and warn users when it is not.

Overall we are forced to conclude that while we have proven part 1 of our hypothesis that link integrity solutions from hypermedia can be applied to the Semantic Web, we have not been successful in demonstrating a truly viable and scalable approach. As a result we must state that we have not successfully proven part 2 of our hypothesis, although as we have already implied in this section our approaches do show promising results. As suggested in this section they have the potential to be developed further into truly viable link integrity solutions for the Semantic Web and we outline some of the possible ways to do this in Section 8.3.

8.2 The Link Integrity problem

Link integrity is an important issue to take into consideration and the problems it creates are widely prevalent on the Web and the Semantic Web today. In this section we make some more general conclusions relating to the general problems faced in attempting to address link integrity and the question of whether and how these might be resolved.

Firstly in Section 8.2.1 we discuss the problems that arise from use of data that may be incomplete, inaccurate or otherwise out of date because it has been retrieved by a link integrity mechanism - whether our own or one of the many systems we have discussed in our research. In the subsequent sections we summarise the overriding themes that we have seen in our discussions of the background to this problem in Chapters 3-4 as well as throughout our own research:

- That unfortunately many users and publishers on the Web and the Semantic Web are either unaware of the problem or they do not care about it.
- That the sheer scale of the Web and the Semantic Web provides numerous technological barriers to getting any solutions to the problem deployed.

We discuss the lack of social awareness in Section 8.2.2 and the technological barriers in Section 8.2.3. The latter of these sections includes analysis of general limitations to certain approaches to link integrity and so we reiterate some of our conclusions about our own work from the preceding section to help contextualise the analysis.

8.2.1 Issues in using partial, incomplete or stale data

One issue that we have briefly touched upon in relation to our work, but which applies to link integrity as a whole, is that there are implications around the use of data recovered by link integrity solutions. The first and most obvious of these is how do you make clients aware that the data they are using is not necessarily complete or up to date? If one was providing a link integrity service then it would be reasonable to expect that clients should be aware of the intention of the service and thus that the data is not complete. In the case where one provided some other form of Semantic Web service which used link integrity only as a means to fix errors in retrieving the data it was using how is the client informed that the data that is served to them may not be complete or up to date?

In essence this is a specific use case for data provenance on the Semantic Web which is unfortunately still very much an open issue and outside the scope of our research. As we mentioned in Section 5.1.1.1 one way of expressing this information is to use one of the available provenance vocabularies such as the Provenance Vocabulary (Hartig and Zhao, 2009) and the Open Provenance Model (Moreau et al., 2007). Another way is to use named graphs and then have another meta-graph that describes the provenance of that graph. The problem is that there is no universally agreed standard way of expressing or managing provenance for Resource Description Framework (RDF) which is why there is currently a World Wide Web Consortium (W3C) provenance working group[2] discussing this in addition to ongoing academic research in this field. Without a standard way of expressing this information we cannot guarantee that a client consuming our data will understand whatever provenance information we choose to include, assuming of course that they choose to inspect any such information at all. Additionally we have to factor in the issue of trust on the Semantic Web which is closely linked with data provenance and again is very much an open research issue. Clients may trust data from source **A** but not from source **B** so unless the data we provide includes provenance information how does a client know which parts of the data to treat as trustworthy and to act upon?

Even if we were to have a reliable and widely agreed way of expressing the provenance of the data returned by link integrity systems this would only solve the issue when the client is accessing the data directly from a system that includes this data. We must be mindful of what happens when a third party goes onto to republish the data in some form? Unless they are correctly propogating or maintiaining their own provenance data a client using their data may be completely unaware of the fact that the data they are getting is incomplete or out of date. In use cases where the client is just retrieving the data for display this may not really be an issue as the end user viewing that data might not be concerned with the accuracy of the data. Yet as the Semantic Web evolves and if it is ever to fulfill Tim Berners-Lee's original vision (Berners-Lee, 1998b) of agents able to make decisions, then that ability may be adversely impacted if the data they are retrieving is inaccurate.

In this vein imagine an agent which makes decisions as to whether to buy or sell stocks based on data retrieved from the Semantic Web. If that data is inaccurate it may cause the agent to make a decision that would be financially damaging to the user or the organisation on behalf of which it trades. In a domain like stock trading data which is even marginally out of date would be totally unacceptable unless that data was historical data examined only to evaluate trends over time. The point we are making here is that using link integrity solutions can actually be harmful in some scenarios so anyone deploying one must carefully consider whether it is appropriate. Our advice would be that for any scenario where outdated data is unacceptable or potentially harmful it

[2]http://www.w3.org/2011/prov/wiki/Main_Page

would be better to simply fail or do nothing because there is no data available rather than attempt to recover data or to use replicated data.

Another problem inherent in using incomplete data is that it may significantly increase the burden of data consumption on the client because of the need for additional domain knowledge. As we saw in the results of our experiments evaluating our expansion algorithm in Chapter 7, having some domain knowledge can make the difference between the recoverable data being accessible and usable or inaccessible and useless. For example with the DBPedia Countries dataset in Section 7.3.2 we saw that in order for the data from our algorithm to be usable for answering queries in the recovery scenario it was necessary for us to rewrite our query so that it used domain specific knowledge. In many cases this burden may cause clients to decide not to use such data at all because the additional difficulties in using it outweigh the benefits of having the data in the first place.

8.2.2 Lack of Social Awareness

The lack of social awareness of the problem presents a serious challenge for anyone like ourselves hoping to provide solutions to it. Not only do you need to develop a viable and scalable solution which poses its own challenges (see Section 8.2.3) but you then need to convince a sufficient mass of users to adopt the solution or you haven't really solved anything in real terms.

Though our work has focused primarily on just providing the technological means to solve the problem we cannot ignore this social aspect. Let us look at this problem from the perspectives of both users accessing the Web and publishers providing content on the Web.

- **User Perspective**

  The typical user perspective from what we have seen in the literature would appear to be that they do not care about the link integrity problem. They are no doubt familiar with encountering dangling links but they probably do not realise that this is a recognised open research problem.

  As we discussed in the section on search engines and portals (Section 4.1.2) the abundance of these sites provides a simple way for users to try and find alternative sources of the content they were looking for. With most modern browsers integrating search engines directly into their user interfaces this is made even easier than it was in the early days of the World Wide Web (WWW).

- **Publisher Perspective**

  The publisher perspective though not typically mentioned in the literature but which we have elicited based upon our discussion of various proposed solutions, including our own, is that you are asking them to provide an additional feature. This feature is potentially costly to them and there is minimal user demand for it, therefore unless they can make a business case for it within their organisation they are most likely to dismiss providing link integrity out of hand.

  The only link integrity we can reasonably expect to see on a website is that provided automatically by the content management system (CMS) running the website e.g. the way MediaWiki[3] highlights links that are to non-existent pages.

For us as advocates of solving the problem this creates a vicious cycle. The publisher argues that they won't run any such solution because the users don't want it and the users aren't aware of the problem so don't ask for a solution. With this in mind the sensible approach would seem to be to look for areas in which link integrity is of particular significance or importance to the parties involved.

The obvious example of this would be digital libraries as evidenced by Ashman (2000) where the author talks about the problem from the perspective of that area. Digital libraries, particularly those created by academic institutions, are often intended to become permanent records of the materials contained within them. If those materials are hypermedia then ensuring the links between them remain valid over time is often a core requirement of a digital library project. As we have seen in our discussions of early and open hypermedia (see Chapter 3) in a closed system it is far easier to ensure link integrity. As such a feature can be built directly into the system there is not the same social problem of raising user awareness. With the system managing link integrity in the background users and publishers need not be aware of it thus completely bypassing the social awareness problem.

This leads us to suggest that a hypothetical ideal link integrity system would need to

[3]http://www.mediawiki.org

work behind the scenes as far as possible. This comes back to the notion of viability we introduced in our hypothesis (see Section 1.1). For a system to be truly viable it should require minimal user involvement to configure and utilise, unfortunately from what we have seen of existing systems and our own systems this is not currently achievable. Providing a system for an open distributed system like the Semantic Web that can ensure link integrity without requiring any user intervention is not something that anyone has yet suggested a viable solution for and we would suggest is practically impossible. All the solutions we discussed in Chapter 4 and those we proposed ourselves are reliant on some level of user involvement.

Another issue that we must consider is that while a system that is entirely invisible to users might sound ideal, it has a serious flaw which is why we believe such a solution is not realistically possible. This flaw is that users don't like not knowing what their browser or client is doing when they access the Web, users typically do not like being silently redirected from one site to another. This is colloquially evidenced by the raft of pages on the web that tell users before they get redirected and give them the chance to opt out of that redirection. If we were to attempt to deploy automated Just-in-Time (JIT) based link integrity mechanisms on the Web or Semantic Web it would no doubt require some mechanism along these lines.

For the Semantic Web such a mechanism creates another issue in terms of data provenance, if the redirect is completely silent how does a client know that when it attempted to retrieve data from URI **A** it actually cames from URI **B** instead? This may be very important to the client since they may trust data from **A** but not from **B** or wish to process the data differently depending on its source.

In providing link integrity we cannot attempt to subvert the existing behaviour of the Web (whether technological or social) to do so or we will never convince users that link integrity is something they want. If we do so we are more likely to alienate users against the idea and find them campaigning against the introduction of such systems.

One interesting point to consider is whether social awareness of link integrity would drive adoption if a major player on the internet was seen to widely and publicly adopt it e.g. Google, Facebook etc? A major player would be capable of convincing users and publishers to adopt the available technologies because they could provide incentives for people to participate. For example they could guarantee that your search results always pointed to relevant pages on your website or that links you shared socially always worked. As we said earlier, if there was a clear business case for providing link integrity then publishers would start implementing it and promoting it to their users. This is perhaps pure speculation on our part and it is unlikely that such a player will ever choose to adopt such a technology. Primarily because of the scale they would need to deploy it at just to cover their own online assets before they get into the quantity of resources required to expand deployment of such a system to wider portions of the web.

8.2.2.1 Awareness in the Semantic Web community

A key difference between the general Web community and the Semantic Web community is that the latter has been much more proactive in discussing the issues of link integrity on the Semantic Web, particularly in the context of vocabularies and linked data. In the first case it is considered important that vocabularies have stable URIs because the primary motivator of using vocabularies is that you use standard URIs for concepts and relationships so that a user can dereference those URIs to get further information. If a vocabulary is moved to a different location the terms become ambigious because a user can't directly look them up to see what their intended definition was. This has led to a lot of vocabularies using the PURL service (see Section 4.1.3.2) to give themselves a permanent URI while allowing their authors to migrate their actual URI as necessary without breaking links. As we have highlighted in our earlier discussion of PURLs and other similar systems this does not solve the problem simply moves it since if the service used fails then you still have broken links.

Linked Data is the other primary motivator for users of the Semantic Web being concerned with link integrity since the movement is predicated upon the linking of data together and for these links to be useful they need to work. While linked data was initially bootstrapped by academics it is now being widely adopted by governments and corporations as a way to make their data available in a standardised way e.g. US Government[4], BBC[5] and BestBuy[6]. In the case of corporate publishers there is often a financial benefit to linked data (e.g. BestBuy seeing a 30% boost in traffic to their online store[7]) so ensuring it remains a viable technology is in the interest of the publisher.

[4]http://data.gov
[5]http://www.bbc.co.uk
[6]http://www.bestbuy.com
[7]http://www.readwriteweb.com/archives/how_best_buy_is_using_the_semantic_web.php

While broken links on the corporate website may be an inconvinience for users they tend not to harm a company's bottom line, broken links on the Semantic Web might mean that your products don't appear near the top of search engine results causing loss of customers. Thus in this case we avoid the problem we highlighted earlier, publishers aren't waiting for users to ask them for link integrity as they already have commercial incentives to ensure it.

8.2.3 Technological Barriers

8.2.3.1 Data Retrieval Overheads

An issue we have raised several times throughout our research is that providing a link integrity system may entail substantial resource and infrastructure costs. This is particularly true for replication based solutions which rely on storing local copies of the data and therefore as the quantity of data one wishes to replicate increases so does the amount of storage required. In reality larger storage volumes are becoming ever cheaper and one can easily create a system with vast amounts of storage at very little cost which negates this problem if you are only looking to do small scale replication i.e. a few key datasets you have an interest in. However, if you want to do replication at Web scale by replicating large numbers of datasets then it is likely to require vast amounts of storage which can still be very costly.

The primary barrier to scaling replication based approaches is the actual costs of retrieving the data. This is the data retrieval overhead we first highlighted back in Section 5.2.1.1 in the context of our AAT system for replicating linked data. The problem is that there are both technological and social overheads associated with retrieving data from the Web that make scaling link integrity solutions difficult. Note that these overheads are not limited to replication based approaches (though they are more apparent in those approaches) but also affect recovery based approaches since if you want to recover an entire dataset you will incur similar levels of overhead to a replication approach.

The technological overhead of providing link integrity, whether for the Web or the Semantic Web, is the requirement that you make Hypertext Transfer Protocol (HTTP) request(s) to retrieve data at some stage. Every request one makes is subject to overheads in terms of time for network transit and processing both on the server side in producing a response to the request and on the client side in processing that response. On a modern network connection with the speeds achievable today the network transit time may typically be negligible and therefore the processing time becomes the key factor.

This can be affected by multiple things such as whether the server is simply returning flat files or whether it has to do some processing as the content is dynamic e.g. RDF generated from a Relational Database Management System (RDBMS). Even if this processing time is minimal, say an arbitrary 100ms per request, for every 10 requests you make you experience a 1 second overhead. As you scale up the amount of requests you need to make these overheads quickly mount up, 1,000 requests results in a 1 minute 40 second overhead, 10,000 requests a 16 minute 40 second overhead and so on. We must be mindful that these are speculative figures and depending on the servers involved may be much lower or higher. Also there are other factors that may affect this overhead such as how powerful the server is and how many other clients are trying to access the server at the same time.

The server processing is not the only processing overhead involved, a link integrity system will not be making the requests and simply throwing away the results it will be processing them appropriately. Even if the system is only storing the response as-is to files on disk this will require some time to do, any system that involves storage of large volumns of data is going to be hampered by the fact that IO is typically the slowest part of a modern computer. Also we must take into account the fact that most systems, including our own, need to do more than just store the response as is. For example our AAT system has to perform a variety of data processing on the retrieved data in order to update its encoding of the data, the computations performed can take considerable time particularly as the size of the retrieved data increases.

In terms of the social overhead involved in data retrieval there is the social etiquette around retrieving data from the Web. One cannot make unlimited numbers of requests to a server since doing so will quickly get the client blocked from that server as it would be equivalent to performing a denial of service (DOS) attack upon that server. Once a client has been blocked it cannot retrieve any data from that server thus making it much harder, if not impossible, to provide any form of link integrity for data originating from that server. Therefore a client must insert delays between each requests in order to avoid this which adds additional and tangible overheads to the data retrieval process. Typically a 1 second delay is considered to be a reasonable delay between requests to a single server which could quickly mount up as the number of requests made increases. It is possible to mitigate this problem by spreading the requests over different servers but this only works to a certain degree. Unfortunately as we saw with our AAT system this is not always possible if all the data to be retrieved is from a single server, in such cases then a client will be forced to respect this delay regardless.

The only realistic workaround for this scalability problem is to have a cluster of machines each with their own unique Internet Protocol (IP) addresses and each running a copy of whatever data retrieval algorithm or client is to be used. By having multiple IP addresses the cluster can carry out many more simultaneous requests to a server and thus retrieve more data simultaneously. However this does not free a system from the social etiquette of crawling entirely. Even in a system with multiple machines if too many of them are accessing one server at once this is still effectively a DOS attack and the system may get an entire swathe of its IP addresses blocked rather than just one. With this in mind one way to avoid this is to create a crawler specifically designed to run in a distributed environment with the different machines coordinating between themselves to limit simultaneous access to individual servers. Though this mitigates the problem it adds additional coordination overheads into the system and its scale is still limited by the number of distinct servers that data will be retrieved from. As we stated previously if the data comes from a single server or only a few servers then adding extra machines will only scale so far. Eventually a system will reach a point where the additional machines become redundant since it will already be making too many simultaneous requests to the servers of interest.

As stated in Section 5.2.1.1 this social etiquette in retrieving data from the Web which developers of such systems are obliged to respect if they want to build a reliable system is a key limiting factor in the scalability of any such system. However many servers and crawlers a system has the amount of requests it can be making at any one time is fundamentally limited by respecting this. We envisage the typical usage of a system like our own AAT to be the monitoring of a small number of datasets that the user is interested in, therefore systems are always going to run into this problem. There is no easy workaround for this problem and both those implementing and using such systems must be aware of the the issue.

In the broader crawling use case where you attempt to index or replicate the entire Web or large portions thereof this issue tends to become irrelevant as a systems crawlers will always have some server to which they can make a request to. This implies that perhaps we are not thinking big enough. If we truly want to provide link integrity for the Semantic Web then we should be advocating the Sindice[8] approach of attempting to index as much of the Semantic Web as possible. If you were in possesion of a large scale corpus of Semantic Web data and were routinely crawling it as a matter of course providing the value added services like link integrity on top of the basic search style capabilities would be relatively simple. Services like Sindice have offered a cache Application Programmers Interface (API) in the past (and still do though in its current form RDF cannot be directly retrieved) and this is exactly the kind of service that would provide the data to make the expansion algorithm and similiar approaches much more viable a solution to link integrity. As we will discuss in the subsequent section there is unfortunately a lack of such services available on the Semantic Web and we believe these are essential to providing effective link integrity for the Semantic Web.

One final thing to highlight is that these overheads primarily only apply in the scenario where a system is attempting to provide link integrity for large quantities of URIs all at once. Though there may be some clients which would want to use our approaches to provide link integrity for entire datasets the more typical use case we envisage is of a client using it to get link integrity for a few URIs at a time that are relevant to the current task or query. In this scenario the overheads would not really be an issue because if the client needed to recover or replicate only a small number of URIs the total overhead of data retrieval would be relatively negligible.

Despite this last proviso we must still conclude that in their current forms both of our approaches, in fact any system that takes a similar approach, are ulitmately limited in their scalability by both the technological and social factors we have outlined in this section. Unless your system is deployed within an organisation like Google, Bing or Sindice whose core business is the crawling and indexing of the Web, the costs of scaling such approaches to Web scale will typically be out of reach. This is not necessarily a barrier to providing link integrity on a smaller scale as often a client may be interested in link integrity only for a small number of datasets, or for a particular portion of the Semantic Web and these approaches are scalable with relative ease for such use cases.

[8]http://www.sindice.com

Nevertheless we have to conclude that overall we have failed to meet with part 2 of our hypothesis with regards to scalability. While we can state that our approaches will scale to datasets comprising 10,000+ URIs we are unable to show that they scale on a larger scale than this. Also our analysis suggests that doing so is only possible with the use of multiple machines and this clearly conflicts with our stated aim of having such a system be runnable on a single machine. So arguably our stated aim was a simplistic and arbitrary limitation to apply but our original reasoning in our hypothesis still stands. If link integrity is to ever see widespread adoption it must be able to work with limited resources so that is available to as many users as possible.

8.2.3.2 Availability of Services

One problem that we encountered in developing the solutions presented in our research, and particularly in terms of our recovery approach, was that we were designing solutions around using services on the Semantic Web that either didn't exist or had limited capabilities. The prime example of this is our concept of a discovery endpoint that is quite central to the behaviour of the expansion algorithm, yet we are still only aware of two services specifically designed for this purpose. Even with the relevant services being available to use it is still very hard to build an effective system since we are relying on those remaining stable and usable. These services may simply be discontinued from one day to the next or they may suffer from intermittent faults due to lack of processing capacity, network issues etc.

An example of such a service that caused problems is the Sindice Cache API[9]. As described in Vesse et al. (2010) this API was one of the core services that we used to obtain data and test the initial prototypes of our algorithm with. We chose to use this service because it provided the same kind of cache service that Harrison and Nelson had used in their JIT approach (Harrison and Nelson, 2006) to link integrity which we were aiming to apply to the Semantic Web. In our initial testing the service worked impressively well but a few months later when we were conducting more extensive testing we found that they had changed their API in such a way that we were no longer able to use it. This lack of stability in services makes it very difficult for a system such as ours to work effectively because it relies on using them but there are few stable ones

[9] `http://www.sindice.com/developers/cacheapi`

available. As we suggested in the preceding section if such services were more widely available our expansion algorithm would be a far more viable a way to provide link integrity. Unfortunately without good general purpose services like this an algorithm is often reliant on having good domain knowledge as we saw with the British Broadcasting Corporation (BBC) programmes dataset.

Somewhat frustratingly when we were conducting the second round of experiments on our recovery approach the Sindice developers had by that point launched a SPARQL Protocol and RDF Query Language (SPARQL) endpoint that could be used to query their entire catalogue of RDF documents. This meant that their service could be used as a pure SPARQL endpoint to issue `DESCRIBE` queries against and get essentially the same data as we used to be able to get from their cache API. Obviously in the interests of fairness and making legitimate comparisons between the results we could not add this endpoint to our default expansion profile. As was seen in our discussion of the Drugbank results in Section 7.1.3 we did run a variation on that dataset using this endpoint and it actually worsened the performance. This is rather counter-intuitive since the use of a service that effectively provides a cache of RDF should in theory improve performance. Since this was an unexpected result we manually inspected the results of submitting some random example URIs from our datasets to this service and observed the RDF we obtained from a `DESCRIBE` query was often far more limited than the RDF a human user could view on their website.[10]

8.3 Future Work

In this section we discuss a number of possible avenues for future work that would advance our own research or are spin-off ideas motivated by issues that our research has raised. Firstly we look at several avenues of future work that would extend the work we have presented. In Section 8.3.1 we discuss possible enhancements to our replication based approach to link integrity, then in Sections 8.3.2-8.3.4 we discuss both extensions to our recovery based approach and useful systems we envisage being built upon the foundations provided by our work.

[10]Please bear in mind that Sindice offers this SPARQL service as a Beta and makes that clear in the documentation, completeness of results appears to have improved significantly since the time of our experiments

8.3.1 Enhancements to AAT

As we discussed in Chapter 5 while we have asserted that the replication approach to link integrity and our AAT system are not truly scalable or viable in providing link integrity for the Semantic Web, this does not mean that the replication approach is fundamentally flawed. With the right infrastructure and architecture it is possible to build effective systems for both the Web and the Semantic Web as evidenced by work such as the Internet Archive[11] and Memento (Van de Sompel et al., 2009), but these systems are primarily interested only in replicating content at large intervals (typically months). In our experiments we focused upon the idea of being able to replicate regularly changing content where the content may change every day. This poses different scaling challenges, particularly in the area of crawling etiquette as discussed in Section 8.2.3.1, which other systems are able to mitigate simply by spreading their crawling over long periods of time. Despite these problems there are some areas in which AAT could be improved which would reduce the effect of these problems upon the system and enhance its ability to provide replication based link integrity.

The first major improvement that could be made would be to redesign the system so that it is a multi-server application rather than a single server as in the prototype we developed. A multi-server system would scale to larger datasets as the work of retrieving and storing the data can be shared across multiple servers, and it would be able to mitigate some of the data retrieval overheads we discussed in Section 8.2.3.1 that a single server solution encounters. Alongside this change it would be desireable to experiment with using alternative triple stores from the ones we originally developed against. Triple stores have matured considerably during the course of our research and we suspect that by using one of the newer and more performant stores e.g. AllegroGraph 4[12], BigOWLIM[13] or Stardog[14] that are designed to deal with more read/write centric workloads, we would expect to see much better performance than we did in our protytpe. We should also note that the RDF and SPARQL API (dotNetRDF[15]) we used to build this prototype has matured significantly since this prototype was built, particularly in the area of SPARQL performance which we use extensively, and so we would also benefit

[11] `http://www.archive.org`
[12] `http://www.franz.com/agraph/`
[13] `http://www.ontotext.com/owlim`
[14] `http://www.stardog.com`
[15] `http://www.dotnetrdf.org`

from those improvements in any future version of our system.

A variation on this multi-server improvement would be to adopt a genuinely distributed system as opposed to just a multi-server system that simply runs on a cluster machine. In a similar vein to our suggestions for scaling the expansion algorithm in Section 8.3.2 one could envisage having a network of servers spread out across the internet which would coordinate together to replicate data that users were interested in. By distributing the system on a much larger scale resource usage could be reduced rather than having a single server (or cluster of servers) that is run by individual user(s) for their specific needs you would have a network of general purpose server clusters. As a result rather than each individual system having to retrieve and replicate all the data it needs it could query other servers in the network and retrieve existing replicated data from them significantly reducing the need for data retrieval. Also the servers could schedule the regular data retrievals needed to update the replicated data across the network taking into account factors such as bandwith, storage, processing power etc. available to each system and user utilisation levels of each server. So for example a powerful server with lots of storage that was rarely accessed directly by users might do the majority of the retrieval and storage work and could act primarily as a storage and processing node in the network. Conversely a lower powered server with access to a lot of bandwith might be accessed by large number of users and thus could act primarily as a proxy for those users passing on their requests for data to servers in the network that had more spare resoures. Obviously creating such a system would be a non-trivial and complex task that would require multiple organisations to support the creation and maintenance of such a network. Therefore we do not envisage any such system being deployed by major businesses on the Web, but rather by academic institutions and libraries which have an interest in replicating and archiving digital data whether RDF or otherwise from the Web.

A more minor interoperability motivated improvement would be the use of alternative RDF vocabularies, as we explained in Section 5.1.1.1 we chose to use a fairly simple but custom vocabulary for storing the replicated RDF data primarily for technical implementation reasons. However there are several community created provenance vocabularies that could have been used instead e.g. the Provenance Vocabulary (Hartig and Zhao, 2009) and the Open Provenance Model (Moreau et al., 2007). Therefore one area in which AAT could be improved is in incorporating these vocabularies, even if only as an alternative output format. The benefit of doing this is that it would make the replicated data stored within AAT more portable to other provenance aware Semantic Web applications. The primary reason that we never implemented this originally is that it does not really have any value if you are using the system only to provide link integrity. Where the use case for this lies is when you want to use AAT as a form of cache to the Semantic Web and it is important to know where and when the data came from. In essence this is similar to the problems we outlined earlier in Section 8.2.1 that if you intend to use such a system to provide data to clients you should make them aware of the provenance of the data so they can treat the data as appropriate for their use cases.

8.3.2 Scaling the Expansion Algorithm

One issue that has cropped up repeatedly, particularly when discussing our expansion algorithm, is the scalability of the approach. As we have already discussed in Section 8.2.3.1 there are a variety of social and technological overheads which limit the scalability of such an approach whether running as a single or multi-threaded operation. Despite these barriers there is one potential scaling mechanism that we have yet to discuss which has the potential to make our algorithm a much more viable and scalable approach to link integrity. The basic idea is that rather than having all the work be done on the client that is doing the data recovery you distribute the work over a network of link integrity servers. This idea is inspired both by the distributed link integrity mechanism proposed by Kappe (1995) and by a future work suggestion from Harrison and Nelson (2006) whose Opal system was the inspiration for the expansion algorithm.

In practise how this would work is that there would be a network of publicly accessible link integrity servers each of which would provide the expansion algorithm as a web service to clients. When a client wanted to use the expansion algorithm they would route the various requests for data via this network of servers and then merge the response back together on the client side to give the final result. While on paper this sounds like it would actually increase the overheads because it requires additional network communication and some level of distributed coordination. Therefore we believe that in reality it would actually improve performance for the following reasons:

- **True Distribution**

Providing that all the link integrity servers are deployed in geographically distinct locations - perhaps hosted by big companies, institutions or on various cloud providers - then the system will have a large pool of IP addresses to choose from. This will mean that there is little need for coordination to avoid DOS attacks on servers because the requests will be sufficiently spread out to not look like an attack. That is of course providing that the number of link integrity servers used is appropriately limited so that you don't still end up with vast numbers of simultaneous requests to the same server.

- **Data Caching**

  If the network is used regularly individual servers will each build up their own caches of data that they can use to speed up requests since they will need to make less new requests of certain kinds. Note that caching would have to be done carefully to ensure that stale data wasn't unnecessarily used, but this has the potential to speed up the data retrieval process as many requests may become virtual no-ops (i.e. there will be virtually no time cost associated with them).

- **Faster Network Access**

  If the network is appropriately deployed such that all servers have access to very fast high bandwith connections to the internet this would eliminate most of the network bottlenecks that a single client might experience since the requests are spread out over multiple distinct high speed connections.

How difficult implementing this would be is hard to tell since there are lots of issues associated with building any such distributed system, not least of which is how you actually choose to distribute the requests over servers? Not to mention the problem of reliably knowing what link integrity servers there are available for use. In principal one could probably use a protocol similar to Kappe's p-flood algorithm (Kappe, 1995) to propogate information about available servers across the network and provide it to clients. Additionally you could incorporate usage of the handles system (Sun et al., 2003a,b,c) to ensure that you are not reliant on hard-coded URIs for the link integrity servers and thus make the network itself less susceptible to link integrity problems. Regardless this is not a trivial enhancement to our algorithm and would involve significant research work to implement, but has the potential to lead to a much more viable and scalable link integrity system for the Semantic Web.

8.3.3 Automatic Domain Knowledge Discovery and Dataset Characterisation

One issue we have raised at various points throughout our work is that often solutions to link integrity rely upon the system having a degree of domain knowledge, by this we mean information such as possible secondary sources for data, relevant similarity and/or comparision metrics, crawling strategies etc. The problem with this is that it typically necessitates a human user in the loop since none of the systems we have discussed are able to discover this knowledge themselves. If we return to our definition of viability from part 2 of our hypothesis (see Section 1.1) we defined it as being the requirement for minimal user involvelment in a solution. Any system that relies on a user inputting domain knowledge will always be limited in its viability since a user with this knowledge is required to configure the system for each data source, but a typical individual user will often only have sufficient knowledge to do this for a few domains. This also implies that such systems are less scalable because the more data sources you want to provide link integrity for the more domain expert users you need to do this.

Therefore we feel that a useful avenue of future research would be to look into approaches for automatically discovering this domain knowledge. At its simplest one could envisage a system whereby a client (whether human or machine) provides an example URI and perhaps a SPARQL endpoint associated with it and from that goes off into the Semantic Web and discovers the relevant knowledge. While simple on paper and somewhat reminiscent of Tim Berners-Lee vision of the Semantic Web (Berners-Lee, 1998b) this is in practise likely very difficult to do. There are a number of issues any such approach would need to address:

- **Inputs Required**

  What inputs would be required for such a system to perform this automated detection? The inputs would need to be sufficient that the system could return a useful result but not so complex that the user essentially does the work for it i.e. if using the system is indistinguishable from manual configuration then it is arguably worthless.

  In essence what is needed is a system that with minimal information e.g. an example URI can derive useful information like SPARQL endpoints with relevant data, secondary data sources that link back to the original data source etc. The barrier to this is that there are few existing directories of such data sources and SPARQL endpoints, as a result their coverage is far from complete. Additionally there are no official standardised ways to express this information, although there are community standards and best practises in this area e.g. Vocabulary of Interlinked Datasets (VoID) (Alexander et al., 2011) these are not universally used by publishers making reliance upon them problematic.

- **Dataset Annotation**

  With the preceding point in mind there is a need to persuade data publishers to participate in the best practises of annotating their data using vocabularies such as VoID. In doing so we face the same problem of lack of social awareness as we discussed in relation to link integrity in general earlier in this chapter (See Section 8.2.2). Many publishers will need to be sold on some benefit to them of annotating their data in this way, plus we need to educate users to be asking this of publishers in order to make a case that it is a feature users wish to see.

- **Verification**

  Having performed its automatic discovery process how does a system (or the client utilising it) verify that what has been discovered is actually useful and not irrelevant? If we assume a system that works off a single example URI then the discovered domain knowledge may be highly accurate for that URI but we are looking to retrieve knowledge for a particular domain. Thus if that one URI proves to be a poor example of the domain the discovered domain knowledge may be highly inaccurate. In developing such a system one would need to carefully evaluate how different example URIs affected the results and whether it would be better to start from a selection of example URIs instead.

- **Refinement**

  Another potential issue is should such a system refine its knowledge over time? Do you ask it to discover domain knowledge once and trust its answer to remain valid over time or do you ask it to refine its answer periodically? This would be an interesting question for future research into such a system to attempt to address.

8.3.3.1 Dataset Characterisation

Even if we were to build such a system and have it work successfully there is a complication that our experiments have already highlighted. This is the fact that even with some domain knowledge our technique just does not appear to work well for certain datasets e.g. Drugbank (see Sections 7.1.3 and 7.3.3). What we have not really been able to address in our research because of the small number of datasets upon which we have evaluated our approaches is whether there are some defining characteristics of a dataset that determine whether it is a good candidate for data recovery.

As we stated earlier the obvious characteristics - whether concepts are well defined and widely agreed upon, interlinking etc. - of the datasets suggest that apart from the insular BBC programmes dataset all are obstensibly similar in nature. Therefore there is some work to be done in automatically characterising datasets and using these characterisations to make a judgment about whether you can successfully apply recovery based link integrity to a dataset. To do this would require repeating experiments like ours with a much wider range of datasets and then conducting detailed analysis of each dataset to characterise it. If you could do this manually without requiring too much user expertise in the process then it stands to reason that you could then automate the methodology in order to build an automated dataset characteriser.

If a user saw poor results this system could be used to determine whether their domain knowledge was not sufficient to provide effective data recovery or whether the dataset was poorly suited to the approach. This would be much more informative to users of our approach as rather than having to inform them that it just doesn't appear to work on a particular dataset we could tell them why. Either that the dataset is poorly suited to this approach, and if our characterisations were detailed enough why this was the case, or that they need better domain knowledge.

8.3.4 Link Integrity and Link Services for the Semantic Web

One area of research which we think warrants attention is the notion of providing link integrity and link services on-demand to clients. Instead of providing link integrity solutions that are reliant upon a user preparing them in advanced a more usable approach might be to provide it as a service. In doing so clients could get some degree of link integrity but only as and when they needed it, thus removing the potential resource burden from the user. As we have already suggested in Section 8.2.3.1 this kind of service could perhaps be provided by an organisation that already had access to a large corpus of Semantic Web data such as Sindice.

Beyond the obvious service of just providing a just-in-time recovery style service as we have attempted to do with our expansion algorithm there are also other link services one could provide that would allow for a more pretentative approach to link integrity. One example of such a service that already exists is the Okkam Entity Name Service (ENS) (Bouquet et al., 2008) which allows you to discover existing URIs that you can use for concepts in your data. The advantage of using such a service is that it allows you to reuse URIs and therefore reduces the possibility of broken links as if more people agree upon a URI for a particular concept there is hopefully more incentive for the original publisher of that URI to maintain it. Such services also help to solve the coreference problem and is in fact the problem that the (ENS) was originally created to solve but we feel that it and similar services have the potential to help link integrity in a wider sense.

In a more general sense beyond just link integrity we believe that there is a need for a variety of link services on the Semantic Web. This has already been suggested by leading linked data developers such as Leigh Dodds from Talis in one of his blog posts[16] and more widespread use of such services has the potential to lead to more reliable links across the Semantic Web. Unfortunately beyond sameAs.org[17] none of the services Leigh Dodds suggested and we envisage exist yet, or if they do they have not been widely publicised which would thus imply that people are not using them. Going back to the point at the start of this section the problem is that creating these services typically requires access to a large corpus of data so unless you are an organisation like Sindice that already has access to this setting up such a service is difficult. This again highlights both the problem of the data retrieval that we discussed in Section 8.2.3.1 and the bootstrapping problem that Harrison and Nelson discussed in the context of their Opal system (Harrison and Nelson, 2006). While creating a predicate link service would be extremely useful both for the Semantic Web in general and specifically for use in link integrity solutions like our expansion algorithm, doing so is not trivial and

[16]`http://www.ldodds.com/blog/2010/03/predicate-based-services/`
[17]`http://sameas.org`

warrants future research and development efforts.

Another form of general service which would also help link integrity are vocabularly mapping services. These are services that would take a vocabulary, or a specific vocabulary term, and provide alternative vocabularies or terms that could be used in its place. This has clear benefits for the Semantic Web as a whole in that it makes it easier for people to ensure that they express their data using vocabularies that are interoperable and widely known. In terms of link integrity this would help address the problem we saw in our query use case evaluation particularly for the DBPedia Countries and Drugbank datasets (see Sections 7.3.2 and 7.3.3) that although our algorithm can recover data because it is expressed in different vocabularies we cannot answer queries correctly. There are plenty of examples of techniques for mapping between ontologies available in the academic literature yet little of this appears to have been translated into real world services available for use on the Semantic Web.

The only real world example is the Alignment API[18] (Euzenat, 2004) which is a web service API that allows matching of ontologies. They key limitation of this service is that it is primarily an API as they themselves state on their website[19] i.e. if you wished to deploy such a service you'd have to create or obtain a matcher that implements the actual matching logic required yourself. As we saw with the DBPedia[20] Countries dataset in Section 7.3.2 when we modified our query to use vocabularies that we knew to be present, we saw significant performance improvements over using a query without this vocabulary mapping. While the results were nowhere near as good as those obtainable under ideal conditions (the retrieval scenario) they did clearly demonstrate the value of being able to map between vocabularies used in different datasets. The lack of such services makes our job of attempting to recover relevant data to provide link integrity all the harder. Though we may be retrieving useful data if the client does not have a way to automatically take into account different vocabularies then the recovered data may appear to be useless to them despite containing usable data.

[18]`http://alignapi.gforge.inria.fr/`

[19]`Please note: The Alignment API is... an API (this may be unfortunate, but it has been named appropriately). Hence, it is not a matcher`

[20]`http://dbpedia.org`

8.4 Final Conclusions

Link integrity has always been a problem on the Web and is ultimately an unavoidable by product of the decentralised nature of the Web, regardless of what method we use to prevent or repair such issues there is always the chance that we will encounter the problem because no solution we have seen/presented is perfect. Despite this we must still conclude that link integrity is a problem that is fundamental to the Semantic Web particularly in the area of linked data. If we truly intend to build a next generation Web based upon the interlinking of decentralised data and have applications built upon this, then we need to make a concerted effort to address the problem or at a minimum raise awareness so developers take account of it within their applications.

The research we have presented makes a significant contribution towards that goal in that it shows that link integrity can be provided for the Semantic Web using existing tried and tested techniques that have already been applied on the Web. As we have shown the actual techniques required are relatively simple, the primary problem in making these solutions effective are the social and technological barriers posed in part by the nature of the Web itself. The biggest of these barriers is community buy-in, although the Semantic Web community is more aware of the problems than the wider Web community convincing publishers and consumers that they need to address the problem. Although we have some level of awareness in our community we still have little or no action beyond a few scattered researchers like ourselves and the rare demonstration system (e.g. DBPedia Memento support). If we are to solve this problem we need to publicise it more and we need to improve upon the solutions we have proposed, because as discussed while they work to some extent they are not yet sufficiently mature for widespread deployment, nor do they work for every dataset they might encounter.

Although we are nowhere near declaring link integrity for the Semantic Web a solved problem, we have suceeded in demonstrating that it is a tractable problem with workable solutions that with further research can be addressed.

Bibliography

Virtuoso Sponger. Technical report, OpenLink Software, 2009. http://virtuoso.openlinksw.com/Whitepapers/html/VirtSpongerWhitePaper.html retrieved on Feb 4th 2011.

Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing linked datasets: On the design and usage of void, the 'vocabularly of interlinked datasets'. In *LDOW 2009: Proceedings of the 2nd International Workshop on Linked Data on the Web at WWW 2009*, Madrid, Spain, April 2009.

Keith Alexander, Richard Cyganiak, Michael Hausenblas, and Jun Zhao. Describing Linked Datasets with the VoID Vocabulary, 2011. http://www.w3.org/TR/void/ retrieved on 19th August 2011.

M. Andreessen and E. Bina. NCSA Mosaic: a global hypermedia system. *Internet Research*, 4(1):7–17, 1994.

F. Anklesaria, M. McCahill, P. Lindner, D. Johnson, D. Torrey, and B. Albert. The Internet Gopher Protocol (a distributed document search and retrieval protocol). RFC 1436 (Informational), mar 1993. http://www.ietf.org/rfc/rfc1436.txt retrieved on July 14th 2011.

Helen Ashman. Electronic document addressing: dealing with change. *ACM Computing Surveys*, 32(3):201–212, 2000.

Amit Bagga. Evaluation of coreferences and coreference resolution systems. In *LREC 1998: Proceedings of the 1st International Language Resources and Evaluation Conference*, pages 563–566, Granada, Spain, 1998.

Hal Berghel. The inevitable demise of the Web. *SIGICE Bulletin*, 21:19–22, October 1995.

T. Berners-Lee and D. Connolly. Hypertext markup language - 2.0. RFC 1866 (Historic), Nov 1995. http://www.ietf.org/rfc/rfc1866.txt retrieved on Feb 4th 2011.

T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext transfer protocol – http/1.0. RFC 1945 (Informational), May 1996. http://www.ietf.org/rfc/rfc1945.txt retrieved on Feb 4th 2011.

T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax. RFC 2396 (Draft Standard), aug 1998. http://www.ietf.org/rfc/rfc2396.txt retrieved on July 15th 2011.

T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), jan 2005. http://www.ietf.org/rfc/rfc3986.txt retrieved on July 15th 2011.

T. Berners-Lee, L. Masinter, and M. McCahill. Uniform resource locators (url). RFC 1738 (Proposed Standard), Dec 1994. http://www.ietf.org/rfc/rfc1738.txt retrieved on Feb 4th 2011.

Tim Berners-Lee. Cool URIs don't change, 1998a. http://www.w3.org/Provider/Style/URI retrieved on Feb 4th 2011.

Tim Berners-Lee. Semantic Web Roadmap, 1998b. http://www.w3.org/DesignIssues/Semantic.html retrieved on Feb 4th 2011.

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.

Chris Bizer, Richard Cyganiak, and Tom Heath. How to publish linked data on the web, 2007. http://sites.wiwiss.fu-berlin.de/suhl/bizer/pub/LinkedDataTutorial retrieved on Feb 4th 2011.

David Booth. The uri lifecycle in semantic web architecture. In *IR-KR 2009: Proceedings of the 4th International Workshop on Identity and Reference in web-based Knowledge Representation*, Pasadena, California, USA, 2009. http://dbooth.org/2009/lifecycle/.

P. Bouquet and H. Stoermer. OKKAM: Enabling an Entity Name System for the Semantic Web. In *Proceedings of the Semantic Interoperability: A Practical Approach Workshop at I-ESA 2008*, Berlin, Germany, 2008.

P. Bouquet, H. Stoermer, and B. Bazzanella. An entity name system (ens) for the semantic web. In S. Bechofer, M. Hauswirth, J. Hoffmann, and M. Kourbarakis, editors, *ESWC 2008: Proceedings of the 5th European Semantic Web Conference*, volume 5021 of *Lecture Notes in Computer Science*, page 258, Tenerife, Spain, 2008. Springer.

P. Bouquet, H. Stoermer, and D. Giacomuzzi. OKKAM: Enabling a web of entities. In *I3 2007: Proceedings of the 2nd International Workshop on Identity, Identifiers, Identification at WWW 2007*, Banff, Canada, 2007.

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

J. Conklin. Hypertext: An Introduction and Survey. *Computer*, 20(9):17–41, Sept. 1987.

G.F. Coulouris and J. Dollimore. Distributed systems: concepts and design. *Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA*, page 366, 1988.

Michael L. Creech. Author-oriented link management. *Computer Networking and ISDN Systems*, 28(7-11):1015–1025, 1996.

M.A. Cusumano, Y. Mylonadis, and R.S. Rosenbloom. Strategic maneuvering and mass-market dynamics: The triumph of VHS over Beta. *The Business History Review*, 66 (1):51–94, 1992.

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

Hugh C. Davis. Hypertext link integrity. *ACM Computing Surveys*, 31(4es):28, Dec 1999.

Ian Davis and Sam Tunnicliffe. Changeset protocol, 2007. http://n2.talis.com/wiki/Changeset_Protocol retrieved on Feb 4th 2011.

Norman Delisle and Mayer Schwartz. Neptune: a hypertext system for CAD applications. In *SIGMOD 1986: Proceedings of the 6th International Conference on Management of Data*, pages 132–143, Washington, D.C., United States, 1986. ACM.

Douglas C. Engelbart and William K. English. A research center for augmenting human intellect. In *AFIPS 1968: Proceedings of the Fall Joint Computer Conference*, pages 395–410, San Francisco, California, 1968. ACM.

Jérôme Euzenat. An API for Ontology Alignment. In Sheila McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *ISWC 2004: Proceedings of the the 3rd International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 698–712. Springer Berlin / Heidelberg, 2004.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2068 (Proposed Standard), jan 1997. `http://www.ietf.org/rfc/rfc2068.txt` retrieved on July 15th 2011.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), jun 1999. `http://www.ietf.org/rfc/rfc2616.txt` retrieved on July 15th 2011.

Roy T. Fielding. [httpRange-14] Resolved, 2005. `http://lists.w3.org/Archives/Public/www-tag/2005Jun/0039.html` retrieved on Feb 4th 2011.

Andrew M. Fountain, Wendy Hall, Ian Heath, and Hugh C. Davis. MICROCOSM: an open model for hypermedia with dynamic linking. In *ECHT 1990: Proceedings of the 1st European Conference on Hypertext*, pages 298–311, Paris, France, 1990. Cambridge University Press.

L. Nancy Garrett, Karen E. Smith, and Norman Meyrowitz. Intermedia: issues, strategies, and tactics in the design of a hypermedia document system. In *CSCW 1986: Proceedings of the 1st International Conference on Computer-supported cooperative work*, pages 163–174, Austin, Texas, 1986. ACM.

Hugh Glaser, Tim Lewy, Ian Millard, and Ben Dowling. On Coreference and the Semantic Web. In *ESWC 2008: Proceedings of the 5th European Semantic Web Conference*, Tenerife, Spain, December 2007.

Hugh Glaser, Ian Millard, Afraz Jaffri, Tim Lewy, and Ben Dowling. On Coreference and The Semantic Web. In A.P Sheth, S. Staab, M. Dean, M. Paolucci, D. Maynard, T. Finin, and K. Thurynarayan, editors, *ISWC 2008: Proceedings of the 7th International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, Karlsruhe, Germany, May 2008.

Jennifer Golbeck. Trust on the World Wide Web: a survey. *Foundations and Trends in Web Science*, 1(2):131–197, 2006. ISSN 1555-077X.

Frank Halasz and Mayer Schwartz. The Dexter hypertext reference model. *Communications of the ACM*, 37(2):30–39, 1994.

Frank G. Halasz. Reflections on NoteCards: seven issues for the next generation of hypermedia systems. *Communications of the ACM*, 31(7):836–852, 1988.

Frank G. Halasz, Thomas P. Moran, and Randall H. Trigg. Notecards in a nutshell. In *CHI+GI 1987: Proceedings of the SIGCHI/GI Conference on Human Factors in Computing systems and Graphics Interface*, pages 45–52, Toronto, Ontario, Canada, 1987. ACM.

Harry Halpin. Social meaning on the web: From wittgenstein to search engines. In *WebSci 2009: Proceedings of the 1st International Conference on Web Science*, Athens, Greece, 2009.

Terry L. Harrison and Michael L. Nelson. Just-in-time recovery of missing web pages. In *HYPERTEXT 2006: Proceedings of the 17th International Conference on Hypertext and Hypermedia*, pages 145–156, Odense, Denmark, 2006. ACM.

Olaf Hartig and Jun Zhao. Guide to the provenance vocabulary, 2009. `http://sourceforge.net/apps/mediawiki/trdf/index.php?title=Provenance_Vocabulary` retrieved on Feb 4th 2011.

B. Haslhofer and N. Popitsch. DSNotify–Detecting and Fixing Broken Links in Linked Data Sets. In *Proceedings of 8th International Workshop on Web Semantics*, Linz, Austria, 2009.

David Ingham, Steve Caughey, and Mark Little. Fixing the “broken-link” problem: the W3Objects approach. *Computer Networks and ISDN Systems*, 28(7-11):1255–1268, 1996.

Afraz Jaffri, Hugh Glaser, and Ian Millard. URI Identity Management for Semantic Web Data Integration and Linkage. In *SSWS 2008: Proceedings of the 3rd International Workshop On Scalable Semantic Web Knowledge Base Systems at ISWC 2008*, Karlsruhe, Germany, 2007. Springer.

Afraz Jaffri, Hugh Glaser, and Ian Millard. Managing URI Synonymity to Enable Consistent Reference on the Semantic Web. In *IRSW2008: Proceedings of the 3rd International Workshop on Identity and Reference on the Semantic Web at ESWC 2008*, Tenerife, Spain, 2008.

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

F. Kappe, K. Andrews, J. Faschingbauer, M. Gaisbauer, M. Pichler, and J. Schipflinger. *Hyper-G: A new tool for distributed hypermedia*. Number 388. Institutes for Information Processing Graz, 1994.

Frank Kappe. Hyper-G text format (HTF). Technical report, December 1993. `ftp.iicm.tu-graz.ac.at/pub/Hyper-G/papers/HTF.ps` not retrievable as of July 13th 2011.

T. Kindberg and S. Hawke. The 'tag' URI Scheme. RFC 4151 (Informational), oct 2005. `http://www.ietf.org/rfc/rfc4151.txt` retrieved on June 16th 2011.

Graham Klyne and Jeremy Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, 2004. `http://www.w3.org/TR/rdf-concepts/` retrieved on Feb 4th 2011.

Wallace Koehler. Web page change and persistence—a four-year longitudinal study. *Journal American Society for Information Science and Technology*, 53:162–171, January 2002. ISSN 1532-2882.

S. Lawrence, D.M. Pennock, G.W. Flake, R. Krovetz, F.M. Coetzee, E. Glover, F.A. Nielsen, A. Kruger, and C.L. Giles. Persistence of web references in scientific research. *Computer*, 34(2):26–31, 2001.

R. Moats. Urn syntax. RFC 2141 (Proposed Standard), May 1997. `http://www.ietf.org/rfc/rfc2141.txt` retrieved on Feb 4th 2011.

Luc Moreau, Juliana Freire, Joe Futrelle, Robert McGrath, Jim Myers, and Patrick Paulson. The Open Provenance Model. Technical Report 264979, Department of Electronics and Computer Science, University of Southampton, December 2007.

Luc Moreau and Nicholas Gray. A Community of Agents Maintaining Links in the World Wide Web (Preliminary Report). In *PAAM 1998: Proceedings of the 3rd*

International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, pages 221–235, London, UK, March 1998.

Atsuyuki Morishima, Akiyoshi Nakamizo, Toshinari Iida, Shigeo Sugimoto, and Hiroyuki Kitagawa. PageChaser: A Tool for the Automatic Correction of Broken Web Links. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, pages 1486–1488, Shanghai, Chine, 2008. IEEE Computer Society.

T. H. Nelson. Complex information processing: a file structure for the complex, the changing and the indeterminate. In *Proceedings of the 1965 20th national conference*, ACM '65, pages 84–100, Cleveland, OH, USA, 1965. ACM.

T.H. Nelson. *Literary Machines*. Mindful Press, 1980.

J. Nielsen and U. Lyngbæk. Two field studies of hypermedia usability. In R. McAleese and C. Green, editors, *Hypertext: state of the art*, pages 64–72, New York, NY, USA, 1989. Ablex.

Jakob Nielsen. The art of navigating through hypertext. *Communications of the ACM*, 33:296–310, March 1990.

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

A. Pam and A. Vermeer. A Comparison of WWW and Hyper-G. *Journal of Universal Computer Science*, 1(11):744–750, 1995.

Norman Paskin. Digital Object Identifier (DOI®) System. pages 1586–1592, 2010.

A. Pearl. Sun's Link Service: a protocol for open linking. In *Proceedings of the second annual ACM conference on Hypertext*, HYPERTEXT '89, Pittsburgh, Pennsylvania, United States, 1989. ISBN 0-89791-339-6.

Thomas A. Phelps and Robert Wilensky. Robust Hyperlinks: Cheap, Everywhere, Now. In P. King and E.V Munson, editors, *Digital Documents: Systems and Principles*, volume 2023 of *Lecture Notes in Computer Science*, pages 514–549. Springer, 2004.

James E. Pitkow. Summary of www characterizations. *Computer Networks and ISDN Systems*, 30(1–7):551–558, 1998. WWW 1998: Proceedings of the 7th International Conference on the World Wide Web.

Niko P. Popitsch and Bernhard Haslhofer. DSNotify: handling broken links in the web of data. In M. Rappa, P. Jones, J. Freire, and S. Chakrabartia, editors, *WWW 2010: Proceedings of the 19th International Conference on the World Wide Web*, pages 761–770, Raleigh, North Carolina, USA, 2010. ACM.

S.M. Powsner and NK Roderer. Navigating the internet. *Bulletin of the Medical Library Association*, 82(4):419, 1994.

Dave Raggett. HTML+ (Hypertext markup format). Internet Draft (Expired), November 1993. https://datatracker.ietf.org/doc/draft-raggett-www-html/ abstract retrieved on July 15th 2011, full text no longer retrievable.

C. Schwartz. Web search engines. *Journal of the American Society for Information Science*, 49(11):973–982, 1998.

Diomidis Spinellis. The decay and failures of web references. *Communications of the ACM*, 46(1):71–77, 2003.

S. Sun, L. Lannom, and B. Boesch. Handle System Overview. RFC 3650 (Informational), nov 2003a. http://www.ietf.org/rfc/rfc3650.txt retrieved on July 19th 2011.

S. Sun, S. Reilly, and L. Lannom. Handle System Namespace and Service Definition. RFC 3651 (Informational), nov 2003b. http://www.ietf.org/rfc/rfc3651.txt retrieved on July 19th 2011.

S. Sun, S. Reilly, L. Lannom, and J. Petrone. Handle System Protocol (ver 2.1) Specification. RFC 3652 (Informational), nov 2003c. http://www.ietf.org/rfc/rfc3652.txt retrieved on July 19th 2011.

H. Van de Sompel, M.L. Nelson, R. Sanderson, L.L. Balakireva, S. Ainsworth, and H. Shankar. Memento: Time travel for the Web. *Computing Research Repository*, abs/0911.1112, 2009.

H. Van de Sompel, R. Sanderson, M.L. Nelson, L.L. Balakireva, H. Shankar, and S. Ainsworth. An http-based versioning mechanism for linked data. In *LDOW 2010: Proceedings of the 3rd International Workshop on Linked Data on the Web at WWW 2010*, Raleigh, North Carolina, USA, April 2010.

L. Veiga and P. Ferreira. Turning the web into an effective knowledge repository. *ICEIS 2004: Proceedings of the 6th International Conference on Enterprise Information Systems*, 14(17), 2004.

Luís Veiga and Paulo Ferreira. RepWeb: replicated Web with referential integrity. In *SAC 2003: Proceedings of the 2003 ACM Symposium on Applied Computing*, pages 1206–1211, Melbourne, Florida, 2003. ACM.

Robert Vesse, Wendy Hall, and Les Carr. All About That - A URI Profiling Tool for monitoring and preserving Linked Data. In A Bernstein, D.R Karger, T Heath, L Feigenbaum, D Maynard, E Motta, and K Thirunarayan, editors, *ISWC 2009: Proceedings of the the 8th International Semantic Web Conference*, volume 5823 of *Lecture Notes in Computer Science*, Washington DC, USA, August 2009.

Robert Vesse, Wendy Hall, and Les Carr. Preserving Linked Data on the Semantic Web by the application of Link Integrity techniques from Hypermedia. In *LDOW 2010: Proceedings of the 3rd International Workshop on Linked Data on the Web at WWW 2010*, Raleigh, North Carolina, USA, April 2010.

Laurie Wedeles. Professor Nelson Talk Analyzes P.R.I.D.E, February 3rd 1965. `http://faculty.vassar.edu/mijoyce/MiscNews_Feb65.html` retrieved on July 11th 2011.

William E. Winkler. The state of record linkage and current research problems. Technical Report 99-04, Statistical Research Division, U.S. Census Bureau, 1999.

Appendix A

Queries for Query Use Case Evaluation

Each section of this Appendix lists the queries used for the query use case evaluation presented in Section 7.3. It also includes brief notes on the properties chosen for the queries and the SPARQL Protocol and RDF Query Language (SPARQL) endpoints of the datasets used to evaluate the queries in order to generate the benchmarks.

A.1 Queries for BBC Programmes Dataset

The British Broadcasting Corporation (BBC) Programmes dataset does not use RDF Schema (RDFS) properties to state facts like labels and descriptions preferring to use `dc:title` for labels and a property in their own vocabulary for descriptions. In the relationship query for this dataset we ask for all the episodes associated with a program.

Since the BBC themselves do not provide a SPARQL endpoint for this dataset we instead used the mirror of the data and its endpoint on the Talis platform at http://api.talis.com/stores/bbc-backstage/services/sparql

```
PREfIX dc: <http://purl.org/dc/elements/1.1/>

SELECT DISTINCT ?label
WHERE
{
        ?uri dc:title ?label .
}
```

Listing A.1: Label Query for BBC Programmes Dataset

```
PREfIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema\#>
PREFIX owl: <http://www.w3.org/2002/07/owl\#>

SELECT DISTINCT ?label
WHERE
{
        { ?uri dc:title ?label . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias dc:title ?label . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias dc:title ?label . }
}
```

Listing A.2: Advanced Label Query for BBC Programmes Dataset

```
PREFIX po: <http://purl.org/ontology/po/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?description
WHERE
{
        ?uri po:medium_synopsis ?description .
}
```

Listing A.3: Description Query for BBC Programmes Dataset

```
PREFIX po: <http://purl.org/ontology/po/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?description
WHERE
{
        { ?uri po:medium_synopsis ?description . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias po:medium_synopsis ?description . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias po:medium_synopsis ?description . }
}
```

Listing A.4: Advanced Description Query for BBC Programmes Dataset

```
PREFIX po: <http://purl.org/ontology/po/>

SELECT DISTINCT ?episode
WHERE
{
        ?uri po:episode ?episode .
}
```

Listing A.5: Relationship Query for BBC Programmes Dataset

```
PREFIX po: <http://purl.org/ontology/po/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?episode
WHERE
{
        { ?uri po:episode ?episode . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias po:episode ?episode . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias po:episode ?episode . }
}
```

Listing A.6: Advanced Relationship Query for BBC Programmes Dataset

A.2 Queries for DBPedia Countries Dataset

The DBPedia[1] dataset uses the RDFS properties to express labels and descriptions in addition to their own custom properties. We use the RDFS properties in the query since it is more likely that secondary data sources would also use these. For the relationship query we look for the wiki page links denoted by the property `http://dbpedia.org/ontology/wikiPageExternalLink`, this is a potentially hard query as we expect that secondary data sources may not include this data.

These queries were run against the DBPedia SPARQL endpoint at `http://dbpedia.org/sparql`

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?label
```

[1]`http://dbpedia.org`

```
WHERE
{
        ?uri rdfs:label ?label .
}
```

Listing A.7: Label Query for DBPedia Dataset

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?label
WHERE
{
        { ?uri rdfs:label ?label . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias rdfs:label ?label . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias rdfs:label ?label . }
}
```

Listing A.8: Advanced Label Query for DBPedia Dataset

```
SELECT DISTINCT ?description
WHERE
{
        ?uri rdfs:comment ?description .
}
```

Listing A.9: Description Query for DBPedia Dataset

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?description
WHERE
{
        { ?uri rdfs:comment ?description . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias rdfs:comment ?description . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias rdfs:comment ?description . }
}
```

Listing A.10: Advanced Description Query for DBPedia Dataset

```
PREFIX dbp-owl: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?link
```

```
WHERE
{
        ?uri dbp-owl:wikiPageExternalLink ?link .
}
```

Listing A.11: Relationship Query for DBPedia Dataset

```
PREFIX dbp-owl: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?link
WHERE
{
        { ?uri dbp-owl:wikiPageExternalLink ?link . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias dbp-owl:wikiPageExternalLink ?link . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias dbp-owl:wikiPageExternalLink ?link . }
}
```

Listing A.12: Advanced Relationship Query for DBPedia Dataset

A.2.1 Vocabulary Mapped Query

This is the additional variation of the label query we evaluated to see whether taking into the account that one of our major secondary data sources for this dataset uses an alternative property for `rdfs:label` would make a difference to our results.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX fb: <http://rdf.freebase.com/ns/>

SELECT DISTINCT ?label
WHERE
{
        { ?uri rdfs:label ?label . }
        UNION
        { ?uri fb:type.object.name ?label . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias rdfs:label ?label . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias fb:type.object.name ?label . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias rdfs:label ?label . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias fb:type.object.name ?label . }
}
```

Listing A.13: Vocabulary Mapped Advanced Label Query for DBPedia Dataset

A.3 Queries for ECS People Dataset

The ECS People[2] dataset uses the custom `akt:full-name` property for labels but does not include longer textual description of the people, thus the type query is used in place of the description query. For the relationship query we ask for the publications that a person has been an author on which are related by the `akt:has-author` property.

The official SPARQL endpoint of this dataset can be found at `http://southampton.rkbexplorer.com/sparql/` but unfortunately the triplestore software on which this endpoint runs does not support queries using `UNION` clauses properly. Fortunately the site does provide full downloads of the data files that make up the dataset at `http://southampton.rkbexplorer.com/models/` which we downloaded and loaded into a local SPARQL endpoint using the open source Fuseki[3] software. Having created this local endpoint we then generated the benchmark using our local copy of the data since Fuseki supports the full range of SPARQL features correctly.

```
PREfIX akt: <http://www.aktors.org/ontology/portal#>

SELECT DISTINCT ?label
WHERE
{
        ?uri akt:full-name ?label .
}
```

Listing A.14: Label Query for ECS People Dataset

```
PREfIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?label
WHERE
{
        { ?uri akt:full-name ?label . }
```

[2] `http://southampton.rkbexplorer.com/`
[3] `http://openjena.org/wiki/Fuseki`

```
        UNION
        { ?uri owl:sameAs ?alias . ?alias akt:full-name ?label . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias akt:full-name ?label . }
}
```

Listing A.15: Advanced Label Query for ECS People Dataset

```
SELECT DISTINCT ?type
WHERE
{
        ?uri a ?type .
}
```

Listing A.16: Type Query for ECS People Dataset

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?type
WHERE
{
        { ?uri a ?type . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias a ?type . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias a ?type . }
}
```

Listing A.17: Advanced Type Query for ECS People Dataset

```
PREfIX akt: <http://www.aktors.org/ontology/portal#>

SELECT DISTINCT ?publication
WHERE
{
        ?publication akt:has-author ?uri .
}
```

Listing A.18: Relationship Query for ECS People Dataset

```
PREFIX akt: <http://www.aktors.org/ontology/portal#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?publication
WHERE
{
        { ?publication akt:has-author ?uri . }
        UNION
        { ?uri owl:sameAs ?alias . ?publication akt:has-author ?uri . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?publication akt:has-author ?uri . }
}
```

Listing A.19: Advanced Relationship Query for ECS People Dataset

A.4 Queries for Drugbank Dataset

The Drugbank[4] dataset uses the standard `rdfs:label` property for labels but does not include longer textual description of the drugs thus the type query is used in place of the description query. For the relationship query we ask for the targets of the drugs using the `db:target` property.

These queries were run against the Drugbank SPARQL endpoint at http://www4.wiwiss.fu-berlin.de/drugbank/sparql

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?label
WHERE
{
        ?uri rdfs:label ?label .
}
```

Listing A.20: Label Query for Drugbank Dataset

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?label
WHERE
{
        { ?uri rdfs:label ?label . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias rdfs:label ?label . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias rdfs:label ?label . }
}
```

Listing A.21: Advanced Label Query for Drugbank Dataset

[4] http://www4.wiwiss.fu-berlin.de/drugbank/

```
SELECT DISTINCT ?type
WHERE
{
        ?uri a ?type .
}
```

Listing A.22: Type Query for Drugbank Dataset

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?type
WHERE
{
        { ?uri a ?type . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias a ?type . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias a ?type . }
}
```

Listing A.23: Advanced Type Query for Drugbank Dataset

```
PREFIX db: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>

SELECT DISTINCT ?target
WHERE
{
        ?uri db:target ?target .
}
```

Listing A.24: Relationship Query for Drugbank Dataset

```
PREFIX db: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT ?target
WHERE
{
        { ?uri db:target ?target . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias db:target ?target . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias db:target ?target . }
}
```

Listing A.25: Advanced Relationship Query for Drugbank Dataset

A.4.1 Literal Mapped Query

These are the additional variations of the label queries that we evaluated to see whether taking into account the fact that the Drugbank dataset does not include language specifiers for literals while secondary data sources do would make a difference in our query use case results.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT (STR(?baseLabel) AS ?label)
WHERE
{
        ?uri rdfs:label ?baseLabel .
}
```

Listing A.26: Label Query for Drugbank Dataset

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT DISTINCT (STR(?baseLabel) AS ?label)
WHERE
{
        { ?uri rdfs:label ?baseLabel . }
        UNION
        { ?uri owl:sameAs ?alias . ?alias rdfs:label ?baseLabel . }
        UNION
        { ?uri rdfs:seeAlso ?alias . ?alias rdfs:label ?baseLabel . }
}
```

Listing A.27: Advanced Label Query for Drugbank Dataset