UNIVERSITY OF SOUTHAMPTON

Open Semantic Hyperwikis

by

Philip R. Boulain

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

in the
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

28th February 2011

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

Open Semantic Hyperwikis

by Philip R. Boulain

Wikis are lightweight, community-editable, web-based hypertext systems, which can be described as a website that anybody can edit. From this collaborative base has grown significant efforts at large-scale knowledge management such as Wikipedia. Recently, 'semantic' wiki systems have been developed with typed links, such that the structure of nodes and links is analogous to an RDF graph of resources and arcs: a machine-processable representation of the relations between articles which can form part of the web of linked data. Despite this, the hypermedia side of wiki systems has so far largely been constrained to the web model of simple embedded, unidirectional links.

This research considers the hypertext origins of wiki systems, asks, and answers how the technologies developed during decades of hypertext research may be applied to better manage their document, and thus knowledge, structure. We present experimental evidence supporting the hypothesis that additional hypermedia features would be useful to wiki editors on both macro- and micro-scales. Quantitative analysis of editing logs from a large-scale wiki shows that hyperstructure changes form a substantial proportion of editing effort. Conversely, qualitative user studies show that individual user editing can be better supported by classical but since overlooked hypertext features such as first-class links and transclusion.

We then specify an extensive model for a 'open semantic hyperwiki' system which draws from these fields, based around first-class links with support for transclusion and advanced functional link types, with defined semantics for the role of versioning and parametric nodes in the linked data world, while mindful to preserve the core simplicity that allows non-expert users to contribute. This is followed by a practical approach to its implementation in terms of an existing experimental modular wiki foundation, and the actual prototype implementation, which has been made available as open source software. Finally, we work through applying the system to a set of real-world use cases which are currently employing classic, non-semantic wiki software, and evaluate the implementation in comparison to a conventional semantic wiki in a user study.

Contents

List of Figures

List of Tables

Declaration of Authorship

I, Philip Richard Boulain,
declare that the thesis entitled
Open Semantic Hyperwikis
and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- this work was done wholly or mainly while in candidature for a research degree at this University;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- where I have consulted the published work of others, this is always clearly attributed;
- where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
- none of this work has been published before submission

Signed:
Date:

Acknowledgements

Thanks to my parents for tolerating their son spending yet more time in education rather than getting a real job; to Nick Gibbins and Nigel Shadbolt for attempting to focus my efforts on generating something academically palatable; to Harry Mason for a fifteen-times speed-up in the Levenshtein string comparison by suggesting top-and-tailing the identical parts of the strings; to the LaTeX, XY-pic, and GNUPLOT maintainers for the software used to create this document and its diagrams, and likewise the whole Apache/Perl stack for the prototype; to `#ecs` for being full of likewise unsympathetic, burnt-out postgraduates and alumni, and Mair Allen-Williams in particular for encouraging me to get a Brompton and cycle more; and to Boston, Whitesnake, 38 Special, et. al. for producing the large quantities of classic rock required to survive the all-nighters without going insane and deciding to rewrite the entire thesis in Reverse Polish Notation.

Chapter 1

Introduction

In this document, we look at the overlap between two research areas: Open Hypermedia, and the Semantic Web, with emphasis specifically on Semantic Wikis.

A hypermedia, or a hypertext, is a document which extends beyond the limits of paper. There are many facets to hypertext research, but two common extensions beyond non-electronic documents are greater cross-reference, by means of links which can be used to navigate between or within documents, and greater re-use, as content may be freely duplicated in multiple contexts. Open hypermedia extends this by considering links to be standalone entities, which can be applied to a set of potentially immutable documents in arbitrary groups; these links may also span heterogeneous systems.

The World Wide Web is a distributed set of interlinked documents. The Semantic Web, an extension of this, intends to be a distributed set of interlinked *data*. It allows data sources to be discovered and combined from disparate sources in the same manner as one may discover, browse, and mentally cross-reference web pages.

A “wiki”, or “WikiWikiWeb” in full, is a lightweight, collaborative hypertext system built on top of the existing World Wide Web. It is lightweight in that it offers only the most basic of web-style linking primitives, and often an ad-hoc markup system which maps to a subset of HTML. It is collaborative, as it allows modification from multiple users—in many cases, any web user—usually with basic conflict-detection semantics, and ideally with a history system to allow abuse to be undone as easily as it is made.

1.1 Problem

Wikis have been hugely successful systems, with prominent examples such as Wikipedia[1] attracting millions of users writing over a million articles[2]. They also expand beyond en-

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

[2] `http://en.wikipedia.org/wiki/Special:Statistics`

cyclopædic uses: wikis have seen use in areas such as enterprise knowledge management, software development, and community building.

However, we hypothesise that a significant proportion of the substantial effort expended by these users is on tasks which could be automated. Reducing the maintenance burden of working with these ad-hoc knowledge bases allows for more effort to be spent on improving the quality of, or working with, the content.

1.2 Solution

There are already efforts to add Semantic Web features to wikis ('semantic wikis'), of particular note Semantic MediaWiki [1], which solve some of these problems, but we believe that the overlap with hypermedia allows for greater improvement still. The objective of this research is to design and develop a prototype modified wiki to demonstrate how open hypermedia systems can provide this further improvement. To accomplish this, we surveyed the past developments of hypermedia research, tested the hypothesis that there is actually link-editing inefficiency to improve upon, identified the priorities in implementing an improved system, developed a model which addresses these improvements, planned and implemented a prototype system, and evaluated it compared to a conventional wiki.

1.3 Structure

In the next chapter, the semantic web is explained in more depth, and notable hypertext and semantic wiki systems are covered and compared in more detail. Chapter 3 looks at the correlations discovered between the areas of Semantic Web, Open Hypertext, and wikis. It begins with a description of the Semantic MediaWiki system in terms of the Dexter hypertext model, then extends and inverts this to express common hypertext features in semantic wiki terms. Chapter 4 expands upon the concept of *Open Semantic Hyperwiki*: a semantic wiki reinforced with a richer hypertext structure. It puts forward arguments for specific improvements which should prove beneficial to users of such a system. To demonstrate the current limitations of wikis, chapter 5 presents an experiment performed on the large-scale Wikipedia system, which measures the proportion of editing effort expended on link maintenance tasks, compared to providing encyclopædic content. We follow this with a small-scale experiment in chapter 6 to identify the tasks individual editors set themselves, and how they act to achieve them. This informs our parallel work on a model for open semantic hyperwikis in chapter 7, where we formalise the hypothetical system of past chapters. Chapter 8 then addresses the task of implementing the model as a prototype system, building on a past system for large components, and working through the detail of more intricate linking problems. Chapter 9 details our creation of this prototype system, along with the complexities introduced to a modular system of orthogonal components by the interlinked nature of some hypertext features. Chapter 10 then shows how our model and system could be applied to concrete, non-encyclopædic examples of wiki use, and how its functionality would be beneficial. To then evaluate the prototype system in the hands of an untrained novice user, chapter 11 revisits the small-scale experiment, this time using our technology in place of Mediawiki. We capture user reaction to the enriched hypertext experience and how they decide and attempt to apply the new capabilities to familiar real-world tasks. We finish with chapter 12, in which we summarise our findings, and propose directions in which this research can be continued, such as more advanced versioning support, non-text content types, and forming a distributed, seamless hyperdocument between multiple wikis.

Chapter 2

Literature review

2.1 Introduction

This chapter explains the Semantic Web in more depth, covers how it extends from the World Wide Web, and how it relates to research fields such as Intelligent Agents. It covers technologies used, both of the existing Web and new, and some of the issues around them. We then move on to notable hypertext systems and models and, in particular, defining features which apply to, or could be applied to, semantic wikis. This is followed by coverage of current semantic wikis, comparing their approaches to the idea of adding semantics to wikis, and their different feature-sets. Finally, we consider Halasz's seminal "seven issues" in the overarching context of the modern hypertext and semantic wiki environment.

2.2 Semantic Web

If the World Wide Web can be seen as a distributed network of human-readable documents, then the Semantic Web can be thought of as a distributed network of machine-processable data. It follows the same open principles: that anyone can publish using open standards, with no centralised control. The "information bus" [2] between heterogeneous server and client implementations remains, such that any consumer should be able to use and combine data from any set of providers, but instead of a generic hypertext markup language, the Semantic Web uses a generic data model: RDF [3].

The Semantic Web aims to add enough structured metadata to web content to elevate it from being just human-understandable, to also machine-understandable. As well as the existing natural language documents, the web would contain information about the key classes, instances, and the relations between them, which computer programmes could then use to perform more intelligent queries, and even derive new conclusions.

Search engines could ensure that they found pages with the correct concept, ignoring homonyms but possibly finding synonyms. Intelligent Agents—autonomous, goal-driven programmes, to which a user delegates tasks at a higher level than individual actions—could potentially use this information to fulfil more complicated requests. The well-known Scientific American article by Berners-Lee et. al. [4] gives an example of scheduling therapy appointments and making travel plans based on availability and location information offered in a consistent manner—or at least a manner which maps to a consistent representation—by the therapists' websites.

2.2.1 Web technologies

The unique identifier of the web, the URI[1], is extended to refer to any kind of resource, such as a class, rather than just documents. RDF serialisations (e.g. using XML) are still sent over the same transport layer, HTTP.

2.2.2 Data model

One can describe semantic web resources using nothing more than RDF statements: triples that state that a *resource* has a *property* with some *value*, where the first two, and possibly the value, are identified as URIs. This use of URIs allows for uniqueness and disambiguation between meanings of English language terms, such as 'title' ('Mr.'; 'of a document'; 'title deed'). These can each be given distinct identifiers, such as '`http://purl.org/dc/elements/1.1/title`' for the title of a document. To state that the EPrints document record with identity '`oai:eprints.ecs.soton.ac.uk:12900`' (the *resource*) has a title (the *property*) of "Weerkat: An extensible semantic Wiki" (the *value*: a literal, in this case), one could use the triple '`<oai:eprints.ecs.soton.ac.uk:12900> <http://purl.org/dc/elements/1.1/title> "Weerkat: An extensible semantic Wiki"`'.

On an uncontrolled, decentralised—and thus, ultimately, scalable—system such as the Semantic Web, it is not possible to enforce consistent terminology: one business may describe part of their mailing address using a property of '`http://purl.org/snailmail/-zipcode`', and another '`http://www.postoffice.co.uk/rdf/postcode`'. While both mean the same thing, a programme has no reliable way of determining this by itself. Ideally, the two businesses, and many others, above would use a common *ontology* for mailing addresses, effectively agreeing on that particular model of the real-world concept. An ontology, in the context of knowledge management, is "an explicit specification of a conceptualisation" [5]: it defines precisely how a system is modelled, such that the model may be used by and combined with with other data sources. RDF Schema [6] (RDFS) is one vocabulary which offers a simple type system for the RDF model. Such

[1]Uniform Resource Identifier

schema can declare taxonomies of classes and properties, and simple type constraints on the range and domain of properties. RDFS also provides properties for annotating these resources for human presentation, and to document the intended usage of the schema.

OWL [7] is another common ontology language, which builds on RDFS and provides semantics based on description logic. It can describe classes in terms of restrictions such as cardinality, range, and value constraints on their properties, and the relations between these classes. An OWL ontology can define class and property equivalence relations, which can specify that the ‘zipcode’ and ‘postcode’ above are actually the same thing. Inference can be used for both consistency checking of data, ensuring that statements do not contradict (e.g. ‘YellowThings are only coloured yellow’, ‘apples are green’, ‘yellow and green are distinct’, ‘apples are YellowThings’), and for classifying instances (e.g. ‘rubber ducks are yellow’, therefore ‘rubber ducks are YellowThings’).

2.2.3 Triple stores

To manipulate large semantic web datasets, the RDF statements are usually loaded into specialised databases, known as triple stores. These systems are optimised for the efficient storage and query of large numbers of interrelated triples, and have evolved from earlier layers on top of conventional relational databases, such as 3store [8] or Redland[2], to outright bespoke storage models, such as 4store [9].

2.2.4 SPARQL

Much as a relational database is queried using SQL, one must be able to make queries of a triple store. The contemporary, storage-independent language for expressing such queries is the SPARQL Query Language [10].

SPARQL queries have a syntax reminiscent of SQL, and can contain a `WHERE` clause for pattern-matching with wildcard variables. Within this, `FILTER` constraints can be added for more complicated matching, such as numerical inequalities or matching strings against regular expressions.

The results of a SPARQL `SELECT` query are a set of *bindings*, not a list of triples. Figure 2.1 shows a trivial example. The query 2.1(c) returns a set of bindings 2.1(b) for each variable when executed over the dataset 2.1(a). In this case, we receive two possible bindings, where our single variable `c` has been bound to `red` in one possible satisfaction of our query, and `yellow` in the other.

A HTTP service which acts as a SPARQL interface to some knowledge base is generally referred to as a *SPARQL endpoint*. Theoretically, software other than triple stores

[2]`http://librdf.org/`

Subject	Predicate	Object
ball	colour	red
ball	colour	yellow
ball	pattern	striped

(a) RDF statements

Result N^{o}	c
1	red
2	yellow

(b) Resultant bindings

```
SELECT ?c WHERE { ball colour ?c . }
```

(c) SPARQL query

FIGURE 2.1: An small set of triples and the bindings returned by an example query upon them. Namespacing is not show for conciseness.

may provide such interfaces, although currently semantic wiki-related systems such as Semantic MediaWiki and DBPedia implement SPARQL endpoints by automating RDF export into a sideband triplestore which then services such queries.

It is worth noting that, unlike SQL, current SPARQL only deals with read operations—adding, modifying, or removing triples from a store requires another, store-specific, mechanism.

2.2.5 Linked data

In 2006, Tim Berners-Lee published a design issues paper on 'linked data' [11], an attempt to refocus the Semantic Web on the ability to navigate between sets of related machine processable data, rather than simply 'dumping' it on the web.

Of his points, crucial were the use of HTTP protocol URIs for naming resources, so that there is a simple and obvious mechanism to retrieve something pertaining to that resource; and to link to other URIs.

2.2.5.1 303 convention

Conventions have been proposed [12] to use HTTP's content type negotiation and redirection capabilities to distinguish between and relate URIs for the *identity* of a resource (e.g. the person "Wendy Hall"), a *representation* of a resource (e.g. an image of Wendy Hall), and *metadata* about a resource (e.g. a RDF document describing Wendy Hall). In particular, the '303 convention' uses HTTP status code 303, "See Other", to redirect a request for an identity URI to either an appropriate representation, or to the metadata document, depending on whether the client requested a result of RDF type. This redirection, rather than merely returning the requested type directly, allows the client to become aware of the URI for this particular representation. To get from an HTML representation to a metadata document, the "link" header element can be used; this document then defines the resource in terms of its identifier.

An alternative approach, suitable for small, largely-static data sets, modifies this to use the fragment identifier in identity URIs. For example, `http://www.w3.org/1999/02/22-rdf-syntax-ns#Resource` identifies the class of resources in the RDF model. HTTP requires that clients remove the fragment identifier before retrieving URIs, resulting in an URI of `http://www.w3.org/1999/02/22-rdf-syntax-ns`. This may return the RDF metadata directly (thus serving as an identifier for it), or may go via a stage of 303 indirection to allow for alternate representations, as above.

These conventions are a fairly pragmatic way to both keep distinct, yet associate, the facets of a resource, but they are not without shortcomings. An obvious wrinkle is the increased number of HTTP requests needed to resolve the identifier URI down to a suitable representation. Another is the inability to provide the relation between representation and metadata in formats which do not support embedding such header data (e.g., plain text), or to access this relation without downloading the whole document. Attempts have been made to preserve the HTTP Link header for this purpose, which could express the relation in the protocol layer, and in response to HEAD requests (which do not include the content body), but have expired without acceptance. [13, 14]

A more serious problem is that of overloading the RDF content type. The 303 approach redefines part of content negotiation such that a request for data of RDF type is understood as a request for *metadata*: rather than returning the content in that format, the server returns a description *about* the content. As a result, there is no unambiguous way to request the actual content in RDF format, even though this may be sensible for, for example, a press release stating that a new product has been made available. Likewise, one cannot request metametadata for the metadata: if the 303 convention has provided a client with the URI of metadata for Wendy Hall, the client cannot then ask for metadata about this data, such as who provided it, or when it was last updated. Requests for RDF to the metadata address will be interpreted as a post-redirection requests for the metadata itself. URIQA [15] disambiguates this by adding explicit 'metadata' operations to HTTP, but these are not standard. An explicit relation to the metametadata, e.g. via the HTTP Link header, would also allow the client to discover the correct URI to request.

2.2.5.2 RDF in attributes

One way to embed metadata into regular web documents is RDFa [16]. It allows type information to be added to links within a page via additional XHTML attributes, and for literal content to be used as RDF literals. The implicit subject of such statements is the page itself, as with semantic wikis, but the tree structure of the document can be used to scope branches to describe other URIs.

2.3 Open hypermedia

Hypermedia is a long-standing field of research into the ways in which documents can expand beyond the limitations of paper, generally in terms of greater cross-referencing and composition (reuse) capability.

2.3.1 Documentation

Otlet's work on documentation [17], near the beginning of the 20th century, is some of the earliest evidence of structured organisation of documents, with indices into, and references between, them. While his technology was based on index cards and sheets of paper, the system he envisaged would effectively be a universal, dynamic Encyclopedia, "formed by linking together materials and elements scattered in all relevant publications" [18]. Knowledge resources such as books were broken down into the elements of information they presented, as one may do with the Semantic Web, not the presentational divisions, such as chapters and paragraphs. Documents were indexed by a 'Universal Decimal Classification', which formed a taxonomy of subjects, and was intended to allow indexing upon facets of documents, such as their date. Otlet even envisaged that documents would be accessed remotely via a hypothetical, globally-distributed Universal Network of Information and Documentation, using a workstation with screens and speakers. The system was let down by sheer lack of scalability arising from the overheads of dealing with physical searching and copying, but the vision was surprisingly prescient, as more recent systems still work to fulfil the same goals.

2.3.2 The Memex

Bush's *As We May Think* [19] introduces the standard hypothetical early hypertext machine, the 'memex', and defines the "essential feature" of it as "the process of tying two items together". This *linking* between documents is the common feature of hypertext systems, upon which other improvements are built. Bush's 'trails' formed arbitrary linked lists over the set of available documents, but their sharable nature, and the model used by later hypertext systems, effectively forms a graph of documents, which greatly enables exploration of subjects. As well as the original binary (two endpoint) links, hypertext systems have been developed with features including n-ary links (multiple documents link to multiple other documents), typed links (links which indicate something about *why* or *how* documents are related), and composite documents, which are formed by combining a set of other, linked, documents.

Open Hypermedia is the sub-field of hypermedia which focuses on how hypermedia can interoperate, both with other hypermedia systems and users, and with non-hypermedia resources. Two of the key features of open hypermedia systems are first-class links—links which have identity, and are treated at objects at a similar level to documents—and the separation of storage of links from document content into linkbases. By storing the links in external repositories, rather than requiring them to be kept with documents, they may refer to resources which are not hypermedia-aware, or cannot be modified. It also becomes possible to share linkbases, as with Bush's trails, and to change which linkbases are currently being applied. As well as being one possible way to implement adaptive hypermedia—the modification of which links are shown based on user context—this allows users to have their own linkbases, group-shared linkbases, and any other organisation which proves useful. This section covers some of the more notable systems and models on the subject of open hypermedia.

2.3.3 Xanadu

One of the earliest projects attempting to implement globally-distributed, globally-edited hypertext was Xanadu [20]. While implementation was infamously troubled, Nelson's compelling vision was that of a 'document pool' of content, contributed by users across the system. The users could form arbitrary links between these documents, and create compound documents via 'transclusion', an inclusion-by-reference mechanism with licensing control ('transcopyright'). The intent is to capture arbitrarily structured information in such a way that comparison of adjacent versions and commentary is easy to achieve, which makes this a very general form of hypertext. To support this, an interesting addressing system known as 'tumblers' is used, which supports infinite subdivision of addressing across multiple dimensions: for example, it is possible to subdivide the concept of 'user' by organisation, department, down to individual, with as many levels as is necessary. The ability to always subdivide a tumbler address allows for arbitrary insertions to be made without the need for renumbering, which assists in maintaining the permanence of link addresses.

Xanadu is interesting in that it was designed explicitly to allow anyone to effectively edit anything by re-using it with new identity, even if only the original author could then retire (and effectively replace) the older version. The ability to link *from* resources owned by others, and transclusion, encourage communal commentary and 'remixing' in ways familiar to the current 'Web 2.0' movement: in particular, the paradigm of commenting on a web resource by making a blog post linking to it with commentary attached, and the Creative Commons movement[3]. However, neither of these are as powerful as Xanadu: the former comments are only associated with the resource in one direction, such that one cannot find the comments from the resource, and must exist on a distinct page. They cannot be interweaved with the resource as one may add marginalia to a printed document without explicitly copying the resource, which may be prohibited, or

[3]`http://creativecommons.org/`

otherwise undesirable. While Creative Commons metadata can clarify these permissions, it does not provide the same structure for fair licensing fees—micropayments—as was envisaged for transcopyright. What transclusive mechanisms exist in the web today (such as embeddable video players and inline frames) have restrictions on granularity, only able to operate on entire documents rather than create composites from excerpts.

2.3.4 HyTime

HyTime [21] was an SGML-based format for the representation of hypermedia, evolved from an effort to develop a standard music description language, with an emphasis on presentation. It aimed to provide an abstraction of the common, linking parts of arbitrary data formats, such that this information could be shared between applications. HyTime supports five kinds of hyperlink: *independent*, *property*, *contextual*, *aggregate*, and *span*. Independent links are 'ordinary' n-ary links, and contextual links are what are more generally known as 'embedded' links—one end is the location of the link itself. Aggregate links handle co-reference resolution: if two locations are linked via such a link, they are logically considered to be the same location. This foreshadows the need for equivalence relations in the Semantic Web (e.g. `owl:sameAs`): when combining documents or data from disparate sources, multiple identifiers may be used for the same resource. Span links are an implementation detail which allows SGML elements to be ignored in documents conformant to a DTD other than that of HyTime. Property links are similar to triples in RDF: they associate one link end with some other value or link end, for a given attribute. Note that, unlike RDF, the attribute is not given as another first-class object of the system: it does not have equal standing to other resources, and cannot itself be a link endpoint (thus you cannot create property links which annotate attribute types). The intent of property links are not to be a general annotation mechanism, but to allow description of entities which cannot be modified to describe itself 'in the normal way' (as part of the element itself).

HyTime also has a mechanism of 'batons' and 'wands' to control presentation of objects (multimedia content, occuring within a chronological event, and with a spatial extent) within a co-ordinate space. Batons are schedules of projectors, which map object events from their source co-ordinates into display co-ordinates: for example, a two-dimensional map dataset may be scaled to an arbitrary ratio by a suitable baton, and the object is then displayed within this new context. HyTime supports basic types of batons, such a linear scaling, but allows for application-specific batons. Wands are schedules of modifiers, which can perform arbitrary modifications upon the objects themselves. As objects are treated as generic, opaque media items, wands are likewise opaque and not standardized. Both projectors and modifiers are scoped not only by their schedule, but by the area in the co-ordinate space to which they apply.

2.3.5 Microcosm

Microcosm [22, 23] was an open hypertext system. One of its key features was that links were kept separately from content, thus allowing links to be overlaid onto read-only documents. This also allows links to be 'generic', i.e. not applicable only to documents they were originally conceived for: such links have endpoints defined as tokens which may appear in documents, for example, individual words. Early wiki systems featured a limited form of generic links, where the tokens were the wiki page titles. 'Local' links offer scope control on generic links, by only linking from such tokens within a given set of documents. Microcosm's clean and modular architecture applies linking to documents by means of a chain of filters which act upon the document: arbitrary sets of linkbases can be used by implementing them as filters and adding them to the chain. In this way, there can be a common linkbase of links for a document set of interest to all users, and upon this individual users, or groups of users, can add additional linkbases which are specific to them.

Microcosm was also open in terms of integration with third-party applications: because documents do not have the link data embedded, they can remain in their native formats, handled by their original applications. To provide hypermedia functionality when using these applications, either they must be extended (for example, using macro functionality) to be Microcosm-aware, or a general-purpose 'shim' could be used which worked via the system clipboard.

Such extension is still performed today to combine data sources: for example, it is common to overlay Google Maps with other geographical data [24]. Advertising services such as IntelliTXT[4] use client-side shimming scripts to add generic links throughout the body of normal web pages, without the need to modify the original document.

2.3.6 Hyper-G

Hyper-G [25] was a distributed hypermedia system which offered access via multiple methods: a native Hyper-G client, a web interface, and a cut-down Gopher interface. As well as documents within Hyper-G, links could reference documents on Gopher and HTTP servers outside of the system. This mapping to a web interface was an interesting development, as it was already allowing for the prevalence of web clients, and working with these rather than trying to enforce use of the native Hyper-G client. Hyper-G's data model consists of SGML documents, including searches; clusters of documents which form multilingual/multimedia aggregates; collections; and arbitrary, single-arc links. Documents are indexed upon insertion, and the system provides a boolean search service using these indicies, although the scope is limited to the local server. Collections provide a "unified view of distributed resources", by collating documents together in

[4]`http://www.intellitxt.com/`

some order: they are compound documents. A document may be in many collections, and collections may contain other collections, to provide multiple variations or orderings, and to form hierarchies of structured data. Links are external to documents, and use anchors stored with the link as byte offsets into the document at each endpoint. Only the system, via the Hyper-G client, updates and deletes links, thus avoiding 'dangling' or broken links, although not to external HTTP resources. Because links are external, even though they are unidirectional, they can be found and followed in reverse. Links are intended to be used for cross-referencing, overlaid on the structure of collections. Hyper-G handles large-scale distribution with link integrity, as envisaged but unimplemented by Xanadu's back-end–back-end protocol, by means of a specialised naming scheme "scalable flood algorithm"; it ran systems with content to the order of tens of thousands of documents.

2.3.7 Auld Linky and AHA!

Auld Linky [26, 27] was a 'sculptural' adaptive hypertext system. Sculptural hypertext is a form of adaptive hypertext which starts with an assumption that all links should be presented to the user, then removes those found to be inapplicable based on the context of the link and associated documents. It acted as a link server, suitable for use with Microcosm as a filter, and used FOHM (section 2.3.11) to represent hyperstructure. The interaction with Microcosm shows that adaptive hypertext can be added as a link filter to a suitably modular open hypertext system.

Auld Linky has been compared to the AHA! adaptive hypertext system [28]. AHA! took a more deterministic, 'scripting' approach, where pages had behaviours which set user context allowing other, specific pages to be seen. Pages themselves displayed content selected from a mutually-exclusive set of representations. Linky's sculptural approach, however, resulted in the addition of links between the systems. There were concerns that these links would ignore proper contextual preconditions when crossing over to the other system, especially as the conditions are not semantic: rather than AHA! encoding that some page A tells the reader about 'cheese', and that page B requires the reader to have read about 'cheese', it merely encodes that page A allows the reader to proceed to page B. Millard et. al. also found that Auld Linky's representations of context needed the ability to be subclassed, such that more complex taxonomies could be supported: without this, a page which teaches 'wine' must also explicitly specify 'red wine', and any other specialisations covered, else the user will not be considered to have the context to view pages on such specialisations.

2.3.8 Trellis

Trellis [29] is an adaptive hypermedia model, and a pair of systems (αTrellis and χTrellis), based on the mathematical field of Petri nets; a generalisation of state machines consisting of places which may contain tokens, and transitions between places. Transitions are enabled when tokens are present on all incoming places, which means that they may be fired. Firing a transition consumes a token from all incoming place, and produces a token on all outgoing places.

Trellis emphasises the separation of content and structure. The content exists as a set of ‘content elements’; in χTrellis, these also have generally applicable anchor points defined within them. The structure exists as the Petri net, and the two are mapped together. Each place on the Petri net is associated with a content element: when a token is present in a place, that content element is visible. Petri net transitions take the role of navigational buttons; activating a button triggers the transition, and updates the visible contents based on the new token distribution. In χTrellis, each place may specify mappings between anchors for the content element it displays, and the transitions from that place. Because Trellis is built on a formal mathematical model, it is possible to perform static analysis on the net, and determine which anchors can be triggered immediately, eventually, or never; this can be used to style or hide them.

The separation of content and structure allow one to be replaced, while re-using the other. Not only can the same content elements be re-used with a different structure of links between them, but the structure can be maintained and the contents changed, for example, to allow for translations. The model also makes the distinction between semantics and pages found missing from AHA! above: the representation of a learnt topic, a place with a token; and the entity which sets that topic as learnt, a transition leading to it. As the model is amenable to analysis, it is possible to determine which other places should also have tokens if a cross-system jump is made. There is not any mechanism for subclassing topics in the α- and χTrellis versions; however the model’s formal basis allows it to experiment with mathematical extensions to Petri nets.

2.3.9 World Wide Web

Berners-Lee’s World Wide Web [2, 30] is the system which has become, in popular culture, ‘the web’. The Web was specifically designed with abstraction layers which would allow it to distribute to a global scale across heterogenous systems: in particular, a network-stream-based interface (HTTP) which allows a wide range of client and server systems to interoperate; and HTML, which (as designed) presents documents in terms of their content semantics, rather than layout specifics, thus allowing clients to adapt display to their capabilities.

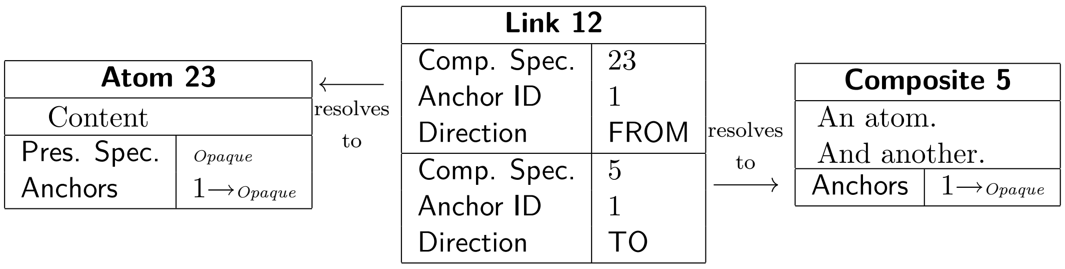


FIGURE 2.2: Dexter hypertext reference model components

The system also has a global naming mechanism, the URI, which is interesting in that it is a reference which may be taken out of the system. URLs were specifically designed to be 'manageably short' and consist of printable characters, such that they could be used in non-hypertext-aware systems, such as Internet mail, or simply noted on a scrap of paper. These addresses were intended to be more persistent than experience has since shown, largely as they have not been widely used as designed. Web addresses 'should refer to a document's registration with some "publishing" organization rather than any physical location, so that its location may later be moved', with the intent that the publishing organisation would redirect to the server actually hosting the document; they also 'should not contain any information...such as the particular formats available for a document'. This approach has found greater foothold on the Semantic Web, with services such as PURL: the URI for `title` in Dublin Core is `http://purl.org/dc/elements/1.1/`. `purl.org` is a publishing organization and, when the URI is resolved, PURL redirects the client to a file `dces.rdf` on the `dublincore.org` server. These guidelines help to avoid implementation and network infrastructure detail from 'leaking' into the document identifier. Berners-Lee clarified this intent in an informal 1998 W3C posting [31].

The widespread usage of the Web on the Internet demonstrates that it is quite capable of distributing to a global scale. However, we believe that there is scope for improvement in its hypermedia features: not that such improvements are strictly *necessary* for its function, but that they would be sufficiently *beneficial* as to be worthwhile. Bieber et al. give an example of an inter- and intranet web application for an insurance company [32], then speculate how high-level hypermedia could improve it. Some suggestions, such as node and link typing, are now better covered by current research into the Semantic Web, but others remain unaddressed: for example, transclusion is given as a way to generate consistent insurance policy documents from a library of standard clauses. Computed, user-private links are given as a way for a sales agent of the company to navigate from customer names to all of their insurance policies with the company.

2.3.10 Dexter hypertext reference model

The Dexter hypertext reference model[33] provides a formal representation of the abstractions used by the (envisaged) hypertext systems of its time. While it does not make explicit allowance for open hypermedia, it does provide first-class links, modelled separately from documents. It deals with systems as a stack of layers: ‘runtime’, ‘storage’, and ‘within-component’, and how these layers interact (presentation and anchoring). The model is of the classic ‘rigid’ hypermedia school of thought, in that it includes invariants which ensure formally correct behaviour. Indeed, the entire model is formalised in Z notation.

The hypertext is treated as a set of components, which are a union of component information and a base component: either atoms (document content, generally a ‘page’ granularity), links, or composites (constructed from other base components). Components—note that this includes links, thus they are first-class—have globally unique identity, expressed as a unique identifier (UID): this is comparable to the use of URIs in the Semantic Web. Resolver functions are used to map from some specification of a component (such as a query, or even just a plain UID) to component UIDs (for example, to perform searching); an accessor function can then retrieve a component given its UID.

Within-component addressing is provided by anchors, which have component-scoped identifiers (therefore an anchor can only be addressed within the scope of the component it applies to). The ‘value’ of an anchor—its mechanism for specifying some subset of the component—is opaque, and meaningful only to the specific within-component layer. ‘Specifiers’ consist of a component specification, anchor id, direction (whether this is the source, destination, both, or neither of the link), and possibly presentation specifications, which are opaque, and meaningful only to the presentation layer. A link is a sequence of two or more of these specifiers.

Figure 2.2 gives an overview of these components. On the left is an atom of content, with some presentation information which is of interest to the presentation layer and opaque to the rest of the system. This atom defines an anchor with an ID of 1, whose ‘target’ is likewise opaque to the wider system. On the right is a composite, which also defines an anchor, and since anchor IDs are scoped by component, it also has an ID of 1 without ambiguity. In the centre is a link, consisting of two specifiers. Both specifiers use literal component specifiers which point to a fixed component, and both specify the first anchor within. Because the top specifier has a FROM direction and the bottom a TO, the link is a simple directional one from the left Atom 23 to the right Composite 5.

Runtime behaviour, including editing, is addressed by an interesting approach of ‘instantiating’ components (and their anchors, as ‘link markers’) via an instantiator function. Components are instantiated within the context of an explicit browsing/editing session, and the instances are given a unique instantiation indentifier (IID). The instantiator function arbitrates between presentation detail provided to it within the context of the session, and that which exists in the component, in order to determine the final presentation information for the instantiated component: this allows for adaptive hypermedia. Instantiated components may be edited, but in order to make these changes permanent, the instances must be 'realized' back via a 'realizer' function. Although the model explicitly supports multiple, simultaneous instantiations of a given component, it does not cover how to resolve multiple realisations of modified versions: an 'edit conflict' in the terminology of wiki systems.

The Dexter model has been used as the basis for extensions [34, 35] to address some of the facets of systems such as the Web: for example, embedded link anchors. Section 3.3 of this report suggests some modifications to Dexter to represent the class of hypermedia systems containing Semantic MediaWiki.

Grønbæk and Trigg [34] define an object-oriented derivative of Dexter based on location specifiers ('LocSpecs') and reference specifiers ('RefSpecs'). LocSpecs permit within-component addressing, defining a location either by some named position (e.g. the fragment identifiers of an URI), by structure (e.g. "the third paragraph"), or by computation (e.g. "the location of the string 'hypertext' "). A specialisation of these, ComponentLocationSpecifiers, addresses components by one of the same mechanisms. RefSpecs wrap LocSpecs with some presentation detail within the scope of a parent component. They specialise as either anchors, endpoints, or composite references. Anchors exist purely as location markers, and may be 'transient' if constructed by run-time actions (e.g. to represent the point in the document at which the user clicked). Endpoints are comparable to Dexter endpoints, although they do not permit a directionality of NONE: the authors argue that there is no need to distinguish this from a direction of BOTH, although example links in [33] use NONE in a role more similar to FROM. CompositeReferences are used to construct composite components. Queries as location or reference specifiers are given as a way of implementing generic links; unlike Microcosm, however, the genericity of the link applies to the individual endpoints, not the link as a whole. It would be possible to represent a link with both specific and general anchors.

A hypertext is a collection of components. Components are much as in Dexter, although the 'base' component distinction has gone: composite components can contain links and other composite components. Links and composites are considered specialisations of components, with the content of a component being defined in the basic component class: thus such a link may also have content. Components can form simple tree structures, which provides composition with automatic deletion: if a node in this tree is deleted, then so are all child nodes, recursively.

LocSpecs are designed to have the same "free-floating pointer" behaviour as WWW URIs: they can be recorded and passed to friends via non-hypertext-aware means, and even constructed from scratch based on logical guesswork (the authors give an example of attempting to find the WWW site for a company named ACME by guessing the identifier `http://www.acme.com`). LocSpecs can also represent at-link-time computed documents, such as search results. With the WWW, one cannot link to search results, only the specification required to generate them, and they may be destroyed at the whim of the implementation. Conversely, in distributed hypermedia systems with a garbage collection-style approach, such results could be rendered permanent by creating links to them using location specifiers. RefSpecs make link structures external and explicit, with the same advantages as Microcosm (such as application to read-only data and, by extension, linking with other people's documents on the WWW). The authors expected that graphically browsing such structures would be a significant part of hypermedia, along with smooth integration of local and embedded, remote links.

D'Inverno et. al. [35] propose a model which addresses what they see as overcomplication in Dexter; however, they do this at the cost of representing a simpler hypertext system. Their model is based on binary links, which have specialisations to allow for typing. Links and nodes (components with contents) are completely disjunct types, and there is no readily-available mechanism for representing compound documents. Of note is the reduced rigidity of the model in some areas: for example, the link-following operation is permitted to fail with an error, not unlike the HTTP errors of the web. It also makes the concept of history explicit, rather than being abstracted into session data, and raises (but does not attempt to answer) several questions regarding how hypermedia history may function, other than the stack-pointer behaviour familiar to modern WWW browsers. In particular, the model allows for the removal of duplicates from the history list, instead keeping only the most-recent visit to a page; and for filtering of which operations add to the history.

The Dexter model is a relatively clean representation of hypertext, but more recent hypertext systems do not map completely onto it. Features such as generic links cannot be represented without revised versions of the model, and these revisions have oft omitted the runtime layer, which is not often considered when comparing hypertext systems in terms of the model.

2.3.11 FOHM

In 1996, with the Web showing growing popularity thanks to its distributed and interoperable nature, the Open Hypermedia Systems Working Group (OHSWG) began work on a standard Open Hypermedia Protocol (OHP) to allow intercommunication between open hypermedia systems.[36] However, this work focused only on navigational hypertext, with the expectation that it would be extended to cover other domains in future work. This triggered several researchers to instead consider the commonalities between the domains, and attempt to develop a unified data model.

A 'Fundamental Open Hypertext Model', FOHM [37] covers classic navigational, spatial, and taxonomic hypermedia systems. Spatial systems are those which use graphical positioning of documents to classify them within fuzzy sets: relations are generally implicit. Taxonomic systems deal with individual categorisations of documents: each user categorises documents as they best see fit, and the system creates a taxonomy (tree of specialisations and generalisations) of categorisations based on this. Where users disagree, the tree may gain 'perspectives': branches which are based on difference of user opinion.

FOHM offers a core model, based on data objects to which context and behaviours may be attached, and associations (links) which bind to references to data, again with context and behaviour. A mapping is provided from the three basic systems covered to the core model and back. Because hypermedia operations can also be described in terms of the core model, it is possible to deal with the hypertext entirely in terms of the core model, and map this to a specific system at display time.

FOHM's links can capture structure, to allow for spatial hypermedia's collections, such as sets, lists, and matricies. Defining mappings between domains stimulates "cross fertilization" of ideas: spatial sets are navigational *n*-ary links; lists fit the role of tours; do matricies suggest another navigational feature? The taxonomic concept of perspectives is seen as an analogue to context in navigational hypermedia, with the property that the user could follow links to alternate contexts. Using these commonalities is seen as the strength of FOHM, and the means by which it can address multiple domains fully without unmanageable complexity.

2.3.12 Other models

2.3.12.1 State machines

Moreau and Hall [38] represent hypermedia systems as state machines. A state consists of the current document, the current linkbase, and a set of documents; transitions between states represent the action of following links, which are bidirectional and represented as a pair of source and destination anchors. Each anchor contains a document pointer and offset. They use this abstraction to compare the expressiveness of various classes of hypertext systems. It is shown that there exists a 'compilation function' which converts embedded links into an external linkbase, and an inverse, 'decompilation' function, which embeds linkbases. As a result, embedded links are as expressive as simple linkbases.

The model is formally extended to cover other kinds of systems, and the expressivity reasoned about. Simple adaptive links, where the linkbases depend on browsing history, are shown to be no more expressive. This includes Trellis, as the 'linkbase' is a function of token distribution in the Petri net, itself a function of transitions triggered, and therefore browsing history. Generic links are as least as expressive as embedded links, and potentially more so, as they can introduce additional relations. Whether these relations are usefully expressive depends on the notion not of hyperstructure validity (e.g. the endpoints exist), but *semantic* validity: that the link *makes sense*. WWW browser-style history mechanisms, while acknowledged as a feature not of the hypertext but the navigation tool, are proven to give a greater class of hypertext expressivity: embedded links are as expressive as regular languages, yet history mechanisms raise this complexity to the class of push-down automata.

Finally, Moreau and Hall consider functional links. These are defined as a pair of functions: a predicate which, given an anchor and history list, determines if this anchor is the source of a link; and a relation which, given the same inputs, returns a set of destination anchors. Functional links offer the greatest expressiveness, equal to the class of finite Turing machines (as hypermedia documents and linkbases cannot be infinite).

2.3.12.2 XLink

XLink [39] is a W3C recommendation for representing link structures within arbitrary XML documents. It is implemented in terms of XML attributes, which allows it to be applied to documents without modifying their element structure. It covers both simple (WWW-like, unidirectional, embedded, single-target) and extended (unconstrained) links. There are four basic types of object involved in XLink links: resources, which indicate document-local elements participating in the link; locators, which address remote resources also participating; arcs, which define how the link is traversed; and titles, which provide human-readable labels. To provide embedded link behaviour, simple links act as local resources, including all their element content.

Links consist of resources, locators, and arcs; a link may, however, have no arcs, and less than two endpoints, to allow for incomplete placeholders, or for links which only provide attribute annotations. Arcs, and the link as a whole, may have a ‘role’, which corresponds to an RDF property: this allows the arc, or every arc in the link, to be interpreted as a RDF triple. There is a predicate which indicates that the target is another XLink linkbase which should be applied to this document, comparable to `owl:imports` for ontologies. Links may have behaviours—covering common cases such as new windows, replacing previous contents, and delegating to document-specific behaviour—which are ‘actuated’ by some mechanism, such as the document being displayed, or at the user’s request.

Unlike Dexter, which is a formal model, XLink specifies a vocabulary, and relaxes many correctness constraints, such as endpoint counts. XLink allows for browser-time behaviour, as in FOHM, but does not make explicit allowance for contexts, nor composite documents. It has been suggested, and demonstrated in a simple case [40], that XLink could act as a common interchange or export format for open hypermedia systems. This would aid the cross-system interoperability which is one of the goals of open hypermedia.

2.3.12.3 COHSE and Magpie

The COHSE project developed a ‘Conceptual Open Hypermedia Service’ which allowed linkbases to be layered over web pages [41]. It used thesauri to be able to refine or broaden terms as needed to seek an optimal number of links-to-concepts to add to a page. The concepts are then mapped to a set of target documents, and these are then culled for quality using ‘editorial knowledge’ before finally being linked into the HTML document. Of note is that the linking information is entirely external to the documents, with anchor points being implicit based on content matching, and sets of links can be added or removed by enabling or disabling linkbases.

Magpie [42], a similar system, highlighted terms recognised from a pre-provided ontology, with the ability to toggle the display of individual top-level classes. As terms are spotted, they are added to an adjacent list interface, intended to augment browser history, and which can also show related instances. For example identifying a person may also list projects they worked upon. Direct manipulation of highlighted terms in the page is also possible in order to display a listing of items associated via a given pre-specified relationship, such as a projects in the same research area as an identified project.

2.3.12.4 ScholOnto

ScholOnto [43] is an ontology for argumentative hypertext within the domain of academic publications. It defines a vocabulary for explicitly semantically annotating the relationships between research claims, allowing discourse such as confirmation, refutation, and extension of claims to be expressed. The intent of such annotations is to aid discoverability of useful and related papers as academic research moves towards ‘open access’ models where large bodies of documents are available and begin to strain the usefulness of simple keyword, category, and citation-following systems. This provides a solid use case for typed metadata links which can be traced in both directions, allowing one to track down the foundation upon which new claims are based, and which future developments they lead to.

2.4 Conventional wikis

While not precisely defined, the general understanding of a wiki is an online hypertext which anyone can edit. Wikis are constructed as a set of pages which are interlinked within the system, and authoring is performed through a web interface, usually using a markup language designed to be less cumbersome and requiring less expert knowledge for basic tasks than HTML. Most wiki systems also provide a degree of per-page version control as a mechanism to limit the damage caused by malevolent editors.

2.4.1 Systems

2.4.1.1 WikiBase

WikiBase[5], originally WikiWikiWeb is the name for the original wiki system developed by Ward Cunningham for use on the Portland Pattern Repository website, and thus the de facto standard for defining the term. It was designed to be a simple, collaborative platform for a mailing list of software developers discussing programming patterns to write about them with a greater degree of permanence, and was opened to them in 1995. To this end the system largely eschewed, and mostly continues to do so, complicated markup or much functionality beyond being a set of almost-plain text documents of which new versions could be submitted.

Hyperlinking between articles is achieved by writing the target page's name into the flow of text in title case with no spacing, known as 'CamelCase'. This means that, in this earliest form of wiki software, linking to a page is as simple as mentioning it: every page effectively has an implicit generic link to it from its title. Limited stylistic markup is handled in an ad-hoc fashion with rules such as a number of adjacent quotes (`'`) around a word marking it as italic: `''like so''`.

Many following wiki systems, such as MoinMoin[6] and UseModWiki[7], remained close to this model, in particularly its use of CamelCase linking.

2.4.1.2 MediaWiki

The Wikipedia online collaborative encyclopædia project originally used UseModWiki as its engine, but migrated to a bespoke PHP wiki implementation during 2002–2003. This system became known as MediaWiki.[8]

MediaWiki does not support CamelCase linking to page names; instead, links must be explicitly marked with square brackets. For example: `[[Camel Case]]`. Its markup remains ad-hoc, with traits such as using sequences of quotes for italic text or splitting paragraphs by leaving a blank line being carried forth all the way from WikiBase. However, it has grown considerably and can result in pages which are as structurally complicated as HTML documents, containing tables, inline images, links with differing anchor texts and targets, and stylistic markup without the benefit of a consistent syntax such as SGML or XML. This growth can be built upon yet further from the base system by using "extensions": additional code which hooks into the software to provide further functionality.

[5]`http://c2.com/cgi/wiki?WikiBase`
[6]`http://moinmo.in/`
[7]`http://www.usemod.com/cgi-bin/wiki.pl`
[8]`http://www.mediawiki.org/wiki/MediaWiki_history`

MediaWiki also offers a macroing capability known as “templates”. Pages can be included inline by linking them with braces instead of square brackets, and by default they are sourced from a dedicated Templates namespace. Templates may be provided arguments which are then substituted into placeholders in the template. On Wikipedia, and sites using a similar set of extensions, templates may even have limited computational ability, such as “if” clauses. This transformation occurs at the source level and, due to the many quirks of the markup language, may lead to some remarkably convoluted pages in order to achieve the desired effect.

For example, this is a small, single-line excerpt from a template created by the author:

```
{{#if:{{{image|}}}|{{TR}}{{TD}}[[image:{{{image}}}|250px]]|}}
```

It’s purpose, in the middle of a table, is to add a row, cell, and image should one have been provided as an argument, and it follows the documented idioms for achieving this type of goal. The triple-bracketed terms are placeholders for parameters; the pipe within the brackets provides a default blank value should the parameter be omitted rather than the parameter’s own name. This is necessary to get a ‘false’ value for the if-test. Note also that the template row and column, TR and TD, are themselves template invocations. This is because the table markup in MediaWiki uses pipe and hyphen characters which would be ambiguous with the surrounding markup, and only work at the beginning of a line. Those template pages contain a necessary linebreak and pipe character. The entire template is highly sensitive to issues such as whitespace and has many cases where the markup must be yet further complicated by the addition of HTML comments to ‘hide’ line-endings from the wiki parser while trying to retain some semblance of readability.

2.4.1.3 Weerkat

Weerkat [44] is a highly modular wiki previously developed by the author for experimentation with wiki development. Most aspects of the system are implemented as pluggable components, including parsing, rendering (which uses an incremental chain, the chain itself being a plugged component), storage, conflict resolution, and user management. Given this, we build upon it for the prototype developed in later chapters; this newer version is distinguished by the name *Open* Weerkat.

The default markup used by Weerkat is a consistent syntax with a simple grammar comparable to S-expressions. The system included an experimental ontology-based generic linking capability where terms defined in an installed (but not editable from the web interface) RDF/XML ontology were automatically linked to concepts in an integrated ontology browser. These automatic links would also display certain annotated properties in-line when hovered over with the mouse cursor. Since Weerkat does not handle type information like most semantic wikis, we do not list it here under that category.

2.4.2 Research

As a relatively newly emerged class of hypertext systems with a strong social aspect and a tendency toward exposing their data and editing history openly, wikis have been subject to various aspects of research.

2.4.2.1 User studies

Désilets, Paquet, and Vinson studied the usability of a conventional, WikiBase-style wiki system by inexperienced, non-technical users.[45] They found that non-technical users with minimal training could use the system to author an interactive fiction hyperdocument successfully.

The main source of problems they identified were related to creating and managing links; using appropriate syntax to trigger some text to be a link, renaming pages and links to them, changing the anchor text of links, and confusion between these aspects of the link. The unidirectionality of links limited the usefulness of their mapping feature to show the user adjacent pages in the hypertext.

Our experiments in chapters 6 and 11 on systems with support for finding incoming links (MediaWiki and Open Weerkat respectively) show that this aspect can be resolved technically, but we have limited overlap in the general editing behaviour due to differing tasks and systems. For example, renaming pages was not part of the process. We, like they, discovered that users may sometimes have problems with distinguishing the target of the link from the anchor text of the link, albeit to a lesser degree, perhaps due to a more experienced audience.

Research by Gaved et. al. sought information on user behaviour in wiki systems with geolocation capability, oriented around providing local information.[46] They achieved this via an e-mail survey directed at developers and administrators of such systems on a mailing list. Their findings acknowledged the existence of “stub” pages which have been created by users as an implicit form of request that another contributor provide information on the subject, consistent with explicit justifications for this behaviour at varying levels of structure (empty sections, dangling links) from participants in our studies.

In addition, some light statistical data collection shows a sharp expotential drop-off in edits (strictly, nodes created) against authors, consistent with our more in-depth findings on Wikipedia. They proposed a typology of wiki users, with categories such as those who create many stubs to cover a wide area, those who perfect a smaller number of pages, and those who perform “housekeeping”, managing links between and maintaining articles.

2.4.2.2 Statistical analysis

Ortega and Gonzalez-Barahona performed a quantitative study on Wikipedia's history, looking at editor changes over time.[47] They look first at splitting editors by their permissions level, and that as users are promoted to administrative status their contributions to regular articles fall off. They then split editors into five buckets by number of contributions, and look at these over time. This shows that the number of editors making very few edits is growing faster than any other group, and is a rising percentage of all contributions: this is comparable to the "long tail" result in our own quantitative analysis showing that there is a strong power relation between the number of editors and how few edits they make each.

They also compared across languages, to show that in Swedish Wikipedia, the greatest fraction of total edits is still made by highly active editors, unlike English Wikipedia where it has been overtaken by the combined edits of many less active editors. Their research also suggested that the most-active editors in each group remained active over time, but editors who are highly-active may have initially had periods of lower activity.

Wilkinson and Huberman have also looked into Wikipedia across time, from an article perspective and demonstrating the effect of an article being marked as "featured".[48] A reasonably intuitive finding is that featured articles have more edits and more distinct editors, and they also show that there are more edits in close temporal proximity to each-other on such pages. They also show that a small number of articles attract a large number of edits, and use this as an argument that the most important articles are subjected to more critique and quality. As the number of edits to an article increases, its visibility rises and provides a feedback mechanism encouraging further editing. This is held over time: not only is the number of articles over the number of edits a lognormal curve, but the mean edits per article rises logarithmically with article age.

Adler et. al. criticized the limitations of assessing editors by their edit counts or quantity of text added, and proposed a metric of *edit longevity* instead.[49] With this they sought to recognize the efforts of editors such as the "housekeepers" identified by Gaved et. al. who may make very small changes which still increase the quality of the content. It also allows spammers and vandals to be distinguished since their changes will have very little longevity, being rejected by the community. The found that about half of what most authors contribute survives to persist in the article, and that automated editing "bots" can form outliers in both forming many contributions which are kept, and also generating large quantities of vandalism.

They did not study quality of editors against number of editors, but did find support that prolific editors tend toward being either all "good" or all "bad", and that plotting quality against individual edits shows that most edits are "good". 10% of revisions had a quality of less than 5%, and 66.67% of revisions had a quality of over 95%. This former finding is higher than our own estimate of vandalism in section 5.6.3 of our macro-scale experiment (5%), as our categorisation only looked at a cruder metric of complete, immediate rollbacks.

Brandes et. al. studied relations between editors in the Wikipedia dataset to find groups of co-operating and conflicting contributors.[50] By graphing these contributors and connecting them with edges where they remove part of another's edit, they are able to show a visual distinction between articles with strong differences of opinion, and those that are merely being heavily modified. They compared the degree of conflict to the quality labels applied to the articles by the Wikipedia community, and found some support that higher-quality, featured articles had less inter-group conflict.

Suh et. al. have made an argument that Wikipedia's growth is slowing, supported by the number of active editors reaching a plateau and the growth of articles themselves slowing.[51] Like Ortega and Gonzalez-Barahona, part of their research involved bucketing editors into wide categories by number of edits made. Suh et. al. then also looked at what proportion of edits from editors in each group are *reverted* over time, and demonstrated an increased "resistance" to edits from less frequent editors. Not only do editors who make fewer contributions get reverted more often, but the separation of this proportion from editors who make more contributions has widened over time. A by-group breakdown of *active* editors—ones which have made at least one edit within a given month—shows that it is these smaller groups which are also substantially more active. This is subtly different from the finding that there are many more editors who have made few edits, since the requirement for activity means it does not "accumulate" editors who once made a single edit then never returned, and will "retire" prolific editors who have since stopped or taken a break.

Another aspect which they conclude is leading to a slowdown in Wikipedia growth is increased overhead in managing the size of the site. While we look at the cost of maintaining the hyperstructure in our macro-scale experiment—the links, categories, and lists—they looked at rise in activity of administrivia such as deleting pages, protecting pages from editing from new users, and blocking users. They also argue a rise in costs of co-ordination and bureaucracy such as forming and discussing policies for the community.

Fong and Biuk-Aghai looked at categorisation and differencing of Wikipedia edits.[52] This work postdates our macro-level study. Their approach differs from ours in that we first parsed the article, then measured changes to the plaintext. We used the parsed information from the markup to detect other types of changes.

Their approach is to first perform a lexing stage, then perform differencing at a sentence and lexer token level. From this they can determine low-level categories such as insertions, deletions, moves, and replacements. At a higher level they can determine that certain patterns of insertions, for example Mediawiki's square bracket tokens around a word, are link additions. Other syntactic patterns allow them to detect category or interwiki link changes, and guess at when a link is being disambiguated.

The differencing capability results in clearer displays of changes than are available in stock Mediawiki thanks to a greater degree of sentence awareness. They make comment that for deployment performance would be improved if the lexing for the differencing algorithm were performed when a page is saved, and cached. Open Weerkat actually works this way, keeping pages stored as completely parsed syntax trees, and thus in theory is amicable to adopting such a differencing system with marginal changes: differencing is already a pluggable module in the Weerkat design.

While they do not present large-scale statistical results, they performed a small-scale evaluation against the expectations of a set of human classifiers, with an average agreement of 84.1%.

2.4.2.3 Hypertext capabilities

Tolksdorf and Simperl devised a set of requirements for a *semantic hypermedia* system and evaluated the World Wide Web, Semantic Web, and wikis against these.[53] These requirements covered node and link typing, transclusion, annotation of resources, computed links, separation of nodes and links (i.e. not embedded), global and local overviews, history-based navigation, trails, and editing from the normal viewer.

They concluded that, while improving upon deficiencies of the Web, (semantic) wikis are more *closed* systems with no interaction capabilities to external services or other wikis, and stressed the need for first-class linking, distribution, and a common API and data model. We make a similar call for moving toward distribution in our own future direction in section 12.3.4.

The current state of Open Weerkat compares favourably to semantic wikis in terms of their requirements. As an open hypertext system, Open Weerkat stores links distinct from nodes, albeit with a caveat compared to an idea open hypertext system than conceptually “within page” addressing requires embedded *anchors*; other wiki systems use entirely embedded links. Our support for transclusion is also more oriented toward practical and common use than templating for content-sharing, which they categorise as being used “sporadically”. Finally, since we take great care to precisely define URIs for our pages across revision history, they are “on the Semantic Web” and can be annotated by making statements about them. Within the wiki itself, the model allows this using `.meta` pages, although the current prototype does not recognize these as being related to their parents.

2.5 Semantic wikis

There has been much work on attempting to extend wikis to support semantic relations; this section touches on some of the more notable examples. There is a notable lack of consistency across systems of basic features one might expect: versioning of not only node content but semantics, even if out-of-band (not in the page's source markup, where they would be versioned by normal wiki features); backlinks; typed inter-node relations as navigational hyperlinks; typed instances/relations/attributes; first-class types; import/-export of RDF, RDFS, or OWL. In general, these systems are academic experiments and, as such, are neither complete nor longer maintained.

2.5.1 IkeWiki

IkeWiki[9][54] is a 'complete rewrite' in Java of MediaWiki (the non-semantic WikiWiki system which powers Wikipedia[10]). It is one of the systems where types are first-class (they take the form of nodes in the wiki) and can thus themselves be annotated and described exactly as instances. The system can select a context-sensitive 'view' based on the semantic annotations of a resource, such as a 'taxonomy box' for biological entities. The domain and range of typed links can be specified, and then used in automatic consistency verification. Link targets of an appropriate type for the domain can be automatically suggested. Unfortunately, the reverse (suggesting the link type from the range and domain) cannot be performed: if a user links a person with a novel, the system does not suggest that the relation might be 'author' or 'editor'. This is, however, outside the scope of description logic-based inference. IkeWiki is still in development.

2.5.2 Makna

Makna[11] allows for more flexible semantics than many of the other wikis considered here: statements can be made both inside the page's markup, and separately, outside it. Because of this, it is possible to express statements about subjects other than the wiki node URIs. The semantics can be exported (in RDF/N3), optionally including inferred, not just explicitly provided, statements.

External ontologies can be referenced, but hyperlinks to them (e.g. in the side panel, if a predicate from such is used) will point to the concept's URI, rather than a node on the wiki. While this is arguably correct behaviour, from a user perspective it is a frequent source of 404 errors; as many concept URIs, including in the pet ontology used in the demonstration site, do not resolve to web documents, and even the RDFS

[9]`http://ikewiki.salzburgresearch.at/`

[10]`http://www.wikipedia.org/`

[11]`http://www.apps.ag-nbi.de/makna/`

ontology just presents the user with a page of RDF/XML. Interestingly, this does not hold for a link created within the body text: it takes the form of a link to a local wiki node. This makes for a rather unhelpful disconnect between the hyperlink target of the same semantic statement when made within the flow of prose, and its partner in the 'properties of this page' dialogue. If the object of the statement does not exist as a node, yet is created via the page text, the new node appears to bear no explicit relation to the concept in the ontology it related to in the origin page.

Features intended to mitigate Makna's complexity include semantic markup references during editing, and apparently using the 'rdfs:label' property to display predicates in a 'friendly' way.

2.5.3 Kaukolu Wiki

Kaukolu[12]'s [55] user interface, once loaded with RDF Schema declarations, can use JavaScript to provide 'autocompletion' of triples. For example, having typed in a known subject, autocompletion can provide a list of the predicates whose domain includes the type of the subject. This is a nice way to encourage users to be consistent with the terminology they use, and reduce the need for co-reference resolution.

Semantics are encoding in a N3-like plaintext format within the nodes, with plans to expand this to handle a fuller set of N3, and be more tolerant of natural language constructs. It is possible to create "semantic keywords", which are basically more readable aliases for full URIs that can be determined by querying an external semantic web service. Schema data may be obtained by importing RDFS files into nodes, but this mechanism is currently crude, expanding the URIs fully rather than using semantic keywords. RDF export is also possible.

2.5.4 COW

COW[13][56] is an ontology editor with support for communal editing, rather than a wiki with semantics—it has approached the combination from the other direction. Resultingly, while weak as a wiki it is one of the few systems which actually uses type information for the instances in the system: form fields are presented for attributes defined in the class of the instance being edited, along with information on their expected range.

[12]`http://kaukoluwiki.opendfki.de/`

[13]`http://www.informatik.uni-freiburg.de/cgnm/software/cow/`

2.5.5 OntoWiki

OntoWiki[14][57] is a community-editable knowledge base: the interface is geared around dealing with metadata rather than content. It has some interesting features for actually using the metadata, such as displaying the locations of entities with geographic position information on an embedded Google Maps display, and displaying those with temporal information on a simple calendar. The wiki functionality itself supports features such as inline editing, although the demonstration system did appear rather fragile and dependant on client-side scripts. Conventional full-page editing is possible which presents the relations as a list of textbox pairs; the right-hand textboxes may be augmented by type-dependant “widgets”, such as a calendar for selecting date values.

The system has also accumulated a few features from ‘social’ websites, such as user ranking of resources, and monitoring of estimated popularity by number of viewings. These statistics do not appear to be integrated into the resource metadata as further properties of the resource: they do not appear in the exported RDF, for example. Like MediaWiki, each page also has an associated discussion page. This may be used to annotate the statements themselves.

2.5.6 OpenRecord

OpenRecord[15] is more of a collaborative database than a semantic wiki. Most crucially, pages do not particularly have content in the conventional wiki sense—they have a summary (which cannot be annotated), and a number of sections. Each section may then contain views to the data in the system, but does not solely define the data itself; if multiple sections (quite possibly on multiple pages) refer to the same data, then a change to one is a change to all. There is no ‘main copy’ which the others reference; the approach is more akin to UNIX hard links, or garbage-collected objects, than to transclusion. Standard views include tables, lists, and graphs. Because most of the wiki content is in a database, pages are intended to be fewer in number than a conventional wiki, and in fact populate a sidebar for quick navigation; it is possible to navigate to a single instance to view its details as a single page, but this displays no more detail than the instance in the context of a table view.

These data are typed instances (by ‘category’) with loosely typed links and attributes (it is possible to force a value into a property which does not match its defined type). A few primitive types are supported (strings, integers, dates, etc.), and others may be defined, as types are first-class: custom types are always assumed to be to other instances, not literals, and such instances are generated (and hyperlinked to) based on a string representation entered by the user when setting a value. Backreferences, or inverse

[14]`http://3ba.se/`

[15]`http://openrecord.org/`

attributes, can be automatically created and maintained for all instance types: this is done by setting the ‘inverse attribute’ attribute of the attribute’s instance—attributes are also first class (e.g. attribute ‘childOf’ may have ‘inverse attribute’ of ‘parentOf’). This introspective ability is considered one of OpenRecord’s strengths.

Conflict resolution is also handled in a novel fashion. Should multiple, conflicting, concurrent changes be made to an attribute, all of them take effect: the cardinality of all attributes is unbounded, so both new values are added to the attribute.

It is possible to create additional views as plugins in JavaScript, but access to the system is deliberately promoted as being rather ‘shallow’.

2.5.7 Semantic MediaWiki

Semantic MediaWiki[16][1] is a MediaWiki extension (it is an add-on module, as opposed to a fork of the MediaWiki codebase). Interestingly, it can export individual nodes as RDF which is ‘browseable’, meaning that RDFS isDefinedBy triples are provided which link to URLs which, when resolved, yield further RDF which defines the external, yet related, entities.

Inline queries can be used to create nodes which display information sourced from other nodes: the semantic information can be put to an advantage within the system itself. The implementation approach as an extension of MediaWiki provides all of MediaWiki’s existing and fairly widely tested features, such as discussion pages and templating. Templating can be used to provide simple macros for expressing common relations while also displaying them in a useful manner; for example, recording an e-mail address triple, but also displaying it as a ‘mailto’ hyperlink.

2.5.8 General observations

There are at least three ways in which a semantic wiki can potentially make use of its type information:

- Objects have an explicit type, and would prompt instances to have all the attributes and relations explicitly specified in their class.
- Objects have an explicit type, and the class would implicitly consist of the attributes and relations common to all instances of it.
- Classes have explicit attributes and relations, and objects would be implicitly of that class if they had those attributes and relations.

[16]`http://ontoworld.org/wiki/`

However, while several of the systems supported typing of instances (usually pages), very little was actually *done* with this information, such as actually applying cardinality constraints to their attributes and relations. As such, they did not meet any of the above forms, and type information was largely meaningless to the systems themselves outside of the scope of literals.

Generally, the systems are very much HTML wikis with support for adding metadata about whole pages and, in richer systems, the relations between the pages. They are generally *not* hypermedia systems, and still treat non-textual content as special.

2.6 Seven issues

In his 1988 paper, Halasz identified seven issues for the next generation of hypertext systems [58]. He then revisited these as a keynote speech for the 1991 Hypertext conference. [59] We cover issues from both here, as we feel that semantic wikis are well positioned as a technical basis from which to look back upon some of these concerns. (We do not address Halasz's "market" issues from the renewed set, as they fall outside of our scope.)

The first issue identified was that of search and query of content and structure, which we shall not expand upon in depth. Content search has been well-covered by deployed systems such as Google, and linked data and open hypertext mean that more advanced searching can still be addressed by external tools.

Second was composition of articles from smaller parts, which was supported by models, and partially by systems, of the time, but has since declined in the Web with pages typically being constructed using ad-hoc server-side scripting rather than explicit hypermedia semantics. Even in web systems performing composition from structured components, such as content management systems, the compositional relationships are not exposed in an explicit and generally processable manner beyond the boundary of the server: the view to the web client is 'flattened'. Some wiki systems offer a degree of support for this via templating mechanisms, and in this paper we define a wiki system with rich support for a variety of compositing mechanisms, the prototype of which implements transclusion. We cover the possible approaches in section 7.7.6.

The third issue was that of virtual structure: implicit or generic links, and contextual linkbases. These issues have previously been addressed in systems such as Microcosm [23] and COHSE [41], although have not met popular use. Our prototype supports generic linking, and the role of linkbases can theoretically be performed by sourcing link data from an external RDF source in the model.

Issue four was computation, which is well-supported on the current Web with a wide variety of server-side facilities such as CGI, and client-side facilities such as JavaScript, hence we shall not expand upon it.

Fifth was versioning, including branching to alternate configurations. This is an area in which wiki systems have made an advancement over the Web in general, as at least limited versioning is part of the general wiki design, necessary to protect against casual vandalism. As a wiki, our model and prototype support this. Halasz commented that “maybe we’ve all lived with UNIX, MS-DOS and MacOS so long we don’t know what we’re missing”, but this environment is now beginning to change with primitive automatic versioning systems become standard in desktop operating systems. We believe that tackling branching and merging of versions is a worthwhile goal for future development of our system.

Issue six was collaboration, a fundamental aspect of wiki systems. Basic concurrency control and a degree of mutual intelligibility are fundamental to wikis to meet the “web page anybody can edit” goal. Our research in chapter 6 shows that a degree of collaboration is implicit, with editors using page state to communicate intent to potential others; a certain degree of higher-order explicit organisation can be achieved non-semantically using templates. Systems such as Mediawiki also provide per-article talk pages, and limited change notification support.

There is still progress to be made in this area. MediaWiki markup can escalate to considerable levels of complexity for less trivial tasks, restricting some editing operations only to more experienced users. For example, we found in chapter 6 that the use of templates could be troublesome. Modifying these templates involves even more complicated syntax. For example, the markup to display an optional template argument ‘year’ with a label if it exists is: `{{#if:{{{year|}}}|Year:  {{{year}}}|}}`

There is also room for improvement in providing richer semantics for editing intentions and discussion of pages beyond free text at article granularity. The ability to subdivide articles into component parts is naturally helpful here, tying this to issue two. Long-term transaction support can be added via branching versions (part of issue five) to allow editors to work on local versions of a page when making extensive edits before merging back their changes. This necessarily requires capable versioning support comparable to software development version control systems, and thus relates to issue five.

Finally, Halasz’s seventh original issue was that of tailorability and extensibility: Emacs-like scripting, schemas, templates, paths, and guided tours. Scripting of hypertext system interfaces is, if ad-hoc, these days possible through custom javascript injection support in popular browsers, such as the Greasemonkey system in Firefox, although it is still an expert task to author such scripts, and the goal of novice-friendly scripting is possibly unobtainable. Templating support, as we have mentioned, exists in some wiki systems, although currently the mechanism for creating templates is moderately advanced. In his revisited look at these issues, Halasz observed that paths and tours can be provided by external services in an open hypermedia context, and we agree with this assessment.

Halasz's first renewed point was an amalgamation of his original three, which we have addressed above. The second regarded open systems: how to decompose content into parts, and how to communicate those parts. We look to the creation of a distributed wiki system in future work, section 12.3.4, but some of the technologies are already in place. In particular, RDF provides a standard format in which to exchange semantic web data, and some current semantic wikis provide one-off import and export capability in this format.

The third renewed point was the same as the original sixth. Point four regarded the manipulation and navigation of large hypertext structures. With the benefit of hindsight, the Web, and large wiki installations like Wikipedia, have shown that this is not a pressing issue for reading hypertexts: interaction is largely initially search-oriented followed by localised browsing, and manageable. It is possible that such tools could be useful when editing and rearranging document collections, but the lack of prominent effort expended towards developing such a tool would appear to indicate limited demand.

The fifth and final technical issue is scalability to large data. Halasz defined 'very large' as 'tens or thousands of thousands of documents'. For comparison, at the time of writing, Wikipedia—a single, non-distributed collection—contains over three million encyclopedic articles, contributing to a total of over twenty million total pages, including administrivia. The Web, of course, greatly exceeds this, although at the cost of not supporting features such as reverse links. While the issue addresses not just engineering for scalable algorithms, but questions of usability, privacy, and navigation, we feel that the success of the web and Wikipedia show that sheer number of documents is not a problem in and as of itself.

2.7 Conclusions

Beyond a general introduction to the research areas, this chapter has covered notable hypertext systems such as Xanadu, an early hypertext, 'transpublishing' system; Microcosm, an open hypermedia system; and the Web, a highly successful distributed hypermedia system. It has covered the classic Z-based Dexter model, and the FOHM model, which maps three subareas of hypertext (navigational, spatial, and adaptive) to a common model. Finally, it has highlighted the current state of semantic wikis; in particular, the active Semantic MediaWiki project, upon which the remainder of this document will largely focus.

The next chapter looks at correlations between these areas, starting with a more formal description of Semantic MediaWiki's capabilities in terms of the Dexter model. It then proceeds to discuss a broad set of standard features of hypermedia systems in terms of their usefulness in a Semantic Web, and specifically semantic wiki, context.

Chapter 3

Open Hypermedia and Semantic Wiki

3.1 Introduction

This chapter identifies correlations between the fields of Open Hypermedia and Semantic Web, with a bias towards semantic wikis. To better specify the capabilities of a typical semantic wiki, to demonstrate that wikis fit within the set of hypertext systems, and to provide a grounding for further exploration, Semantic MediaWiki is expressed in terms of the Dexter hypertext model, covered in chapter 2. In order to achieve this, first a small modification is proposed to Dexter to allow it to model WWW-style embedded links, where not only are anchors defined within the document, but the link itself also forms the only outgoing anchor.

This is then followed by a discussion of key hypermedia features, and their possible applications in terms of semantic wiki. The focus is not merely on representing hyperstructure in terms of Semantic Web constructs, as that would merely be a trivial change of storage detail. Instead, the intent is to preserve whatever semantic information may lie behind the hyperstructure in such a way that it can be understood by generic semantic web tools: for example, that a n-ary link relates all its participating resources in an arbitrary RDF graph viewer.

3.2 Adapting Dexter to embedded, anonymous links

As described in Chapter 2, Grønbæk and Trigg's extensions[34] only really deal with embedded *anchors*, and treat embedding links as composition. However, a WWW-style (and thus wiki-style) link's *FROM* endpoint is defined by its location. In this section, we present an alternative approach to modifying the Z-based Dexter model to represent embedded links (and their limitations) more accurately. We consider links as a superset of atoms, in order to capture the content which forms their source anchor. Their source content is *not* a base component because links cannot nest; nor can a single link's *FROM* endpoint be a sequence of disparate atoms (e.g. some text *and* an image).

For reference, a specifier in Dexter is defined as:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{SPECIFIER}} \\
\hline
\mathit{componentSpec} : \mathit{COMPONENT\_SPEC} \\
\mathit{anchorSpec} : \mathit{ANCHOR\_SPEC} \\
\mathit{presentSpec} : \mathit{PRESENT\_SPEC} \\
\mathit{direction} : \mathit{DIRECTION} \\
\hline
\end{array}$$

A Dexter link is defined as:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{LINK}} \\
\hline
\mathit{specifiers} : \operatorname{seq} \mathit{SPECIFIER} \\
\hline
\#\mathit{specifiers} \geq 2 \\
\exists\, s : \operatorname{ran} \mathit{specifiers} \cdot s.\mathit{direction} = \mathit{TO} \\
\hline
\end{array}$$

In the proposed approach, Component specifiers are redefined to have a *THIS* value, which indicates a *COMPONENT_SPEC* which specifies the component which is using the specifier. An *ANCHOR_SPEC* is meaningless in combination with *THIS*. *COMPONENT_SPEC_OTHER* maps to the Dexter given set *COMPONENT_SPEC*. An alternative would be to include a more complicated condition in *LINK_ADDS* below, which requires that there exists a specifier which resolves to the component whose *compBase* is actually that link (or a *LINK* whose *specifiers* sequence contains that specifier). Unfortunately, the resolver and accessor which map specifiers to components (via a UID) are in the scope of the *PROTO_HYPERTEXT*, and are not accessible at this level of the model: hypertexts reference links; links do not reference hypertexts. This check could be performed in the storage layer operations *CreateNewComponent* and *ModifyComponent*, but then is no longer being expressed as an invariant of the storage model: instead, of the interfaces for modifying it.

$$\begin{array}{l}
[\mathit{COMPONENT\_SPEC\_OTHER}] \\
\mathit{COMPONENT\_SPEC} ::= \mathit{THIS} \mid \mathit{spec}\langle\langle \mathit{COMPONENT\_SPEC\_OTHER} \rangle\rangle
\end{array}$$

The constraints on links must be modified, such that there is one specifier of direction *FROM* from the link itself, and one of direction *TO*. This models the simple binary links of the Web and Wikis, although 'fat' links with multiple destinations (but still only a single origin) can be supported with the minor modification of changing the specifier cardinality requirement to *at least* two. To do this, *LINK* is defined as a union of *ATOM* and the new schema *LINK_ADDS*, which describes the additional constraints.

$$\begin{array}{|l}
\textit{LINK\_ADDS} \\
\hline
\textit{specifiers} : \mathrm{seq}\, \textit{SPECIFIER} \\
\hline
\#\textit{specifiers} = 2 \\
\forall s : \mathrm{ran}\, \textit{specifiers} \cdot (s.\textit{direction} = \textit{FROM} \land s.\textit{componentSpec} = \textit{THIS}) \\
\qquad\qquad \lor\ s.\textit{direction} = \textit{TO} \\
\hline
\end{array}$$

$$\textit{LINK} = \textit{ATOM} \cup \textit{LINK\_ADDS}$$

As indicated by *THIS*, the source of a link is itself; it is the *ATOM* part of its definition. As a result, it should be treated as any other atom by the interface: displayed as a component, or as part of a composite component. It is, of course, also a link which leads from an anchor specifying exactly that atom.

We have designed these modifications so as to not disrupt the type-safety of the remainder of the Dexter model, and thus contain the required changes to the smallest possible set of schemas.

3.3 Modelling Semantic MediaWiki in Dexter

In order to more formally consider the current state-of-the-art of semantic wikis (wikis with some form of semantic web functionality) in comparison to hypermedia systems, it is useful to express an example system in terms of the Dexter model. As covered in section 2.5.7, Semantic MediaWiki is a well-featured semantic wiki system, building on a widely-used ‘regular’ wiki.

The main obstacle with expressing Semantic MediaWiki in a formal hypertext model such as Dexter is that the development approach has not been concerned with models; instead with adding features. For example, MediaWiki allows a form of transclusion for marked-up text portions, and one for images. However, there is no commonality between these kinds of links—they are distinct features. While presentation hints can be provided for images (e.g. display size, figure title), they cannot for text. In fact, the author counts as many as six distinct types of links in the Semantic MediaWiki system[1] with no commonality beyond a source document, which is always the document containing the link.

[1]Templates, external images, internal images, external URLs, inline queries, semantically-typed links

3.3.1 Nodes

In Dexter terms, a node maps to a composite base component which is a sequence of atoms (pieces of markup text) and links. As in Dexter, a component cannot contain other components: only base components (there are no sequences of sequences).

$$\begin{aligned} \mathit{BASE\_COMPONENT} ::=\ & \mathit{atom}\langle\!\langle \mathit{ATOM} \rangle\!\rangle \\ \mid\ & \mathit{link}\langle\!\langle \mathit{LINK} \rangle\!\rangle \\ \mid\ & \mathit{composite}\langle\!\langle \mathit{BASE\_COMPONENT} \rangle\!\rangle \end{aligned}$$

$$\mathit{NODE} = \mathit{BASE\_COMPONENT}$$

MediaWiki automatically provides a limited set of named anchors into the node, based on the headings in the markup; Dexter covers named component anchors in the component information:

$$\begin{array}{|ll} \multicolumn{2}{l}{\mathit{COMP\_INFO}} \\ \hline \mathit{attributes} : \mathit{ATTRIBUTE} \mathrel{+\!\!\!\!\!\rightarrow} \mathit{VALUE} & \\ \mathit{anchors} : \operatorname{seq} \mathit{ANCHOR} & \text{(Anchors within this component)} \\ \mathit{presentSpec} : \mathit{PRESENT\_SPEC} & \\ \hline \#\mathit{anchors} = \mid \mathit{first}\,(\operatorname{ran} \mathit{anchors}) \mid & \\ \hline \end{array}$$

Some kinds of link can use this as a within-component specification for their destination endpoint; it is the only kind of such specification supported (i.e. no character ranges).

3.3.2 Internal node links

Ordinary MediaWiki node links (both inter-node and—rarely—intra-node) are a narrow subset of Dexter links. Crucially, they are not first class: they are anonymous and embedded. They always have precisely two endpoints, with fixed directionalities: one, the FROM end, is the location of the link definition in the markup of a node, which forms a source anchor and endpoint; the other (TO) is explicitly specified. A method for expressing such links through minimal modifications to Dexter has been presented in section 3.2. Both component and within-component specification (*componentSpec* and *anchorSpec*) are always to a specific UID: a node title for the former, and an automatic, named anchor, if specified, for the latter. These endpoint specifiers are anonymous and thus cannot be shared, although two links may specify equivalent destinations.

3.3.3 Templates

MediaWiki supports templating, which is a macroing system which allows some of the behaviour of transclusion and computed documents. The most basic case is to simply include a target, ‘template’ node into the current node. While this may appear similar to a transclusive/automatic link, Semantic MediaWiki, in its default configuration, brings in an important distinction: if the included node makes some statement ‘hasColour red’, then this relation applies to the *including* node. The ‘this node’ context of inline queries is likewise affected. Resultingly, MediaWiki templates do not fit well as links, and are better considered as an editing convenience implemented as a preprocessing step.

Templating may also involve macro variable substitution and, with extensions, even expression evaluation and branching. This may have fit the ‘computation specifcation’ of a ComponentLocationSpecifier, as described in [34], but would require that template inclusion/evaluation were a form of linking.

3.3.4 Images

Internal image linking is in the form of a node link to a specific namespace,[2] which contains image atoms uploaded by users. It is largely the same as internal node linking, with the addition of optional presentation hints (*presentSpec*), such as display size, and caption. A full specification of the presentation hints supported, where *LENGTH* and *CAPTION* are the given sets of pixel dimensions and image captions respectively, would be:

$$
\begin{array}{l}
[\mathit{LENGTH}, \mathit{CAPTION}] \\
\mathit{RECTANGLE} == \mathit{LENGTH} \times \mathit{LENGTH} \\
\mathit{IMAGE\_BOX} ::= \mathit{THUMBNAIL} \mid \mathit{FRAME} \mid \mathit{BORDER} \mid \mathit{NONE} \\
\mathit{IMAGE\_LOCATION} ::= \mathit{RIGHT} \mid \mathit{LEFT} \mid \mathit{CENTRE} \mid \mathit{NONE}
\end{array}
$$

$$
\begin{array}{|l}
\hline
\text{—}\ \mathit{IMAGE\_PSPEC} \\
\mathit{box} : \mathit{IMAGE\_BOX} \\
\mathit{location} : \mathit{IMAGE\_LOCATION} \\
\mathit{displaySize} : \mathit{RECTANGLE} \\
\mathit{caption} : \mathit{CAPTION} \\
\hline
\end{array}
$$

$$\mathit{PRESENT\_SPEC} == \mathit{IMAGE\_PSPEC}$$

[2]Syntactically, this is the only trigger that the link type is actually that of an image.

Note, however, that this definition of *PRESENT_SPEC* does not allow for any other presentation hints: e.g. those on links. If *PRESENT_SPEC* is defined as a union of *IMAGE_PSPEC* and other types, then, strictly, it is only this subset of hints which may be applied to images. In MediaWiki, most link presentation is affected by the content of the link source, and the nature of its destination.

The transclusive nature of these links is not well covered by Dexter, which expects composition only in composite nodes, as defined by the node; considering the Image as such a composited atom is probably the closest match for this behaviour. In this case, the *presentSpec* belongs in the *COMP_INFO* of the image atom, rather than in a link to it.

MediaWiki also has optional support for linking to external images. Syntactically, URLs with particular file extensions are automatically treated as WWW-style image transclusion. In terms of the model, they are automatically-generated versions of the internal image links.

3.3.5 External URLs

An external URL link is ultimately identical to a WWW hyperlink. As with internal node links, they are embedded and anonymous; they differ in that their target is expressed as a URL, rather than a node UID (title). They may have some display text provided, else they will display the URL as a source anchor; ultimately, however, this is syntactic sugar for providing the content atom component of the link in the embedded link model presented in section 3.2.

3.3.6 Typed relations

As well as nodes having typed 'links' to literal values, which Semantic MediaWiki terms 'attributes', the system supports links to other nodes, termed 'relations'. (In a Semantic Web context, both of these 'links' are RDF statements: attribute predicates have a range of literals, and relation predicates a range of other resources.) These links are the sole user-typed, node-node relation in the system at present; the type of the relation is expressed by explicitly specifying a 'relation' node. This node both acts as a predicate URI for the triple, and as a navigational link in a 'fact box' in the interface. Other than that, they are identical to internal node links, described above.

Because Dexter links are base components, and thus can be components with component information, the predicate type can be expressed using the arbitrary attribute/value pairs. Dexter does not specify the type of the members of these pairs (they are merely members of the given sets *ATTRIBUTE* and *VALUE*), but one could conceivably reserve an attribute for 'type of relation', and provide a value which is a specifier for the node being used as a predicate.

$$[ATTRIBUTE\_OTHER]$$
$$ATTRIBUTE ::= RELATION\_TYPE \mid spec\langle\langle ATTRIBUTE\_OTHER\rangle\rangle$$

$$\begin{array}{l}
\textit{COMP\_INFO} \\
\hline
attributes : ATTRIBUTE \nrightarrow VALUE \\
anchors : \mathrm{seq}\, ANCHOR \\
presentSpec : PRESENT\_SPEC \\
\hline
\#anchors = \mid first\,(\mathrm{ran}\; anchors) \mid \\
\quad \text{(Require that any attribute specifying relation type does so as a specifier)} \\
\forall\, a : \mathrm{ran}\; attributes \mid a = RELATION\_TYPE \cdot \\
\qquad\qquad (\exists\, s : SPECIFIER \cdot attributes(a) = s) \\
\hline
\end{array}$$

The rationale for using a specifier is that it semantically identifies the target (although it may simply take the form of the node's UID); yet it is not desirable to use a full link as there are no meaningful endpoints. The 'fact box' is a feature of the interface, not of the actual hypertext, which uses this specifier to construct an appropriate link from the display of the predicate in the box to the (entire) node specified.

The relation indicated by MediaWiki categories is defined by a simple heuristic: nodes in the 'Category' namespace are assumed to be classes. A non-category node which is placed in a category is given a `rdf:type` relation (instance–class), and a category which is placed in a category is given a `rdfs:subClassOf` relation (subclass–class). Syntactic restrictions prevent it from being possible to place an non-category node in a non-category node: the 'place in category' link is special-case of the untyped, inter-node link that is triggered by the namespace of the target node being 'Category'. Effectively, categorisation is taken as a taxonomy, which is then expressed such in terms of classes, where the 'leaf' (most-specialised) nodes are instances. In either case, these are different syntax for creating typed links with particular predefined types.

Because Semantic MediaWiki has a one-to-one mapping between nodes and concepts in the ontology it forms, typed links can be created from nodes to literals: the system calls these 'attributes'. These are akin to the arbitrary attribute/value pairs which Dexter supports as component information on the *atom*, although in Semantic MediaWiki the attributes themselves are node identifiers: again, the use of specifiers is a good semantic match. (They cannot be modelled as links, as in the above relations, as literal values are not valid link endpoints; synthesising an atom with that literal as content does not encode the same information.) Semantic MediaWiki presents attributes as implicit links to the node for that attribute, which is usually itself annotated, e.g. with range information: again, this is considered to be a feature of the interface, not of the hypertext.

3.3.7 Inline queries

Semantic MediaWiki has a simple query language[3] which can be used for 'inline' queries[4], which are embedded in a page's markup, executing and displaying results at that location when the page is rendered. The query language itself is largely simplistic, with support for selection of results by the transitive 'category' relation, relation values, and attribute values. If the attribute values are of an appropriate type[5], then inequalities (e.g. "greater than 25m") can be used as conditions. It also supports disjunctions, and specification of additional predicates to retrieve for display.

These queries, and their results, can be embedded in nodes. This is comparable to a compositing link to a RefSpec with a 'computation specifier' in [34]; in plain Dexter, it can be thought of as a specifier where the resolver function returns a generated document which is itself a composite of *LINK*s to all results. Presentation hints control the display of the results, from inline lists, through tables, to JavaScript timelines[6]. It is also possible to conditionally include text if the number of results returned is non-zero, which can be modelled as a further clause of the computation specifier by considering such text as an *ATOM* part of the generated document.

Queries are not first-class; re-use can only be achieved via the template-macroing functionality. Macroing cannot be applied to the query itself, except for various 'special' variables, such as the UID of the including page, or the current year.

3.3.8 Summary

That it is possible to meaningfully express Semantic MediaWiki's key features in terms of slight modifications to the Dexter model demonstrates two key things. Firstly, that wikis can be considered hypertext systems, even if limited ones, by virtue of their Web heritage. The following sections of this document build on this by both considering the possible improvements which could be made to wiki systems by strengthening their hypermedia foundations, and by experimentally demonstrating that there is a cost to the currently weaker features.

Secondly, it shows that there is a meaningful similarity between a set of documents connected by typed links, and a graph of resources connected by RDF statements: in

[3]`http://wiki.ontoworld.org/wiki/Help:Semantic_search`
[4]`http://wiki.ontoworld.org/wiki/Help:Inline_queries`
[5]Semantic MediaWiki has an extensible measurement type system: `http://wiki.ontoworld.org/wiki/Help:Custom_units`
[6]`http://wiki.ontoworld.org/wiki/Upcoming_events`

fact, it is on this very similarity by which most semantic wikis operate. The remainder of this section builds upon this comparison by considering possible semantically-useful and valid parallels for other notable hypertext features.

3.4 Relating Open Hypermedia and Semantic Web

In the preceeding section, we suggested a parallel between the Open Hypermedia model of a graph of nodes connected by possibly typed links, and the Semantic Web model of a graph of resources connected by relations. Therefore, it should be possible to express the Semantic Web in terms of, or as an extension of, Open Hypermedia. As user-editable, WWW-level hypertext systems, semantic wikis also have similarity to this basic structure. This section offers some consideration of potentially useful mappings between these domains.

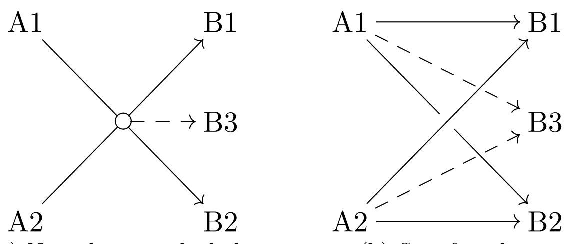
3.4.1 Nodes are resources

A node in Open Hypermedia is approximately a document; a Semantic Web resource, however, is a more general concept, which can represent any kind of addressable thing, be it abstract or concrete. As documents are a subset of concrete, conceivable things, resources are a generalisation of nodes.

Hypermedia has oft complicated itself with the problem of addressing at the sub-node level: e.g. a particular paragraph in a document. “The editing problem is probably the major limitation of the Microcosm model [regarding scalability].”[60, §7.2.1] Resources, however, are atomic; it is not possible for a triple to refer to anything smaller than an entire resource, because it is simply an opaque URI. If you wish to express such semantics, you must explicitly declare that the resource is some manner of composition of other, smaller resources. One cannot express “the second paragraph of this document”; instead, “the document consists of a sequence of paragraphs: #1, #2, #3; we refer to #2”.

Edit-time transclusion within the scope of a single wiki (specifically, within the scope of a single domain of editing privileges) offers a similar solution. Rather than attempting to reliably address a sub-node range, the range is split into a new node which is transcluded in place, without affecting the ability to edit the original node as a whole. This new node is now a first-class entity, and thus addressable without the need for sub-node specifiers.

Let us say that there is a node for the Perl programming language: it exists both as the concept of this language, and as some text describing the language. Within this is a section describing Perl’s Unicode support, which one wishes to address (possibly to specify that Perl’s Unicode support is immature, as opposed to specifying that *Perl* is



(a) N-ary hypermedia link (b) Set of triples

Figure 3.1: Links with multiple source and destination anchors. The dashed lines show the changes needed to add the extra target 'B3'.

immature). Rather than having to specify this by fragile means, such as an offset into the node structure, the section could be promoted to a named node. This named node can be transcluded back into its original position; by also presenting this transclusion at *edit time*, minimal disruption has been caused to the process of editing the Perl node as a whole.

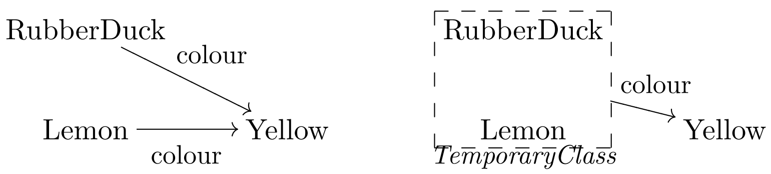
It should be noted that navigational links, from spans of just a few words, are unlikely to justify such an approach. In these cases, embedded links are probably a better approach to ensuring that the link anchor is visible, and thus considered, when editing the document.

> "In any system which uses embedded mark-up within the document to mark the position of anchors, then this will rarely be a problem since, in general, as the document is edited the mark-up will move around with the object which it is marking."[60, §7.2.1]

Moreau and Hall[38] describe a 'compilation' function, and its 'decompilation' inverse (formalisations of the 'joiner' and 'splitter' tools of Brown and Brown[61]), which provide a mechanism to convert freely between embedded and non-embedded links. They use this to prove that the distinction has no bearing on the expressiveness of the underlying hypermedia model; hence, as Brown and Brown found through an experimental implementation, "the Joiner and Splitter can be used to combine many of the advantages of separate and embedded mark-up"[61, §8] without penalty.

3.4.2 Links are triples

A link in OH is some relation between at least two nodes (or sub-node parts, as covered). The nature of this relation is conventionally only weakly specified: it exists for navigational, or possibly 'loose' classification (spatial hypermedia), purposes. Conversely, the links between Semantic Web resources are triples, which have a completely explicit type predicate, and exactly two endpoints.



(a) Independent links (b) Linking from a common class

Figure 3.2: Representing multiple link targets as a temporary class

In a system with four resources/nodes, ‘A1’, ‘A2’, ‘B1’ and ‘B2’, Open Hypermedia can represent the case where a single link exists from either A1 or A2 to both B1 and B2: navigationally, the user can move from A1 to B1 and B2, and from A2 to B1 and B2. This is depicted in figure 3.1a. Because Semantic Web has a simpler connector, the triple, it has to use more of these fundamental units in order to express the same connectivity: one triple for the A1/B1 relation, one for the A1/B2 relation, one for the A2/B1 relation, and so on: it is necessary to assert the cross product of the endpoints, as shown in figure 3.1b. Unlike the link, these triples are not a single entity: if a fifth resource/node, ‘B3’, were to be added, the OH link could easily be modified to also direct all A to B; for the Semantic Web, this higher-level abstraction does not exist, and more independent triples must be added. (It *is* possible to reify an *application-specific* higher-level abstraction, but this will not be understood as entailing the individual statements by generic RDF tools.)

A concrete example of multiple source anchors given in [34] is that of a generic link from either the text “hypertext” or “hypermedia” to some pages about hypermedia. In this case, it is considered that both hypertext and hypermedia are the same concept, so a better knowledge model would be to express this equivalence, and then just link from one of the pair. (In this specific case, expressing hypertext as a specialisation of hypermedia, and then linking from hypermedia, would probably best model the correct semantics.)

If it were desirable to draw a distinction between multiple sources of this nature not connected by some transitive relation, one approach is to create a new class of which both are members, and then link from that class. While this class is transient for the purpose of representing the link, if it is semantically sensible to link from multiple sources to the same (set of) targets, then there is probably some correlation between the sources which also makes placing them in an explicit class sensible.

Figure 3.2a demonstrates this: the apparently disassociated documents about rubber ducks and lemons share a link to the colour yellow. This link can instead be expressed as in figure 3.2b: the class containing rubber ducks and lemons is related to the colour yellow. It then becomes apparent that this class could possibly be made explicit as the class of yellow things, suggesting other suitable source anchors/class members.

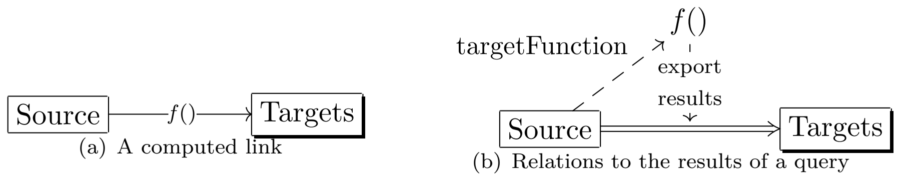


FIGURE 3.3: Computed links vs. query results

3.4.3 Computed documents; computed and functional links

Some hypermedia models cover ‘computed’ documents, where the content is generated upon request: WWW CGI scripts are an example. One of the main uses for such on the web is document search (even if for documents about products); others include minor customisation which do not affect the hyperstructure (e.g. purely stylistic user profiles), and remote procedure calls (e.g. most ‘web applications’). The latter are outside the scope of a knowledge base, and arguably outside the scope of a hypertext, using it only as an *interface layer* to some other system.

Document search, however, effectively returns an ordered set of endpoints for a given query (presented as links from the computed document). The actual information content here is similar to computed links (where the targets are determined at the time at which the link is followed), and to functional links (a generalisation where link endpoints from either direction are an arbitrary function of hypermedia system state) which perform search. Figure 3.3a shows a functional link, where the set of targets depends on the function $f()$.

The RDF model does not, however, allow for anything but completely static relations. Even if such queries are expressed in RDF (e.g. a representation of SPARQL), it is not possible to declare the object of a triple as the result of such in a way which generic RDF tools will understand. Instead, any kind of computed link must be ‘exported’ into static triples; it may be possible to retain the dynamic nature of computed/functional links if these exported triples are only accessed via some kind of accessor interface, such as a SPARQL endpoint which generates them on-the-fly. Figure 3.3b shows this: a relation to the function URI (displayed, again, as $f()$) refers to the function itself, not to its results. The fictional relation ‘targetFunction’ indicates here the function used to generate link targets from the source resource (although it should be noted that this is akin to embedded, rather than first-class, linking). In order to relate the source with the targets, the results must be converted into static triples.

Ashman and Verbyla [62] suggest that functional links could be used to group types of links—the one link provides endpoints which represent all connections of this type. While this may initially seem appealing, the resultant semantics are likely not what are intended: if every `foaf:mailbox` predicate for a set of people and mailboxes were handled by one link, then the meaning would be that each person has each mailbox due

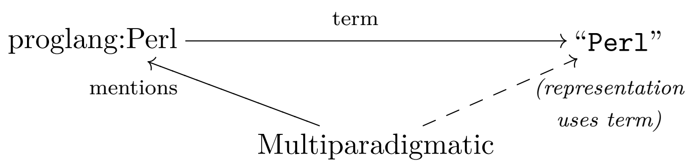


FIGURE 3.4: Example of Weerkat-style generic links

to the cross-product effect discussed above.

3.4.4 Generic links

Generic links automatically link from source tokens (e.g. particular words). Early versions of Weerkat, described in section 2.4.1.3, had a primitive implementation of this which would apply a (strictly navigational) link from a word to a concept in a provided ontology if the concept had a 'term' property matching that word as an `xsd:string` literal. RDF does not allow a more tool-agnostic approach, as literals cannot be used as subjects in triples; however, the Weerkat approach could be improved by also explicitly stating that a relation (e.g. 'mentions') exists between the node containing the term and the concept to which the term belongs.

Figure 3.4 graphs the triples in an example case where the token 'Perl' is linked to the resource 'proglang:perl'. The 'term' relation is part of the generic link and, in Weerkat, originates from the Wiki ontology. The resource 'Multiparadigmatic', a description of languages supporting more than one programming paradigm, contains the term, and thus a suitable 'mentions' triple is constructed by the system.

Microcosm [23] also defines 'local' links [22], which are generic links scoped to a particular set of documents. This effectively provides much the same control as namespacing on the term: for example, specifying the link only applies to 'apple' in the context of documents about fruit, without actually having to use the term 'fruit:apple'. Again, this is not expressible in RDF without tool-specific semantics.

It is interesting to compare scoping of generic link terms to sets of documents, an explicit form of disambiguation, to more recent attempts to disambiguate terms (under the name of 'tags') by attempting to implicitly find such document sets [63, 64].

3.4.5 Non-embedded, first-class hyperlinks

As we covered above, hypermedia links can be considered to map fairly directly to Semantic Web triples. In open hypermedia, such links are always first-class: they are named objects. In the Semantic Web, one can make statements about triples by reifying them [65, §7.3], although this technically does not specify an identity for the triple [3, §4.3]. Such open hypermedia links are also stored external to the documents, in linkbases: databases of links. In the Semantic Web, arbitrary collections of triples may be gathered from any source: the canonical non-embedded source is the *triple store*: a database of triples.

It is important to note that an n-ary link expressed as a set of triples cannot be reified in this manner as a single entity; rather, each triple gains its own, distinct identity. The implications for this, and possible resolutions (including reifying the set as a higher-level construct with a different vocabulary), are discussed in the following minor section, as they affect adaptive hypermedia.

3.4.6 Contextual/adaptive alternation and level-of-detail

Alternations based on context, detail, and such are facets of browsing a hypertext, not of the hypertext itself—the data model only expresses the metadata needed to make the alternations. Reification of triples allows expression of detail levels, prerequisites, temporal constraints, and such to apply to both concepts and links, but specialist tool support would be needed to make use of this information.

It is not clear that such features have any real meaning in a knowledge context. The fact that a link or resource should only be shown at higher levels of detail may imply that it is less significant, but this is not necessarily pertinent to reasoning over it. It may provide a way to reduce the knowledge base down to a simpler set to avoid excessive computational complexity, but visual importance and logical importance may not be related, and the latter is more likely to be dependant on the specific goal of the reasoner than to the triple or concept's overall role within the ontology.

For example, reducing taxonomic detail ("Harry likes cheese"; "Harry likes organic cheese") is unlikely to be useful, as the less-detailed information can be inferred from the more-detailed information as needed. Reducing non-taxonomic detail, such as trivia ("Harry likes cheese", important to the example ontology's domain; "Harry is in IAM", irrelevant), cannot realistically be done: the significance of a piece of information is highly dependent on the query. Omitting 'detailed' (low-relevance to ontology) statements such as "Harry is in IAM" is counterproductive if the query is to find people in ECS research groups who like cheese.

Prerequisites are primarily navigational and have limited reasoning meaning: if one cannot read about 'objects in Perl' without having read 'references in Perl', it follows that Perl objects are in some way more specialised or complicated than, or dependant upon, Perl references, but the nature of this relation is not explicit. Conversely, certain semantic relations *may* reliably indicate that prerequisite knowledge is, if not essential, at least useful, to understanding the description of a resource. In particular, specialisation ('red wine is a subclass of wine'; one should read about wines in general before specific wine types) and composition ('wheels are parts of cars'; one must understand wheels before they can understand cars) suggest sensible prior reading. Care must be taken with such automatic suggestion, as it lacks the subjective judgement of a skilled author. If the class of red wines were to be defined, using OWL, to be a subclass of the restriction of wines which were fermented with grape skins intact, then 'fermentation with skins' is applicable, but 'restriction' is not, if only because it can be assumed as common knowledge.

Adaptive hypermedia exposes a weakness in the approach of converting n-ary links to a cross-product of triples. For ordinary links, being able to provide multiple sources and targets is ultimately an organisational convenience: there is arguably no *semantic* distinction between a link from 'A' to 'B' and 'C', and a pair of links from 'A' which lead to 'B' and 'C' respectively. In either case, 'A' is related to 'B' and 'C'; for editing, these relations can be grouped back together into whichever possible n-ary link representing the same information is the most useful representation.

However, *adaptive* links may have subtler semantics, if the set of endpoints is an alternation. In this case, a link from 'A' to 'B' and 'C' will actually only have the semantics of a link from 'A' to 'B' *or* 'A' to 'C'; not both at once. It could be possible to express this as a relation between 'A' and the blank RDF alternation $\{B, C\}$, but this is not particularly useful representation for reasoners, as RDF Schema defines the various RDF containers as semantically identical[7]. These constructs only semantically define membership in the collection: the distinction between bags, sequences, and alternations is intended for humans. If one accepts that adaptive hypermedia information is for the benefit of presentation alone, then it should suffice to simply provide a representation (reification) of the higher-level n-ary link in some more suitable, but less generic, vocabulary. This makes explicit the desired n-ary link which can be formed from the triples, and gives it identity (i.e. makes it first-class). As such, it can be annotated as an adaptive alternation, and the specific presenting software can understand that only one of its possible triples should result in a navigational link at a time.

3.4.7 Transclusion

Since semantic wikis build semantic relations from hypermedia relations, we must now consider if transclusion has a useful semantic parallel. In metadata alone, transclusion-like constructs have no additional meaning. In OWL, import-by-reference statements (`owl:imports`) are an organisational convenience, to allow practical separation of ontologies, and any dependencies, into separate documents. Importing another OWL document effectively includes its definitions within the scope of the importing ontology: no separation is made. (RDF Schema, upon which OWL builds, also defines `rdfs:seeAlso`,

[7]"The formal semantics of all three classes of container are identical".[6]

but this merely indicates another resource (not necessarily resolvable) which *might* include further metadata about the subject resource: it is more of a 'related link'.)

In semantic wikis, we must consider the semantic meaning of representations which transclude statements about the resources they represent. Let us take the simple example of transcluding an abstract into a page about a particular E-Print:

```
MyEPrint abstract MyEPrintAbstract .
abstract a TransclusiveRelation .
MyEPrintAbstract a Abstract .
```

The 'TransclusiveRelation' type here indicates that the relation 'abstract' should transclude its object's representation into the subject's. This does not imply that transitive relations apply across the relation; it does not follow that MyEPrint is a member of the class Abstract. It *would* be correct, in this case, to say that MyEPrintAbstract is a part of MyEPrint, because this example uses representations which map to meaningful resources: there is such as thing as the abstract of this EPrint, such that one may make statements about it.

This part-of relation does *not* hold in cases where the transcluded representation is not of a meaningful resource: in such cases, the only sensible relation between the two resources would be a generic 'representation includes'. Consider, for example, a node about about a particular country, which transcludes a general information panel, 'CountryPanel':

```
include a TransclusiveRelation .
Brazil include CountryPanel .
```

Clearly, 'CountryPanel' is *not* a part of Brazil. Because transclusion operates on representations, cases such as the abstract only indicate meronymy (part-of relations) because 'abstract' is also applicable to the resource. It would be more correct to say that 'abstract' is both a transclusive relation and a specialisation of a part-of relation, whereas 'include' is only the former.

Semantic MediaWiki's 'semantic' templates, although strictly a macroing system, rather than transclusive, effectively act as role-propagating links; that is, a node 'A' including template 'B' with statement 'colour red' will be interpreted as 'A colour red'. To cover this use of 'semantic' transclusion, let us add two more statements:

```
include a RolePropagator.
CountryPanel a Country .
```

The type 'RolePropagator' is taken to mean a relation along which other roles may be inferred: if 'Brazil include CountryGeneral', and 'CountryGeneral a Country', then 'Brazil a Country', as any relation of 'CountryGeneral' may propagate across the 'include' relation. This models how templating works in Semantic MediaWiki, where this pattern is used[8], extending from widespread templating of infoboxes on Wikipedia. OWL 1.1[9] aims to include *property chain* (or *role*) *inclusion axioms* for representing this kind of property, although these allow *specific* relations to propagate, rather than all. This behaviour can be reproduced in existing versions by various alternate methods. [66]

This is not, however, good ontology modelling. It clouds the resource/representation distinction, and does not make proper use of classification: there is great temptation to include other common country statements in 'CountryPanel', rather than the 'Country' class. The statement that the country panel is, itself, a country, is nonsensical, and only makes sense in its correctly-transcluded context. Parameterised templates are used with this pattern to both assert and display other properties, such as the country's capital city, via the transcluded resource. A better approach would be to simply assert these statements on the country resource (possibly with editing help, as described in section 4.7), and then transclude a template whose only role is to typeset the information panel, using only the URI of the transcluder as a parameter. It may even be desirable to be able to indicate that resources of certain classes should automatically have a particular information panel transcluded into their representations. This approach emphasises the use of correct semantics, and using these to inform useful display, rather than 'hijacking' useful display to try to add semantics.

Hence, it would appear that transclusion *of itself* should not affect the relations between resources. While a useful tool for building representations, in cases where meronymy also applies to the resources being described or represented, it is possible to find a more specific relation between the resources (abstract of, ingredient of, city of) which is more accurately responsible for the part-of relationship. Conversely, in systems such as semantic wikis with tight resource/representation correlations, transclusion as meronymy can be meaningfully applied to representations where it does not make sense for resources: the *text* of CountryPanel may be part-of the *text* of Brazil, but the CountryPanel itself is not part of Brazil itself.

With regard to the hypermedia, rather than inference, effects of transclusion: MediaWiki also supports template 'substitution', which actually imports the template, with parameter substitution performed (with some caveats), into the source text of the including node. It is important to note that the relationship between the including and included nodes is lost, and that the benefits of re-use (such as storage efficiency and later corrections) are not available. The information regarding the origin of the text is also lost without manual documentation effort, including any parameters required for the more complicated templates. Substitution, aside from some technical workarounds, is

[8]This particular example is a simplified form of `http://ontoworld.org/wiki/Template:Infobox_Country`.

[9]`http://www.w3.org/Submission/owl11-overview/`

largely used to provide time-stamping by using a 'current time' global parameter inside the template. This could be better implemented by extending the implementation idea of 'hot' and 'warm' links [32] (transclusive links which are cached and refreshed automatically and manually respectively) to a 'cold' link, which takes a copy of the content, and never updates outside of editing operations, but remains a link—this is at least self-documenting, and easier to modify later. Within the scope of a wiki system, it may be possible to avoid copying the content, and instead reference a specific version of the content.

3.5 Conclusions

This chapter has looked at how Open Hypermedia and Semantic Web relate, initially via the Dexter model, and suggested areas in which overlap may be *possible*. The following chapter now looks at how these overlaps may be *beneficial*, and may lead to a new class of system: the "open semantic hyperwiki".

Chapter 4

Open Semantic Hyperwiki

4.1 Introduction

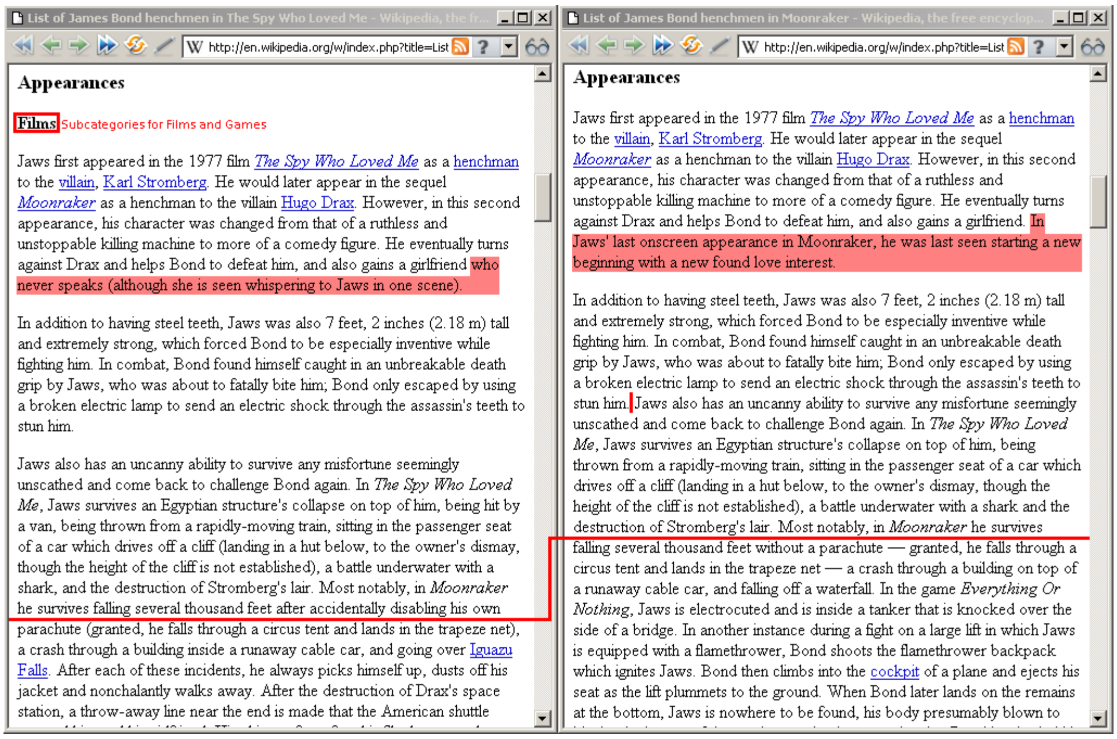
This chapter describes specific examples of how hypermedia features could be applied to semantic wikis, with an emphasis on improvements over existing practice on Wikipedia, and their semantic parallels in Semantic MediaWiki. Given this focus, consideration is given not only of model improvements, but also improvements of the user interface. The result of applying these improvements should be a semantic wiki which can also be classified as an open hypermedia system: dubbed a "open semantic hyperwiki". Not only must it demonstrate typed links such that its node structure can be mapped to a useful RDF graph, which provides for data interoperability, but first-class linking, and reasonable hypermedia capabilities.

4.2 Improved transclusion

Transclusion, as arbitrary embedding of content by reference, is a general solution to needs for both content re-use and aggregation, in addition to its originally envisaged role as a mechanism for creating, and correctly attributing the origins of, derivative works [20].

4.2.1 Content re-use

There are naturally cases where a knowledge base divided into pages for suitably granular presentation will have sections of text which are applicable to more than one page. For a specific case on Wikipedia, take the pages "List of James Bond henchmen in The


List of James Bond henchmen in The Spy Who Loved Me - Wikipedia, the fr...

http://en.wikipedia.org/w/index.php?title=List

Appearances

Films Subcategories for Films and Games

Jaws first appeared in the 1977 film *The Spy Who Loved Me* as a henchman to the villain, Karl Stromberg. He would later appear in the sequel *Moonraker* as a henchman to the villain Hugo Drax. However, in this second appearance, his character was changed from that of a ruthless and unstoppable killing machine to more of a comedy figure. He eventually turns against Drax and helps Bond to defeat him, and also gains a girlfriend who never speaks (although she is seen whispering to Jaws in one scene).

In addition to having steel teeth, Jaws was also 7 feet, 2 inches (2.18 m) tall and extremely strong, which forced Bond to be especially inventive while fighting him. In combat, Bond found himself caught in an unbreakable death grip by Jaws, who was about to fatally bite him; Bond only escaped by using a broken electric lamp to send an electric shock through the assassin's teeth to stun him.

Jaws also has an uncanny ability to survive any misfortune seemingly unscathed and come back to challenge Bond again. In *The Spy Who Loved Me*, Jaws survives an Egyptian structure's collapse on top of him, being hit by a van, being thrown from a rapidly-moving train, sitting in the passenger seat of a car which drives off a cliff (landing in a hut below, to the owner's dismay, though the height of the cliff is not established), a battle underwater with a shark, and the destruction of Stromberg's lair. Most notably, in *Moonraker* he survives falling several thousand feet after accidentally disabling his own parachute (granted, he falls through a circus tent and lands in the trapeze net), a crash through a building inside a runaway cable car, and going over Iguazu Falls. After each of these incidents, he always picks himself up, dusts off his jacket and nonchalantly walks away. After the destruction of Drax's space station, a throw-away line near the end is made that the American shuttle

List of James Bond henchmen in Moonraker - Wikipedia, the free encyclop...

http://en.wikipedia.org/w/index.php?title=List

Appearances

Jaws first appeared in the 1977 film *The Spy Who Loved Me* as a henchman to the villain, Karl Stromberg. He would later appear in the sequel *Moonraker* as a henchman to the villain Hugo Drax. However, in this second appearance, his character was changed from that of a ruthless and unstoppable killing machine to more of a comedy figure. He eventually turns against Drax and helps Bond to defeat him, and also gains a girlfriend. In Jaws' last onscreen appearance in Moonraker, he was last seen starting a new beginning with a new found love interest.

In addition to having steel teeth, Jaws was also 7 feet, 2 inches (2.18 m) tall and extremely strong, which forced Bond to be especially inventive while fighting him. In combat, Bond found himself caught in an unbreakable death grip by Jaws, who was about to fatally bite him; Bond only escaped by using a broken electric lamp to send an electric shock through the assassin's teeth to stun him. Jaws also has an uncanny ability to survive any misfortune seemingly unscathed and come back to challenge Bond again. In *The Spy Who Loved Me*, Jaws survives an Egyptian structure's collapse on top of him, being thrown from a rapidly-moving train, sitting in the passenger seat of a car which drives off a cliff (landing in a hut below, to the owner's dismay, though the height of the cliff is not established), a battle underwater with a shark and the destruction of Stromberg's lair. Most notably, in *Moonraker* he survives falling several thousand feet without a parachute — granted, he falls through a circus tent and lands in the trapeze net — a crash through a building on top of a runaway cable car, and falling off a waterfall. In the game *Everything Or Nothing*, Jaws is electrocuted and is inside a tanker that is knocked over the side of a bridge. In another instance during a fight on a large lift in which Jaws is equipped with a flamethrower, Bond shoots the flamethrower backpack which ignites Jaws. Bond then climbs into the cockpit of a plane and ejects his seat as the lift plummets to the ground. When Bond later lands on the remains at the bottom, Jaws is nowhere to be found, his body presumably blown to

FIGURE 4.1: Annotated sample of differences between Jaws texts. Note the additional hyperlink on the left below the horizontal line, and the extra prose on the right.

Spy Who Loved Me”[1], and “List of James Bond henchmen in Moonraker”[2], depicted in figure 4.1. Because the character of ‘Jaws’ appears in both of these films, and because the pages require some actual content regarding the henchmen, rather than just links to further details, both pages include information on Jaws. This is clearly redundant, and comparison of the pages and their histories shows manual effort being used to attempt to synchronise the sections.

Instead, the section on Jaws could be shared between the pages using transclusion. Either one page could simply transclude the other’s section, or a page dedicated to Jaws could host the section. It is also possible to consider this section a first-class content object, which is only intended to be viewed when transcluded into the context of another page.

While Wikipedia (MediaWiki) has a macroing system which can provide transclusion-like behaviour, it is generally not used outside of the roles its design and documentation has focused upon: simple macros to simplify some aspects of markup, and the inclusion of special, if widely-used, annotation ‘boxouts’. Only entire pages can be transcluded, thus necessitating the first-class approach. However, editing composite pages is an awkward

[1]`http://en.wikipedia.org/w/index.php?title=List_of_James_Bond_henchmen_in_The_Spy_Who_Loved_Me&oldid=155628451`

[2]`http://en.wikipedia.org/w/index.php?title=List_of_James_Bond_henchmen_in_Moonraker&oldid=154876215`

and complicated affair, where each component part must be managed in isolation. There is no navigational device presented from the transluding page to the transcluded one, requiring the user to manually open the latter for editing. When editing, transclusions merely appear as '`{{target page}}`': one cannot edit the transcluded section 'in situ', with the context of the transcluding page visible.

4.2.2 General-purpose aggregation

Transclusion provides arbitrary composition without the need for other specialised constructs; especially in combination with link endpoints which perform document search to select targets to transclude. Let us say that you wished to build a list of abstracts of papers relating to topics you are interested in, which have been recently added to the wiki, and which you have not yet read. The papers are recorded as nodes in a wiki, and their abstracts are also nodes which are transcluded in via an 'abstract' relationship, as with the example in section 3.4.7. They also have 'topic' relationships to nodes for the topics to which they relate. Nodes are related to meta-nodes by 'meta'. A meta-node for the 'Literary Machines' node describes the node in the same way that the 'Literary Machines' node describes the publication Literary Machines. For this example, these meta-nodes describe the creation date of a node via a 'creation' date literal.

A query similar to that below (in pseudo-SPARQL, omitting namespacing) could return a list of nodes to transclude to generate the required result, where `<reader>` is a query parameter resolving to the node representing the current user:

```
SELECT ?abstract {
  <reader> interestedin ?topic .
  ?paper topic ?topic ;
         abstract ?abstract ;
         meta ?meta .
  ?meta creation ?date
} ORDER BY DESC(?date) LIMIT 10
```

A more advanced version may also include patterns to ensure that '?date' is newer than the last access time from `<reader>` for the node which uses the query, although this would rely upon per-user node access time information being available; maintaining such data would not be scalable. A more computationally feasible solution might be to require that '`<reader> hasread ?paper`' is not known, although this technically breaks the open world assumption.

Semantic MediaWiki already allows, and suggests, operations of this kind via its 'inline query' mechanism, but the results of such queries are always presented as links, which is not conductive to producing an overview of the documents found. Even if it permitted MediaWiki-style transclusion, the limitation of only transcluding whole pages is problematic. While pages could be composites of a summary and their detailed content, as with the papers example above, this again introduces the content re-use editing problems.

4.2.3 Editing solution

To alleviate the problem of using MediaWiki-style transclusion for page composition and section re-use—fragmentation of editing into isolated parts—we propose a technique of 'edit-time transclusion', where transcluded text is both displayed and modifiable at the point of editing. If it is possible to edit each of the components in the context of the entire page, then the cost of requiring any transcluded section to be a first-class entity is reduced: the section continues to be editable in its original context. This ability to subdivide, or 'cut', pages into compositions of named regions, combined with 'all pages are writable' approach of wikis, effectively allows transclusion of arbitrary parts of page, by 'cutting' it at either end of the desired range. It may be desirable to make some of this subdivision implicit: if a page contains section headings, then it can be modelled as a composite of these sections, such that they provide a level of sub-page transclusion without explicit cuts.

As mentioned in section 3.4.7, 'cold' transclusion would help to reduce the maintenance burden of non-updating, 'substituted' templates, by not discarding the relation between the copied template body and its source and parameters. Because substitution effectively copies and pastes the processed template into the node, the results are editable, although modifications do not propagate back. Allowing 'hot' transclusion to also expand at edit-time increases consistency with this mechanism.

4.3 Any-anchor link editing

In MediaWiki, as with wiki systems in general, links are embedded and web-style. Even though the links exist within the domain of the wiki system, and there may be a 'back-links' feature, they are subject to web-style directionality limitations, and may only be edited as part of the node in which they are embedded. Semantic MediaWiki builds upon these links to form semantic statements: a relationship between two nodes is an embedded link in the subject node.

In some cases, it may be more suitable to embed a relationship link in the object node. Rather than triples of the form '`*node* relation target`', the form '`source relation *node*`' better suits cases where it is natural to assert that many resources relate to a given resource. For example, it is more natural to assert in a page about Perl that `awk` `inspired` and `sed inspired` it, than to spread this information across the 'awk' and 'sed' pages (namely, that they `inspired Perl`). Whichever direction is expressed, it is a natural extension of backlinks to use OWL inverse property information to display any appropriate link in the other direction with the node. However, both of these directionalities are just views upon the underlying hypertext and RDF model: a source page/subject resource, a target/object, and an arc between them.

If placing a link 'in' a node's source is viewed as merely a metaphor of the user interface, then mapping back from the hypertext to the node source will automatically indicate the presence of the link from either of its ends. This is functionally identical to the process of applying links from a linkbase in Open Hypermedia; crucially, however, it is being applied at edit-time when the hyperstructure is being flattened into a single node's source.

As a result, the user does not have to think about where a link may be stored: if they can see it, they can edit it, even if it was originally created by adding it to the source of the node at the other end. Whether editing 'Perl' or 'awk', the 'inspired' relation is visible and editable. Indeed, Brown and Brown found that the distinction of where links truly reside is not helpful: "it helps, however, if the machinations of the [link embedding/separating tools] are disguised from the user"[61].

4.3.1 First-class links

A link being edited from both ends in this fashion is potentially subject to edit conflicts. Handling this both requires that all links are first-class (such that conflicts can be identified as they are for nodes), and that an implementation can cope with the possibility that a single editing session may cause multiple conflicts (which is also a possible outcome of modifying edit-time transclusions). No known wiki system has these capabilities.

Given that the first-class object of a wiki system is the node, first-class links may therefore themselves be the endpoints of other links. While this presents a mechanism for meta-annotation, such annotations do not fit well within the RDF abstract graph model, which does not grant statements identity: this is left as application-specific knowledge. Meta-annotations could allow links to be owned by, and visible to, only particular users, allowing them to form their own, private, trails over the data.

4.4 Generic links

Early wiki systems, including the initial WikiBase system described in section 2.4.1.1, featured a crude form of generic links, where words capitalised in 'CamelCase' were links to the pages of the same name. However, this feature has been omitted from more recent designs, including MediaWiki. As a result, users must manually mark up terms which should link to articles on that topic each time that they occur. Simple generic links could automate much of this process (ideally with some of the same style constraints as practiced by users, such as only marking the first occurrence) without the former need for unnaturally-capitalised page titles. For a wiki with the wide domain of Wikipedia, local links would allow generic linking for ambiguous terms, and could be scoped by existing node classifications, such as the category system. The term ‘ontology’ used in pages in the category of ‘computing’ may have a different link target to when it is used in pages of category ‘philosophy’.

4.5 Level of detail

Often, it is desirable to accompany a link to a page with a short summary of that page’s topic. In particular, Wikipedia has many cases where articles include a summary of another article, along with a “main article” link. The ‘London’ page[3], for example, has many sections which consist primarily of summaries of more detailed pages, such as ‘Education in London’. If the main article’s summary changes—possibly because its subject changes—this change must be replicated manually out to any page which also summarises it.

Transclusion, as covered earlier in this chapter, provides a solution to re-use, but also requires that some named subset of the page exists to transclude. In this case, rather than an abstract, or other introductory component of the detailed page, what is desired is a low detail version of the entire page. Providing a standard mapping between representations at different detail levels, e.g. via a `lessDetailedVersion` relation, allows arbitrary pages to be summarised without knowledge of which sections, if any, they may have which are suitable for the task. Transclusion also allows sharing of text between detail levels.

4.6 Interoperability

Wikis are web applications, and thus are not truly interface agnostic. While the web may once have been designed as a set of open interchange formats and protocols, forming an “information bus” [2] supported by a range of client and server implementations, ‘web applications’ tend to use it as a glorified thin-client system: the actual information being transmitted is just an interface for a client which actually operates on the web server. Proper separation of interface would allow for alternate clients to be created; for example, rich clients which run entirely on the local machine.

[3]`http://en.wikipedia.org/w/index.php?title=London&oldid=155695080`

Abstraction of data sources is also an area of potential improvement. Most wikis rely on their own, specific datastore, and do not themselves access external resources (although they may generate interface-level links to them). However, one of the great perceived strengths of Web 2 and the Semantic Web is the idea of combining data from disparate sources, and it is unlikely that the manually-activated and unscoped import/export mechanisms offered by current wikis will be sufficient to realistically share information (RDF statements, therefore links) between knowledge bases. Some of the technologies developed in the field of blogging, such as 'TrackBack'[4], are of interest here, e.g. for detecting incoming links despite the unidirectional nature of the underlying Web. This is an area for future research.

4.7 Instance property editing

Wikipedia is a rich resource of potentially semantic information on some topics: for example, cities. There is a template which can be used on a page for a city which will generate a summary box containing such information as the region, population, and mayor. When users have provided this kind of structured information on Semantic MediaWiki, they have again used templates, but now with explicit predicates.[5] However, the task of the template is now not so much a matter of presentational consistency, as it is suggesting which predicates apply to instances of the class of cities. If such a class were to be appropriately described, then the system could provide better encouragement and assistance to users seeking to add or modify instances of it.

For example, let us say that the class of cities is modified to cover instances gaining the new property of settlement date. With Semantic MediaWiki, the template would have to be updated to support this, and it would be necessary for a user to know of the addition in order to provide that parameter to the template when using it. Conversely, if the system knows that instances of the class can, or must, have a settlement date,[6] then users editing the instances of cities—for whatever reason—can be prompted to provide this missing information. Because the system is dealing directly with predicates, rather than being indirected through a text-substitution template, it can potentially display any missing data as form fields, and use type information to provide appropriate rich controls: for example, if the range of 'settlement date' is a date, it can provide a calendar.

The template's role can then be relegated to being *purely* presentational: the semantics are present because the system knows that it should encourage users to provide them for this class. All the template is required to do is access them by inline query, and display them in a suitable box.

[4]`http://www.movabletype.org/docs/mttrackback.html`

[5]`http://ontoworld.org/wiki/Help:Semantic_templates`

[6]The simplest expression of this is probably using OWL cardinality constraints.

While not a research project, the 'Semantic Forms' extension to Semantic MediaWiki[7] covers similar ground. It allows forms to be defined for classes of resource which can be used to edit the properties of instances. These forms, however, are not declared in terms of the properties applicable to a class, but instead based on special template syntax.

4.8 Conclusions

There is a general ideal around these specific improvements: that semantic and hypermedia features should be useful not only once the information exists, but *during the process of creating that information*. Ideas such as edit-time transclusion, and edit-time link application, help reduce the cognitive overhead of using such features by only concerning the user with the final, presented model of nodes with information; not the implementation details of the hypertext/RDF graph structure.

The following chapter confirms that there is scope for worthwhile efficiency gains by experimentally studying large-scale Wikipedia editing behaviour. The hypothesis that substantial effort is being expended on hyperstructure editing is tested: if a significant proportion of edits are of a hypertext nature, then enriching the hypertext nature of wikis should prove beneficial.

[7]`http://www.mediawiki.org/wiki/Extension:Semantic_Forms`

Chapter 5

Experiment I: Macro-scale

5.1 Introduction

This chapter presents an experiment carried out to test the hypothesis that potentially-automated hyperstructure modification formed a substantial degree of total editing effort on large-scale wiki systems. Given this, it therefore follows that improvements to the hypermedia featureset available have great potential to improve the utility of these resources—a significant component of their existence is their interlinked nature—and to reduce the workload required to author such.

This work has been published in the proceedings of SWKM 2008 [67].

5.2 Hypothesis

We carried out an experiment to estimate the proportion of effort expended maintaining the infrastructure around data, rather than the data itself, on a weak hypertext wiki system. We define a 'weak' hypertext system here as one whose feature set is limited to embedded, unidirectional, binary links, as with the World Wide Web. Our hypothesis is that the manual editing of link structure, of a type which richer hypertext features could automate, will show to be a significant overhead versus changes to the text content.

This experiment also seeks to partially recreate a related, informal experiment, discussed in an essay by Swartz [68].

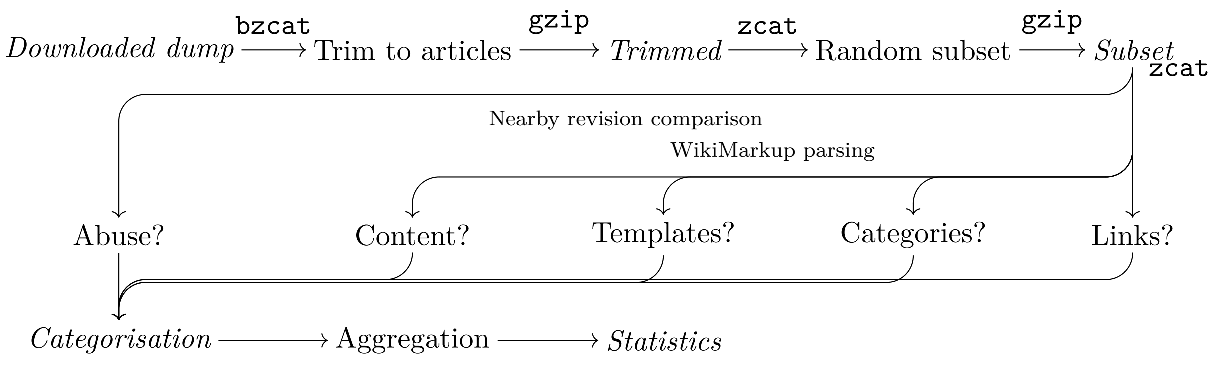


FIGURE 5.1: Data flow of Wikipedia experiment

5.3 Dataset

We chose English Wikipedia[1] as the experimental dataset, because it has both a considerably large and varied set of documents, and a complete history of the editing processes—performed by a wide range of Web users—between their first and current versions[2]. The wiki community keep the dataset fairly well inter-linked and categorised for cross-reference, but they do this via the cumulative efforts of a large body of part-time editors. At the time of writing, Wikipedia contains 3.5 million self-described "content pages" and lists 14 million registered users over its lifespan.[3] In comparison, popular[4] wiki hosting site Wikia counts 4 million content pages and 2 million registered users, but these figures are split over 165 thousand hosted wikis, the general theme of which are domain-specific encyclopedic content.[5] For example, the featured 'Memory Alpha' Wikia-hosted wiki (33 thousand content pages, 370 thousand registered users) is an in-depth encyclopedia for the fictional Star Trek universe. As well as being statistically significant, demonstrating possible improvement of English Wikipedia is socially significant, as it is a widely-used and active resource.

It is important to stress the size of the English Wikipedia dataset. Wikipedia make available 'dumps' of their database in an ad-hoc XML format; because this study is interested in the progression of page contents across revisions, it was necessary to use the largest of these dumps, containing both page full-text and history (unfortunately, also non-encyclopædic pages, such as discussions and user pages). This dump is provided compressed using the highly space-efficient (although time-complex) bzip2 algorithm; even then, it is 84.6GB. The total size of the XML file is estimated to be in the region of two terabytes.

[1]`http://en.wikipedia.org/`
[2]MediaWiki, unlike many wikis, never deletes old revisions of a page.
[3]`http://en.wikipedia.org/wiki/Special:Statistics`
[4]Wikia's Alexa rank is 201 at the time of writing.
[5]Datasheet available at `http://www.wikia.com/Press`

5.4 Procedure

Figure 5.1 shows the simplified data flow of the processing of the dump performed for the experiment.

First, we trimmed down the dataset to just those pages which are encyclopædic articles, as these are the pages of greatest significance to the Wikipedia project's goals, and thus the most important to study. Otherwise, the dataset would include a lot of 'noise' in the form of discussion and user pages, which are likely to have different editing patterns, and be less connected to the hyperstructure. The most practical way to do this was to remove any page placed in a namespace. On English Wikipedia, this also has the effect of removing other page types, such as media and image descriptions, help pages copied from MetaWiki, front-page portal components, and templates. As this stage also required decompressing the data, it ran over the course of several days on a multi-processor server.

We took a random subset of the data for processing. Samples of 0.04% and 0.01% of pages (approximately: see the description of the subset tool below; actual page counts 14,215 and 3,589 respectively) were selected, yielding a compressed dataset which would fit on a CD-ROM, and could be processed in a reasonable timeframe. Further iterations of the experiment may study larger subsets of the data.

We performed categorisation on the revisions, into several edit types which would be automatically distinguished. In particular, a simple equality comparison between a revision, and the revision two edits previous, can detect the most common (anti-)abuse modification: the rollback, or revert (unfortunately, MediaWiki does not record such operations semantically). A sequence of reverts[6] is usually indicative of an 'edit war', where two users continually undo each-others changes in favour of their own. Page blanking was also easy to detect, but identifying more complicated forms of vandalism (e.g. misinformation, spam) was not feasible—if reliable, automatic detection were possible, they would not be present in the data, as Wikipedia could prevent such changes from being applied. Identifying abuse (and abuse management) of the simpler types is important, as otherwise they would appear as very large changes.

In order to detect changes in the text content, templates used, MediaWiki categories, and links from a page, it was necessary to attempt to parse the MediaWiki markup format. Such 'wikitext', as it is known, is not a formally defined language: there is no grammar for it, and it does not appear likely that an unambiguous grammar actually exists. MediaWiki does not have a parser in the same way as processing tools such as compilers and XML libraries; instead it just has a long and complicated set of text substitution procedures which convert parts of 'wikitext' into display-oriented HTML. These substitutions often interact in a ill-defined manner, generally resulting in either

[6]e.g. `http://en.wikipedia.org/w/index.php?title=Anarchism&diff=next&oldid=320139`

more special-case substitutions, or as being defined as a new, hybrid, feature, which editors then use. Because of these problems, and the lack of abstraction in MediaWiki's 'parser', as much as the programming language boundary, a 'scraping' parser was created which attempted to approximate partial processing of the wikitext format and return *mostly* correct results. This parser is a single-pass state machine (42 states) with a few additional side-effects. This yields excellent performance: testing showed that the time spent parsing is dominated by the time performing decompression.

To determine if an edit included a significant ('major') change to the text content, we required a difference metric between the plaintext of the revisions. This metric was then compared to a threshold to classify edits as being content changes or not (in particular, the imperfect parser generates 'noise' from some non-content changes, as it cannot correctly remove all the markup). The default threshold was chosen as 5%: sentences in the English language are generally around twenty words in length, so this considers anything up to changing one word in each sentence as non-major (minor). MediaWiki also allows registered users to explicitly state than an edit is minor; this flag was respected where present.

We chose an approximation of Levenshtein distance[69], as it is a simple measure of insertions, deletions, and substitutions, fitting the kind of edit operations performed on the wiki. However, the algorithm for computing Levenshtein itself was far too time-complex, even with aggressive optimisation, taking two minutes on a tiny test set of just a few thousand revisions of a single page (before trimming away the identical parts at either end of both strings to take advantage of edit locality, this took 45 minutes). The problem was that the matrix-based approach is $O(n \times m)$, where n and m are the string lengths, in all cases: for n and m in the region of 16,000 characters, as found on many revisions, merely iterating through all 256 million matrix cells was prohibitively expensive.

Instead, we developed a new approach to computing such a distance, taking advantage of the domain-specific knowledge that the two strings being compared are likely very similar save for 'local' edits: the difference is likely to be a new paragraph, or a removed sentence, or some changed punctuation. Instead of efficient search within the space of editing operations, as Levenshtein, it is based on the idea of "sliding windows": a pass is made over both strings in parallel; when characters begin to differ, a look-back 'window' is opened between the point at which differences began, and continues until similarity is again found between these windows. At this point, the position through the strings resynchronises, the distance is increased by the offset required, and the windows are again 'closed'. When the end of either string is reached by the far edge of the window, the algorithm can terminate, as any remaining characters in the other string must be unmatched and thus add to the distance. As a result, the algorithm scales with regard to the shorter of the two strings, which is helpful when revisions may add whole paragraphs of new text to the end. To reduce inaccuracy in certain cases, the algorithm

Algorithm 1 ‘Sliding window’ string distance metric

```
function STRING-DISTANCE($A, B$)
    $proc \leftarrow 0$                                    ▷ No. of chars. of string processed
    $procstr \leftarrow$ NEITHER                           ▷ Last string aligned upon
    $dist \leftarrow 0$                                    ▷ Difference accumulator
    $nearA \leftarrow farA \leftarrow A$                   ▷ Near and far pointers
    $nearB \leftarrow farB \leftarrow B$
    Let $endA$ be the beyond-last character of buffer $A$, and $endB$ beyond $B$
    function SCAN($near, far$)
        for $scan \leftarrow near$ to before $far$ do
            if Chars. at $scan$ and $far$ same then return $scan$
        return false
    while $farA \neq endA \wedge farB \neq endB$ do
        $synfarA \leftarrow$ SCAN($nearA, farA$)
        $synfarB \leftarrow$ SCAN($nearB, farB$)
        if $synfarA \vee synfarB$ then                    ▷ Missed alignment
            if $synfarA$ is further into $A$ than $synfarB$ is into $B$ then
                $farA \leftarrow synfarA$
            else
                $farB \leftarrow synfarB$
        else if $synfarA$ then $farA \leftarrow synfarA$
        else if $synfarB$ then $farB \leftarrow synfarB$
        if Chars. at $farA$ and $farB$ same then
            $enA \leftarrow$ MIN($nearA, A + proc - 1$) ▷ Aligned; calc. nears after proc. point
            $enB \leftarrow$ MIN($nearB, B + proc - 1$)
            $unA$ = positive dist. from $enA$ to $farA$   ▷ Unaligned lengths
            $unB$ = positive dist. from $enB$ to $farB$
            function ALIGN($un, far, buffer, other$)
                $distance \leftarrow distance + un$
                $proc = far$'s distance into $buffer$
                if $procstr = other$ then $proc \leftarrow proc + 1$
                $procstr \leftarrow buffer$
            if $unA > unB$ then
                ALIGN($unA, farA, A, B$)
            else
                ALIGN($unB, farB, B, A$)
            if $farA = endA$ then                          ▷ Ending
                $distance \leftarrow distance+$ distance between $farB$ and $endB$
            else if $farA = endA$ then
                $distance \leftarrow distance+$ distance between $farA$ and $endA$
            else                                          ▷ Advanced with closed window
                $nearA \leftarrow farA \leftarrow farA + 1$
                $nearB \leftarrow farB \leftarrow farB + 1$
                $proc \leftarrow proc + 1$
        else                                              ▷ Not aligned; widen windows
            if $farA \neq endA$ then $farA \leftarrow farA + 1$
            if $farB \neq endB$ then $farB \leftarrow farB + 1$
    return $dist$
```

maintains a 'processed point' cursor, to avoid double-counting of overlapping insertions and deletions. Pseudocode is presented as algorithm 1, which works on a pair of string buffers, and `upstr.c` in the tool source contains a C implementation. This approach is still $O(n \times m)$ worst-case, but is $O(n)$ (where n is the shorter string) for identical strings, and degrades smoothly as *contiguous* differences increase in size: instead of two minutes, the tiny test set was compared in a little over ten seconds.

Unfortunately, changes such as 'ABCF' to 'ADCDBCF' can return overestimates, as the localisation which explicitly prevents full lookback (and keeps computational cost below $O(n^2)$) causes the 'C' in 'BCF' to match with the 'C' in 'DCD': 'ADC' is considered a substitution of 'ABC' before the algorithm can realise that 'BC' is still intact in the string, and 'DCD' is merely an insertion. As a result, the later 'B' is considered an insertion, as it no longer matches anything, and the distance is overestimated by one. Synthetic tests showed this overestimation to be minor; tests against Levenshtein on a tiny subset of Wikipedia data (a node's first few hundred revisions, thus under heavy editing) show it to be larger, with errors in the tens, and a peak error of over two-hundred. The reason for such large errors is unclear, as the resynchronisation approach should also keep *error* localised, but it does not greatly affect the result for the purpose of minor/major determination: the majority of changes were correctly classified.

Identifying changes to links, etc. was significantly simpler, and merely required comparing the sets of links identified by the parser across two revisions. These categorisations yielded simple information on which kinds of changes were made by each revision, and removed much of the 'bulk' of the dataset (the revision texts); as a result, simple scripts could then handle the data to aggregate it into various groupings in memory, so as to produce graph data and statistics for analysis. Gnuplot[7] was used to plot the graph data into graphics as part of the build process for this thesis.

We identified the following non-mutually-exclusive groupings to usefully categorise edits:

Revert Edit which simply undoes a previous edit.

Content Major (nontrivial) edit of the page content.

Minor Minor (trivial) edit of the page content.

Category Edit to the categories of a page.

List of Edit to a page which is an index to other pages.

Indexing Edit to categories or listings, possibly both.

Template Edit to the templates used by a page.

Page link Edit to an internal page link.

[7]`http://www.gnuplot.info/`

URL link Edit to a WWW URL link; usually external.

Links Edit to page or URL links.

Link only As 'links', but excluding major edits.

Hyperstructure Any hypermedia change: indexing, linking, or template.

We expand upon the definition and significance of these groups as needed in section 5.6.

5.5 Tools developed

To process the sizable dataset, we created a set of small, robust, stream-based tools in C. Stream-based processing was a necessity, as manipulating the entire data in memory at once was simply infeasible; instead, the tools are intended to be combined arbitrarily using pipes. We used standard compression tools to de- and re-compress the data for storage on disk, else the verbosity of the XML format caused processing to be heavily I/O-bound.[8] The open source Libxml2[9] library was used to parse and regenerate the XML via its SAX interface. A selection of the more notable tools:

dumptitles Converts a MediaWiki XML dump (henceforth, "MWXML") into a plain, newline-separated, list of page titles. Useful for diagnostics, e.g. confirming that the random subset contains an appropriate range of pages.

discardnonart Reads in MWXML, and outputs MWXML sans any pages which are in a namespace; pedantically, due to the poor semantics of MWXML, those with colons in the title. This implements the "trim to articles" step of figure 5.1.

randomsubset Reads and writes MWXML, preserving a random subset of the input pages. In order for this to be O(1) in memory consumption, this does not strictly provide a given proportion of the input; instead, the control is the probability of including a given page in the output. As a result, asking for 50% of the input *may* actually yield anywhere between none and all of the pages: it is just far more likely that the output will be around 50% of the input.[10]

categorise Reads MWXML and categorises the revisions, outputting results to a simple XML format.

cataggr A Perl script which processes the categorisation XML to produce final statistical results and graph data. By this point, the data are small enough that a SAX parser is used to build a custom in-memory document tree, such that manipulation is easier.

[8]Specifically, GNU Zip for intermediate; bzip2, as originally used by Wikipedia, made processing heavily CPU-bound.

[9]`http://xmlsoft.org/`

Edit type	Proportion
Categories	8.71%
Lists	1.97%
Overhead	10.56%

(a) 0.01% subset

Edit type	Proportion
Categories	8.75%
Lists	3.72%
Overhead	12.34%

(b) 0.04% subset

Table 5.1: Proportions of edits related to index management

The tools are available under the open source MIT license, and can be retrieved from `http://users.ecs.soton.ac.uk/prb/phd/wikipedia/` to recreate the experiment.

5.6 Results

Because of the known error margin of the approximation of Levenshtein distance, we computed results from both genuine and approximated distances on the 0.01% subset, so as to discover and illustrate the effects of approximation; the computational cost difference between the algorithms was significant: two-and-a-half hours for genuine, eight minutes for approximated. Results were then generated from the more statistically significant 0.04% subset (27 hours). This latter subset contained some pages on contentious topics, which had seen large numbers of revisions as a result.

5.6.1 Index management

Table 5.1 shows the proportions of edits in categories pertaining to index management. "Categories" are changes to the categories in which a page was placed. "Lists" are any change to any 'List of' page; these pages serve as manually-maintained indices to other pages. "Overhead" are changes which fall into either of these categories: because they are not mutually exclusive (lists may be categorised), it is not a sum of the other two values. Because these metrics do not consider the change in 'content' magnitude of a change, they are unaffected by the choice of string distance algorithm.

The ten percent overhead shows a strong case for the need for stronger semantics and querying on Wikipedia; this is one of the key goals, and expected benefits, of the Semantic MediaWiki project. While virtually every 'list of' node could be replaced with a query on appropriate attributes, the gain in category efficiency is harder to measure. Any semantic wiki must still be provided with categorisation metadata such that the type of pages can be used to answer such queries. However, some improvement is to

[10]A better algorithm, which is O(1) with regards to total data size, but O(n) with regards to subset size, is to store a buffer of up to n pages, and probabilistically replace them with different pages as they are encountered. However, even this would be prohibitively memory intensive on statistically significant subset sizes, as each page may have thousands of revisions, each with thousands of bytes of text, all of which must be copied into the buffer.

Edit type	Proportion
Links	49.60%
Links only	35.53%
Hyperstructure	61.65%
Content	17.81%
Edit type	**Ratio over content**
Links	2.79
Links only	2.00
Hyperstructure	3.46

(a) 0.01% subset, Levenshtein

Edit type	Proportion
Links	49.60%
Links only	23.36%
Hyperstructure	61.65%
Content	35.60%
Edit type	**Ratio over content**
Links	1.39
Links only	0.71
Hyperstructure	1.73

(b) 0.01% subset, Approximated

Edit type	Proportion
Links	49.56%
Links only	25.24%
Hyperstructure	61.90%
Content	35.99%
Edit type	**Ratio over content**
Links	1.38
Links only	0.70
Hyperstructure	1.72

(c) 0.04% subset, Approximated

TABLE 5.2: Proportions of edits related to link management

be expected, as there are current Wikipedia categories which could be inferred: either because they are a union of other categories (e.g. ‘Free software’ and ‘Operating systems’ cover the existing category ‘Free software operating systems’) or because they are implied by a more specialised category, and no longer need to be explicitly applied to a page.

The increase in list overhead seen in the larger subset is likely a result of having a more representative proportion of ‘List of’ pages. Otherwise, the results are largely consistent across sample sizes.

5.6.2 Link management

Table 5.2 shows categories related to link management. “Links” refers to edits which changed either page-to-page or page-to-URL links. “Links only” refers to such edits *excluding* those edits which also constituted a ‘major’ content change: they are edits concerned only with links and other structure. “Hyperstructure” is the category of edits which changed any of the navigational capabilities of the wiki: either categories, ‘List of’ pages, links, or templates. “Content” is simply the category of ‘major’ edits.

The overestimating effect of the approximate string distance algorithm can be seen as a greater proportion of edits being considered ‘major’, with a knock-on effect on reducing

Category	Registered	Unregistered	Total
List of	1,146	453	1,599
Revert	4,069	679	4,748
Category	6,121	954	7,075
URL link	5,548	2,977	8,525
Indexing	7,174	1,397	8,571
Template	7,992	1,330	9,322
Content	10,275	4,182	14,457
Minor	13,776	9,961	23,737
Link only	20,969	7,877	28,846
Page link	27,205	8,871	36,076
Links	29,671	10,606	40,277
Hyperstructure	38,358	11,701	50,059
Total	57,463	23,733	81,196

Table 5.3: Categorisation of edits for 0.01% subset, Levenshtein

the ratios of edits over content edits. However, the results are consistent between the 0.01% subset with the approximated string distance, and the sample set four times the size. As a result, it would appear that the smaller size of the sample set has not introduced significant error in this case, and it is reasonable to assume that a Levenshtein distance comparison of the larger dataset would yield similar results to the 0.01% subset. Therefore, further discussion will focus on the 0.01% subset with Levenshtein distance results.

These figures show the significance of hyperstructure to Wikipedia, to a surprising degree. While we expected that link editing would prove a substantial proportion of edits compared to content, we did not anticipate that *twice as many edits change links alone than those that change content.* Most link changes were page links—those to other pages on the wiki, or metawiki—as opposed to URL links to arbitrary webpages (in some cases, pages on the wiki with special arguments). 36,076 edits modified the former, but only 8,525 the latter.

With such a proportion of editing effort being expended on modifying links on Wikipedia, there is a clear need to improve this process. Introducing richer hypermedia features to wikis, such as generic links, should prove one possible improvement. A generic link can specify that a page's title should link to that page, rather than requiring users to manually annotate it: some early wiki systems offered this capability, but only for page titles which were written in the unnatural 'CamelCase' capitalisation. Advanced examples such as local links, present in Microcosm [22, 23], can specify scope limits on the matching. This would help with ambiguous terms on Wikipedia, such as 'Interval', which should be linked to a specific meaning, such as 'Interval (music)'.

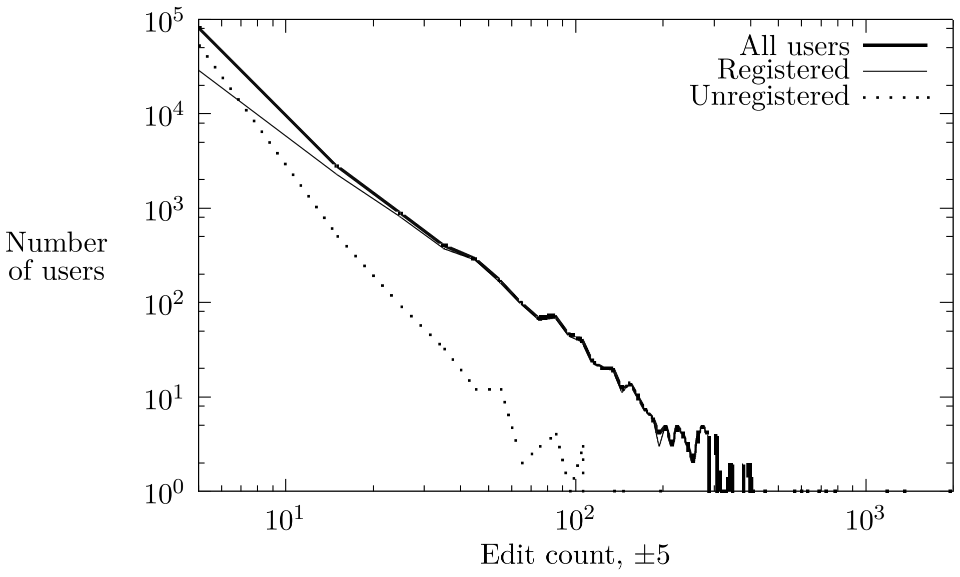


Figure 5.2: User distribution over total number of edits made; 0.04% subset

5.6.3 Overall editing distribution

Table 5.3 shows the categorisation of all edits in the 0.01% dataset, using Levenshtein for string distance, for registered and unregistered users. Note that the edit categories are not mutually exclusive, thus will not sum to the total number of edits by that class of user. "Minor" is the category of edits which did not appear to change anything substantial: either the information extracted from the markup remains the same, and the plaintext very similar; or a registered user annotated the edit as minor. Notably, over 5% of edits are reverts: edits completely rolling back the previous edit; this implies that a further 5% of edits are being reverted (presumably as they are deemed unsuitable).[11] A substantial amount of effort is being expended merely keeping Wikipedia 'stationary'.

Figure 5.2 demonstrates the distribution of users over the total number of edits they have made, in the vein of the Swartz study [68]. There is a sharp falloff of number of users as the number of edits increases (note the logarithmic scale on both axes): by far, most users only ever make very few edits, whether registered or not. Unsurprisingly, registered users tend to make more edits overall, and unregistered users are dominant at the scale of fewer than ten edits.

Figure 5.3 breaks the low-edit end of this distribution down by basic categories. It is interesting to note that, other than being in close proximity (e.g. "content" and "page link"), the lines do not have any definitive overlaps: the breakdown of edits is consistent regardless of the number of edits the user has made. Users who have made 70 edits have

[11]Actual figures may vary in either direction: this does not detect rollbacks to versions earlier than the immediately preceding version, and 'edit wars' of consecutive rollbacks *will* be entirely included in the first 5%, not belonging in the latter.

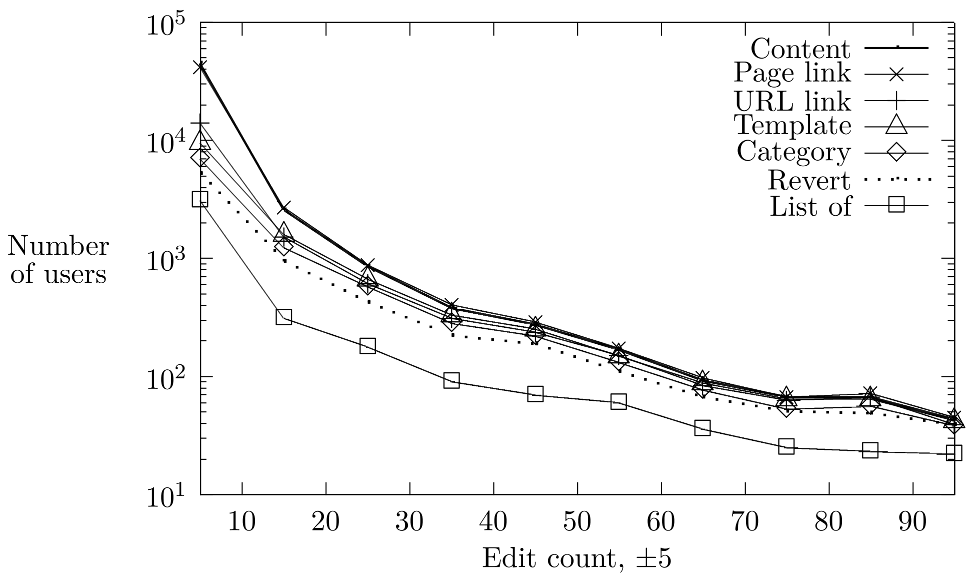


FIGURE 5.3: User distribution over total number of edits made, by category; 0.04% subset

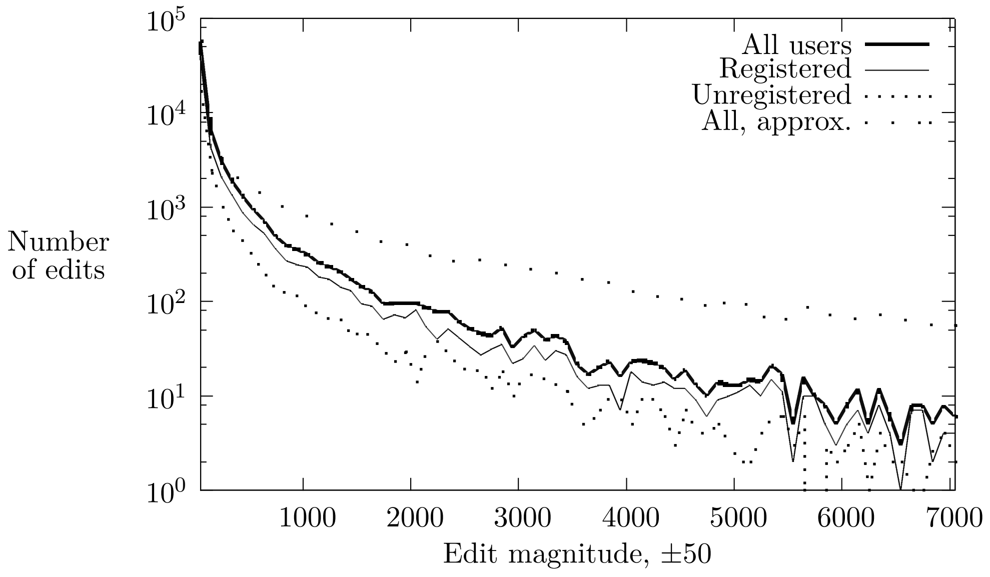


FIGURE 5.4: Edit distribution over magnitude of edit; 0.01% subset

made edits in the same relative proportions (i.e., more “revert” than “list of”) as those who have only made five.

Figure 5.4 shows how the magnitude of edits breaks down by the number of edits of that magnitude, again in the vein of Swartz [68]. Because this is clearly sensitive to the string distancing algorithm, the 0.01% subset was used, with a focus on Levenshtein: the approximate distance for all users is shown as a sparsely dotted line with a consistent overestimate. These results are largely unsurprising: registered users make larger edits, and most edits are small, with the count rapidly falling off as magnitude increases.

5.6.4 Limitations of detection

There are, unfortunately, several kinds of ‘overhead’ costs which simply cannot be detected in a computationally feasible manner by this approach. For example, use of the template substitution mechanism discussed in section 3.4.7 is not semantically recorded by MediaWiki, and is thus largely indistinguishable from the addition of a paragraph of wikitext. As a result, it is not then possible to evaluate the cost of maintaining or documenting these substitutions once the link to the original template has been lost.

It is also not computationally feasible to detect the pattern of a user performing the same fix on multiple pages, which would identify the cost of inadequate, or underused, transclusion. This would detect the “main article” problem described in section 4.5. The ‘London’ page[12], for example, has many sections which consist primarily of summaries of more detailed pages, such as ‘Education in London’. However, without some form of transclusion or composition to share text, if the main article’s summary changes—possibly because its subject changes—this change must be replicated manually out to any page which also summarises it. A transclusion mechanism would allow a single summary of the subject to be shared by all pages which reference it, including the main article on the subject, if desired.

For example, the ‘Education in London’ page may begin with a summary of its topic, highlighting the most notable institutions and successful research areas. The article on ‘London’ may then, within its ‘Education’ section, transclude this summary from the ‘Education in London’ page. Should the summary be updated, perhaps because a University gains significant notability in a new research area, this change would be automatically reflected in the ‘London’ page, as it is using the same text.

While MediaWiki’s templates do function as transclusion, they are not employed for this role: common usage and development effort focus on their use as preprocessing macros.

[12]`http://en.wikipedia.org/w/index.php?title=London&oldid=155695080`

5.7 Conclusions

The experiment consisted of the non-exclusive classification of edits made throughout the history of Wikipedia, a large and public wiki system. Classifications included both the areas of "text editing" (assumed to primarily be maintaining the *information content* of Wikipedia: its encyclopædic articles), and "link editing" (maintaining the *navigational structure* of the content). The hypothesis, that link editing formed a substantial proportion of total editing effort, which may potentially be automated, was supported by the results. Twice as many edits changed links alone, not affecting the article text. Edits which maintained manual indexes of pages constituted approximately a tenth of total edits.

The next chapter follows up with a more detailed, small-scale experiment, to better understand the patterns of real-world wiki editing.

Chapter 6

Experiment II: Micro-scale

6.1 Introduction

We have shown, in chapter 5, that scope exists for hypermedia-based improvements to wiki editing, as suggested in chapter 4. To better understand the relative usefulness of these improvements, formal study must be made of current editing practices on large-scale wiki systems. This is a form of knowledge elicitation task, and thus has no particular hypothesis to test. However, the domain of possible actions, and the steps entailed in performing them, are already known as aspects of the software.

The objective of this experiment is to identify the mental processes behind wiki editing: information on the tasks editors set themselves, and how their actions are used to achieve them. This will then be used to prioritise efforts to develop hypermedia features to assist with these tasks.

This work has been published in combination with the previous experiment as a chapter of [70].

6.2 Procedure

The experiment consisted of two main parts: a week of data collection while the participant used Wikipedia, or a functionally similar system, normally, followed by a meeting of less than an hour, covering a pair of protocol analysis sessions [71, 72]. A small questionnaire preceded the week of collection to record the prior experience of the subject, as well as obtaining informed consent to participate.

The first protocol analysis was an off-line review using logged editing information from Wikipedia. Off-line study is necessary in order to work with real-world problems in a real-world environment: the reduced accuracy of participant recall for the reasoning behind decisions and actions is balanced against the validity of those actions. Wikipedia helps provide partial compensation here by encouraging the participant to record a short motivation for any action, which may prompt their memory.

The second protocol analysis supplemented this with an on-line self-report session in a high-fidelity simulated environment (another MediaWiki install with a tiny sample of Wikipedia's content), and a set of synthetic problems, presented in a randomised order. This then trades validity of the actions for the benefits of immediate, more accurate, feedback regarding the participant's thought processes. The investigator is also present at the time of the decision to observe any other details of the process of which the participant does not make explicit note.

Information was retrieved from the Wikipedia database about the participant's editing within the span of the study: which pages were edited, and how the source text changed. This information was publicly available as part of normal Wikipedia activity. It was not, however, directly analysed: instead, it provided material for the off-line review. The data collected were anonymised transcripts of audio recordings from this, and the on-line self-report, for verbal protocol analysis. Both off-line and on-line review were limited to a maximum of a half-hour session to keep the total meeting time under an hour and avoid fatigue.

Participants were taken from geographically close wiki editors, as a practical limitation of the in-person nature of the data collection. We sought people who already had some experience with wikis, so as to capture the editing process, rather than the initial user interface experiences of a beginner. While this limited the set of potential candidates, the method of analysis is not statistical, and can work at small sample sizes [73].

The questionnaire and synthetic task scripts are presented in full in appendix A. The tasks were designed around the knowledge elicitation goal, to attempt to capture the user's reasoning, and also to solicit their opinions on the perceived effort required for each task. The range of tasks were also designed to exercise areas potentially subject to improvement through richer hypertext functionality.

Edit description of same villain in two movies. Discovers how the user handles having to update a section of text which is used by two articles. The history of the node shows rough synchronisation by manual copying, but is desynchronised at the point captured for the synthetic environment. Transclusion could be used to share this text.

Add fact to specific article, and to summary in general article. A similar case to the above, but this time with a level-of-detail angle. Because the latter is summary of the former article, there is less scope for text to be shared outright. The domain for this task is the London Underground. Adaptation and transclusion could form a solution.

Refine links pointing to the disambiguation node for 'shelf'. Requires the ability to traverse (or edit, but MediaWiki does not support this) links in the reverse direction. The ability to edit links from any of their endpoints, which is facilitated by first-class links, could help with this.

Create page summarising two other articles about type of train. Requires aggregation of summaries, which are suggested to be taken from the pages' introductory overview. This again touches on the issues of synchronising shared content, and also the task of synchronising which trains are on the list with which trains have articles to be summarised. Transclusion of query-endpoint fat links could achieve this.

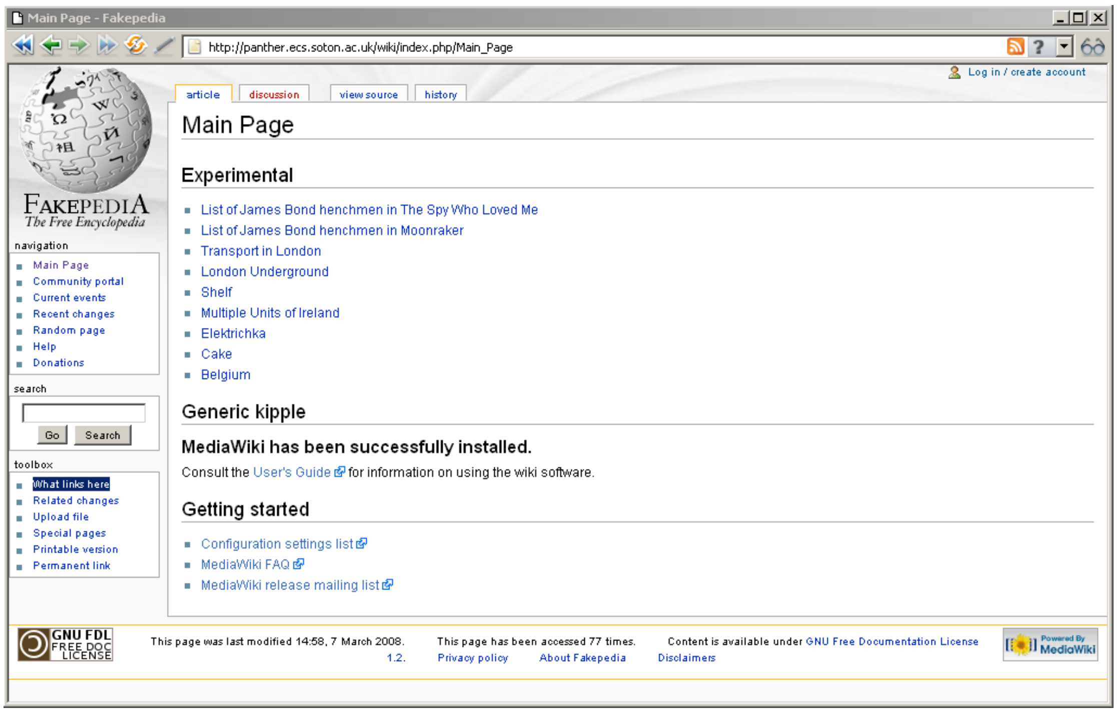
Add links where appropriate to plain text of 'cake' article. Although the task is synthetic, this attempts to capture the reasoning behind which words in an article are hyperlinked. This then informs where use of generic links may, or may not, be appropriate.

Add fact to infobox of 'Belgium' article. Tests resource property editing, which is hidden within template code. Provides information on how users approach what is abstractly a very simple operation. Richer support for semantics, such as forms to edit known class properties, could improve this.

All of the tasks were created from content taken from Wikipedia, for authenticity of the simulated environment. Minimal errors and omissions were introduced where necessary to set up the required problem. The ordering of the tasks was randomised for each participant (and preserved in these results) and, time permitting, they were prompted to perform all of them.

Figure 6.1 shows the front page of the simulated environment for the on-line section of the experiment. Note the small "What links here" link in the toolbox, which has been highlighted for this screenshot. Following it is how one can reveal backlinks for the current page as a list in Mediawiki.

Figure 6.2 provides a glimpse at the complexity of the Belgium infobox task. The top-left window shows part of the infobox as displayed on the page; the complete box is several full screens high. The top-right window shows the source as seen when editing the page the infobox is used on. Note that in this (real-world) case the ordering of the elements in the source does not match the ordering of their display, as they are arguments to a template which will then substitute them into a display form. The bottom-left window is also a view of editing, showing the area below the page source, where Mediawiki displays a list of templates used by the page. For this article there are twenty-six items in this list. The bottom-right window shows part of the result of following one of these links: the template page, which includes some documentation. This documentation is managed manually and in this (again, real-world) case is incomplete; we show the part


Main Page - Fakepedia

http://panther.ecs.soton.ac.uk/wiki/index.php/Main_Page

Log in / create account

article | discussion | view source | history

FAKEPEDIA
The Free Encyclopedia

navigation
- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

search

Go | Search

toolbox
- What links here
- Related changes
- Upload file
- Special pages
- Printable version
- Permanent link

Main Page

Experimental

- List of James Bond henchmen in The Spy Who Loved Me
- List of James Bond henchmen in Moonraker
- Transport in London
- London Underground
- Shelf
- Multiple Units of Ireland
- Elektrichka
- Cake
- Belgium

Generic kipple

MediaWiki has been successfully installed.

Consult the User's Guide for information on using the wiki software.

Getting started

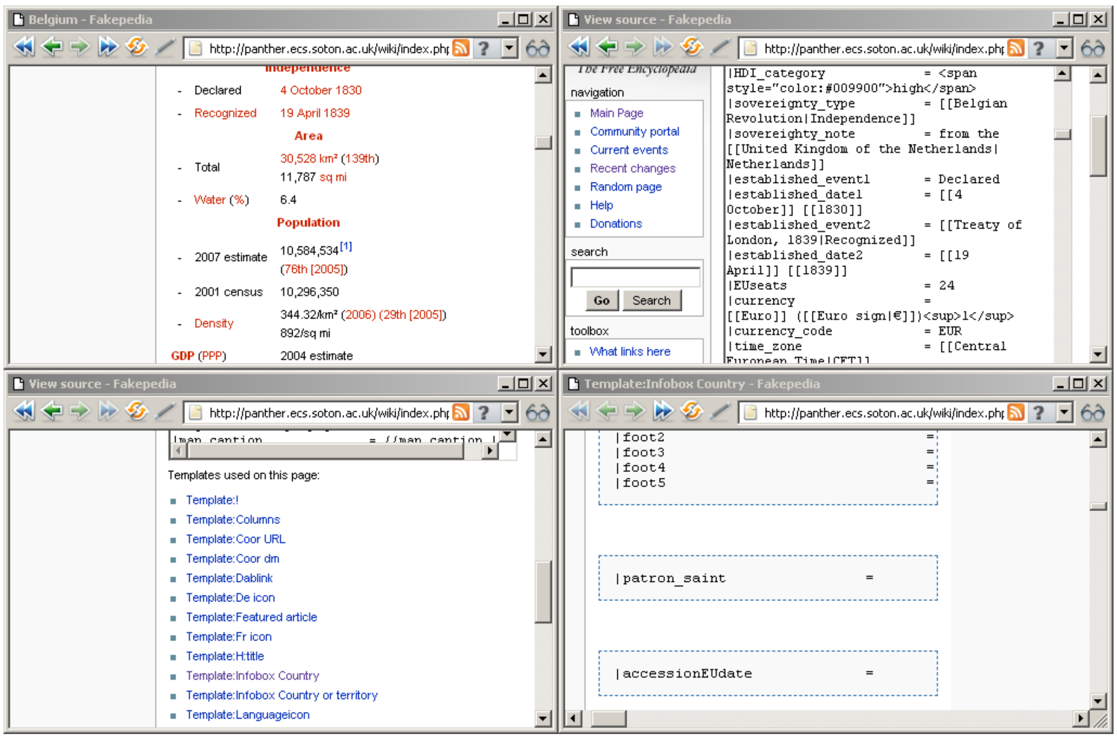
- Configuration settings list
- MediaWiki FAQ
- MediaWiki release mailing list

GNU FDL FREE DOC LICENSE

This page was last modified 14:58, 7 March 2008. This page has been accessed 77 times. Content is available under GNU Free Documentation License 1.2. Privacy policy About Fakepedia Disclaimers

Powered By MediaWiki

Figure 6.1: Fakepedia: a hi-fidelity Wikipedia simulation


Belgium - Fakepedia

http://panther.ecs.soton.ac.uk/wiki/index.php

Independence

-	Declared	4 October 1830
-	Recognized	19 April 1839
		Area
-	Total	30,528 km² (139th) 11,787 sq mi
-	Water (%)	6.4
		Population
-	2007 estimate	10,584,534[1] (76th [2005])
-	2001 census	10,296,350
-	Density	344.32/km² (2006) (29th [2005]) 892/sq mi
GDP (PPP)		2004 estimate

View source - Fakepedia

http://panther.ecs.soton.ac.uk/wiki/index.php

The Free Encyclopedia

navigation
- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help
- Donations

search

Go | Search

toolbox
- What links here

```
|HDI_category                = <span
style="color:#009900">high</span>
|sovereignty_type            = [[Belgian
Revolution|Independence]]
|sovereighty_note            = from the
[[United Kingdom of the Netherlands|
Netherlands]]
|established_event1          = Declared
|established_date1           = [[4
October]] [[1830]]
|established_event2          = [[Treaty of
London, 1839|Recognized]]
|established_date2           = [[19
April]] [[1839]]
|EUseats                     = 24
|currency                    =
[[Euro]] ([[Euro sign|€]])<sup>1</sup>
|currency_code               = EUR
|time_zone                   = [[Central
European Time|CET]]
```

View source - Fakepedia

http://panther.ecs.soton.ac.uk/wiki/index.php

```
|map_caption                 = {{map_caption |
```

Templates used on this page:

- Template:!
- Template:Columns
- Template:Coor URL
- Template:Coor dm
- Template:Dablink
- Template:De icon
- Template:Featured article
- Template:Fr icon
- Template:H:title
- Template:Infobox Country
- Template:Infobox Country or territory
- Template:Languageicon

Template:Infobox Country - Fakepedia

http://panther.ecs.soton.ac.uk/wiki/index.php

```
|foot2                           =
|foot3                           =
|foot4                           =
|foot5                           =

|patron_saint               =

|accessionEUdate            =
```

Figure 6.2: Infobox editing in Fakepedia

of the page where the required template parameter is listed as existing, but there is no further information on it provided.

6.3 Results

Six subjects participated, with a range of experience levels from casual editors with only passing knowledge of the wiki system's complicated markup, to experience with administrative tasks. All participants considered themselves to make fewer than ten major edits a week, and only participants five and six considered themselves in the 10–30 band of overall edits per week. Participant ID one is not used as a participant had to cancel after being allocated an ID. We begin with a summary of the recorded behaviours of each editor, then proceed to cross-reference these.

6.3.1 Participant two

6.3.1.1 Off-line review

The participant considered themselves only experienced with basic markup of the system, and to make fewer than ten edits a week. They were multi-lingual, and found information for some articles in their native language version of Wikipedia which was absent in the English version. They had some real-world association to the articles: their place of study, and home town. Because of the cultural and language differences, rather than a straight translation, they effectively re-constructed the article: "I pick some useful things and translate them here; [I] also search on [the] web". The participant also found a 'citation needed' template which had been left by a previous editor, which triggered them to find and add an appropriate reference.

As part of the editing to an article about a town, the participant rewrote a section about the airport to be part of a section hierarchy on transportation, leaving blank subsections for 'Highway' and 'Port'. They clarified that their intent was that they, or some other author, may fill in those sections later: the structure has deliberately been left without content to act as a prompt for further development.

In general, the participant edited articles in topics they had familiarity with, and mostly made text and within-node structural (section) changes. They added text from their own knowledge where they felt that articles were lacking in content, especially compared to other language versions, supported by references found via web search.

6.3.1.2 On-line self-report

The participant was not familiar enough with the wiki system to know how to handle reverse-traversal of links, but when presented with a page with an ambiguous link was able to correct it.

For hyperlinking terms in the ‘cake’ article, the participant’s criteria for linking a term was “noun, or some other concept; maybe not very common, for example”. They chose not to link what they considered basic terms such as “flour” and “vegetable oil”, but would, after researching the meaning, for less familiar ones such as “crystallised fruit”.

To create the summary of country variations of a type of train, the participant said they would read the pages they were summarising, and gain sufficient understanding of the common ground to write a definition “in [their] own words” for the introduction to the summary. They would then create a section for each country, with a paragraph summary, and possibly any applicable tables and lists. They would not re-use the summary text from the specific train page itself, because they would want less confusing, shorter terminology in the summary.

For the film villain, because the information being added is simple, they would just add it to both articles in which that villain is mentioned. If the edit were more involved, they would only edit one, and link the other article to the information.

The participant was not able to complete the infobox task due to inexperience; they said they normally stick to “very simple methods”. They did not entirely understand the London Underground task and would share the information between the pages by linking from the simple to the detailed article.

6.3.2 Participant three

6.3.2.1 Off-line review

The participant edited an article about a story which they had been talking to somebody about, and had looked up in Wikipedia to help explain the plot. They noticed that it contained some incorrect information, and removed it. Because the incorrect information is “a widely-believed but incorrect fact”, they edited the talk page “because someone will just come and put it back again otherwise”. They did not consider the misconception important enough to warrant a mention in the article itself.

The participant also corrected some broken table markup, which again they encountered while looking up information. “Neither time I particularly went out to go and edit something; it’s just that I happened to come across it.” Unfamiliar with the table markup, the participant looked at adjacent rows in the table, and used a little trial-and-error with previewing, to determine how to correct the broken one.

6.3.2.2 On-line self-report

The participant was not familiar with infoboxes, but realised that the template parameters were being converted into display text supplied from somewhere else, and discovered the list of templates in use shown below the edit area. This led them to the documentation, where they found the correct template parameter (which uses different terminology to the display text), and copy-and-pasted it into the article. They guessed that display ordering was defined by the template, so placed it within the source next to similar parameters “to make it easier for other people to edit”. To format the date, they looked at other dates being used as values within the infobox template.

For the ‘villain’ task, the participant noticed that the villain’s description was not quite the same in both film articles, and reasoned that it “must have been copied and pasted once before, so if I want to edit it I’ll have to edit both unless [the identical lead-in] is a template”. As it was not, they decided to edit both. They contemplated separating out the villain into a third node, but didn’t know how to achieve that in MediaWiki.

Ambiguous links were corrected using the wiki’s backlinking feature, although the participant had to consult the help to work out how to make a link have a different target from the text being linked. They used the unvisited colour of the link as an initial indication that the markup had worked and changed the target.

When adding, for the detailed article, that the London Underground was one of the oldest metros in the world, they left the edit as major as they’d “added a new fact, which might be under debate”. They made the same decision for the general article, Transport in London, adding the fact to the Underground section.

The participant used a dangling link to create a new page for the summary of train types, and looked at other pages to see how they began for comparison. They copied summary text from one of the trains, then edited it down to also apply to the description of the other type. Each type was put as a list item under a subheading; they looked at another article to see how to create subheadings. The participant also then decided to add an image from one of the train types, and updated the caption to fit within the new context. Once saved, they used the backlinks feature to test that each of the types of train they were linking to also linked back to the new summary article.

Adding hyperlinks to ‘cake’, they linked nouns, and searched for more complicated phrases to find the best article that existed, ensuring that the links had valid targets. For example, there was no “sweetening agent” page, so the participant instead linked to “sweet”, which does list appropriate information, found via the previous, related noun, “sugar”. For “binding agent”, they could not find a suitable node through searching, so did not link it. Adjacent in the text were some example agents which they did link, commenting that “we should probably link the first one right through, but not bother linking it anywhere else in the document”. One of the agents was “starch”, and to link to the specific use of starch in food, the participant created a link to a section within an article; they had to do this by looking at an example they come across, as they could not find out how from the help.

They did not link "fats" because it was "so general", instead opting to link specific fats such as "butter", "shortening", and "margarine", certain that each would have a node. Simple concepts like "water" and "fruit juice" were not linked either, although "milk" was, reasoning that someone making a cake may wish to find out what kind of milk would be appropriate. They linked "marzipan" without testing that the node exists, as "if it doesn't...that should be a [dangling] link so that people can [add it]".

They considered the edit major, although it didn't add any new information, because "the point of a minor edit is something where it's so obviously right that it's not worth highlighting"; in this case, another editor may wish to see if there are any more links which are appropriate.

6.3.3 Participant four

6.3.3.1 Off-line review

The participant added an photograph of a notable person which they had taken. They took the photo with the deliberate intention of using it for the person's Wikipedia article. To get the markup for adding an image, they used an existing image in the article and changed the filename to the photo they had uploaded.

The participant reverted some vandalism using the special 'undo' feature available from page histories. They used this feature over simply editing out the vandalism (which was a one word addition) because it was quicker. They also added the vandalism warning template to the editor's talk page, found via the history. In one case, they found the vandalism because they were looking up a person who just appeared on the news.

The participant created some redirects from acronyms to an article, "because some people might try and look [it] up like that". They "have to look up [the redirect markup] every time", which they did by finding an example. Sometimes they search for instructions, but consider either approach time-consuming.

They added a "recently died" template to a person's article having seen the announcement on the news, because they had seen this done before. Again, they copied the template from another example article they found, changing fields they recognized by value: the name and date.

The participant also improved indexing of information: adding an article to a disambiguation page when they couldn't find it via said page; and adding an article to a category it had been omitted from, having seen that category used elsewhere for the same class of items. They removed a link to a page from a template because the link was dangling, reasoning that there "probably used to be a page for this", and that they did not have the knowledge to (re)create it. They added references to another editor's unreferenced text, where the article had references which covered the claims, but which were not cited in-place in the article text.

Many of their edits were simply straightforward text or markup corrections for a wide range of articles.

6.3.3.2 On-line self-report

To add the date of EU membership in the 'Belgium' task, the participant looked through the template for a suitable parameter. They found 'established_event1' and 'established_event2', and reasoned that creating a third event would work, which it did. As they noted when shown that the template has a separate parameter for EU membership, this would not be picked up by systems extracting semantics from infoboxes.

In the 'cake' task, the participant chose nouns which were cake ingredients, because that seemed consistent with the rest of the page. They chose primarily simpler terms which they felt were most likely to exist, omitted compounds such as "sweetening agent". They would check the links and remove any which were dangling if expending more time on the task.

The participant had no difficulty with the Shelf task and made quick use of the system's backlinks feature to find pages to correct.

The participant created a summary page with a bulleted list item for each of the train types, using text copied and pasted from the specific train types. If later changes to this shared text were needed, the participant said that both would have to be edited, although there may be a need for the copied text to be different, because of the different context. For adding new train types, they suggested using categories, although this lacks descriptions for each of the category items in the category page. They contemplated using templates to share text, but "it would be quite complex, and [it] is really confusing when you go to a page to edit it and the text you want to edit isn't actually in the page because it's somewhere in a template, but you don't *know* which template it's in".

The participant made two straightforward edits for the villains articles, adding the fact into the flow of the text in both. They hypothesised that the pages differed from where some people edited just one of the copies.

For the London Underground, the participant also added the starting date to the infobox, using the template documentation, found from the listed of in-use templates when editing the page. They edited the same information into the general article's section on the Underground.

6.3.3.3 Remarks

This participant was more familiar with the tools and processes on Wikipedia, but did not consider themselves adept at ‘advanced’ wikitext. Despite this, they show evidence of succeeding at some more complex tasks, such as creating redirects, and using templates.

6.3.4 Participant five

6.3.4.1 Off-line review

This participant edited a different MediaWiki-based wiki, although the domain was still a communal information resource.

They took an image which had been left plainly at the bottom of the article and added the markup to float as a thumbnail in boxout with a caption. They know the template invocation from experience, but originally learnt it from ‘cheat sheet’ documentation.

They also updated the image used for the site’s logo, as the design changed. They chose to use the ‘upload a new version’ feature of the wiki, rather than uploading each version as a separate image, because they are conceptually all the same image.

The participant decided that one of the articles needed an image, so searched the web for an appropriate one.

In general, they corrected text, and in some cases made a few sequential edits where errors were not caught by previewing. They created some new pages by creating dangling links to them, then following those links. There were some more advanced operations, such as adjusting the structure of the table, but these were ultimately a case of changing markup the participant was familiar with.

6.3.4.2 On-line self-report

The participant made a straightforward addition of the fact to both articles for the London Underground task, finding an appropriate subsection for the detailed one.

They chose ‘cake’ terms that they didn’t know, such as “buttercream”. They would also check that the targets of any links existed.

For the trains, they created a mock summary text, with links to the specific countries of train. If they were more confident with the subject area, they would create new, separate text for the summary article than for the summaries in each specific train article; they would not share the summary text. They also suggested categories as a possible approach to handling future additions of types of train, although they would have to look up how from a past instance where they had done so.

The participant added the Belgium EU membership using 'established_event3', after finding nothing else suitable, as for participant four. They followed the example of other dates in the infobox for formatting the value.

For the villain, the participant would split the villain's section out into a new article, and use the 'main article' approach seen elsewhere on Wikipedia by looking at an example. Text which is specific to that villain's role in that film would be left in the section within the film's article, but everything else would be moved into the villain's article. There would not be any explicit reference from one film to others that also have the same villain; this would be left to be implicit via the villain's article. However, the participant would also look for examples of other films which have shared characters.

After some initial uncertainty, the participant decided that they could resolve the ambiguous Shelf links by copy-pasting the correct link targets into pages found by using the system's backlinks feature.

6.3.4.3 Remarks

The participant administered the MediaWiki-based system their off-line review was based upon edits of, and thus was experienced with the MediaWiki software, and the social processes of that specific wiki. The on-line report involved Wikipedia's policies, with which they were less experienced, hence the need to look for examples despite being comfortable with the technology.

6.3.5 Participant six

6.3.5.1 Off-line review

This participant also migrated some information from a non-English Wikipedia article, about which the participant was "very knowledgeable", to its English Wikipedia equivalent. The English version was of poor quality and wrong, so they used a modified translation of the non-English version, and added a few links to articles they knew to exist because they had seen links to them elsewhere.

They also added some information, corrected language mistakes, and turned a term that they knew was an article title into a link, to some articles. The articles are on subjects they are interested in, and one was a correction made to an article about a television series while looking up information about an episode.

6.3.5.2 On-line self-report

To create the train summary, the participant looked fruitlessly for the UI to create a new page (there is no such dedication functionality, although the search box can be used for this), before editing the browser address bar. They created headings for each type of train, and copied in the first paragraphs, and one image from each. Above this, they created a summary heading with some new, general text. After a preview, they note to create links to the train types, and also copy in the 'see also' and 'external links' sections from the specific trains, renamed to be scoped by the type of train. However, they would probably write the summaries from scratch in a real situation, and not use the same text; if corrections were then needed, they would just have to be made in both places.

They immediately commented that the villain appearing in both films was not merged into a single article. They would do this, moving the text away from the films, and replacing it with, and leaving behind any, paragraphs specific to their appearance in that film.

For the London Underground, they would put the starting year in the infobox, but cannot think of a template keyword to use. They link the year when they write it "because on Wikipedia many people do that". They were initially reluctant to add the fact to the more general article until they spotted the section within it relating to the underground.

The participant describes the 'cake' task's problem as "there will either be loads of links or none, because all the things can be really linked to". They pick the most uncommon things, skipping over terms people "should know", such as "flour" and "sweetening agent", as they links would "be too much for reading the text". Instead they pick what they consider ambiguous: "buttercream"; "marzipan"; "piped borders". They link to "special occasions", but not to the examples of special occasions listed after (e.g. "weddings"), because they consider that clause parenthetical, and wouldn't link to things in parentheses.

They attempt to add the EU membership date to 'Belgium' by finding a suitable place in the markup and adding a parameter 'eu_membership_start_date'. They do not realise that templates work on a fixed set of parameters, of which this is not one, and spend a while attempting to adjust the syntax when it has no effect. They expected adding things to infoboxes to be as easy as adding them to tables. The participant used the system's backlinks feature to find pages pointing to the Shelf disambiguation page, and fixed one of them. They opened the page for the expected specific type of Shelf in a separate browser tab so that they could ensure they were linking to the appropriate meaning.

6.3.6 Participant seven

6.3.6.1 Off-line review

The participant made two large groupings of edits: a continuation of some previously-started work on an article about a section of railway which they were preparing in a personal sandbox under their user account's namespace, and a cleanup of a series of computer games.

The railway article contains a diagram of the rail network for this section of track, which is constructed out of a grid of icons using templates. With this, the participant adds some text, and references taken from web searches. The text contains a 'citation needed' claim, as the participant knew the fact but couldn't find satisfactorily specific citations for it, so they decided to "get the gist in" and let someone else fix it.

They found out how to construct the rail network diagrams by spotting other diagrams in rail articles and looking at how they were constructed. They then searched for the templates used to find the documentation, which includes a catalogue of icons. The participant constructs the diagrams within a sandbox because they do not want to leave 'public' nodes in a half-finished or broken state. The preview feature is unsuitable because the diagram templates are very complex and time-consuming to slowly build up; if they make a mistake, they want to have an edit history available to be able to roll back, else they have effectively lost their work as an incomprehensible mess.

Once it is in a "reasonable" state, the participant moves the sandbox article to its intended page about the railway station. First they prepare the links to the target page. They find articles which should be linking to it by physical properties—for a rail network, this is adjacent stations—and make all the link names consistent, as dangling links can often suffer from co-reference problems in naming. They then view the backlinks of the still-non-existent target page to ensure that all pages they expect to be linking inwards are now doing so. Finally, they copy-and-paste the sandbox to the target, and set the sandbox pages to redirect to the new, 'public' page.

For another railway section, the participant had some information left over in their prepared notes while writing about the station which they could not work into the text of the article. Rather than leave these notes unused as a local file on their computer, they put them on the discussion page for the article, with sources, so that other editors may use them if they see fit.

The participant also found some historical pictures of a railway while browsing, and wondered if Wikipedia would have any. They discovered that Wikipedia did not, so added the pictures, and a reference.

The other major set of edits, about twenty in number, affected a series of computer games. There was one article for the series overall, and one article for the second game in the series; the participant felt that the first game should also be split out, the third was not yet up to quality—it should be copy-edited before it is moved—and that the fourth game's section was too small. First, the participant added a template proposing to split the series article apart. They found the template via cheat-sheet documentation, which they access via a shortcut term in the search box. They added the reasoning for the proposed split to the talk page, and received positive feedback from a previous editor, identified from the page history, not user profile.

The participant added an 'in use' template, seen in the documentation, to the series, which acts as an advisory lock to warn other editors that their changes may be lost in conflicts, and to avoid editing now. They cut down the second game's section within the series article because it already had an article. They then factored out the first game to a separate article, created by following a dangling link, although they changed plan slightly during the process: rather than avoiding an existing disambiguation page, they replaced it with this article and added the template for 'for less common meaning X' to the top. This did lose the very short history of the new first game article up to that point, because only administrators can move pages while preserving or merging histories. They also moved out categories and external links to the game-specific articles. After a lot of adjustments, they removed the 'in use' and 'split apart' templates.

In the process, they converted an external link to an internal link about the same piece of software, because they had "seen it done elsewhere as well", and prefer this because it will link to more encyclopædic information than the software's website. They also linked the word "parody", because "not everyone might know what a parody is", although they "try not to overdo it" with such links.

The participant also tended to use the random article feature, and generally copy-edited articles to be more encyclopædic. They removed an unreferenced section which had gone undefended for a period of months.

6.3.6.2 On-line self-report

The disambiguation was much the same 'find backlinks; determine meaning; edit link' process as for other participants, but the participant did observe that they normally edit Wikipedia with an extension which provides on-hover previews of page targets, useful for checking that the new link target is appropriate.

To create the summary of trains, the participant started with links back to the specific articles, using the 'main article' template, tested with a preview. They added the section headings and looked in the articles for text to use, but did not find the introduction suitable, instead using a later section on the history of the train instead. When saving the edit, the participant would include links in the edit comment for the benefit of the page history. They would save this "rough cut", then go back to copy-edit once the structure is in place. In a real situation, they might improve the train articles first, rather than have to propagate back the improvements. A 'see also' link would help keep people aware of where summaries are shared, but once to four or five countries of train, the participant suggests a navigation box, as it is more visible.

The participant added the London Underground fact to both articles, and in the detailed one duplicated the fact in the appropriate history section. They took care to mark the claim as needing citation, and would have provided their source if making this edit in the real system. They considered the edits major because they added information in a prominent location.

For 'Belgium', the participant looked up the template documentation and found the parameter to use, but initially failed to make the date appear because they accidentally applied it to another template nested inside the infobox template. The template syntax was not indented.

To add links to 'cake', the participant opens a browser tab for searching, to "check these things actually do exist first. Before I even preview I'll go and check that." They would not rely on the colouration of links to indicate if they are dangling, as this "gets annoying" and hides that non-dangling links might still be disambiguation pages or redirects. However, on their normal Wikipedia setup, they might use their hover-preview to test. They link "flour" and contemplate "sweetening agent" and "binding agent", but note that there are "an awful lot of things I *could* link to here...but we're going to overload the user with information here, I'm just going to link to some of the ingredients". They link "just a few of the most important ones", such as "sugar", skipping over non-primary ingredients, and also note that they would cut down the text of this intro. "Yeast" and "dough" are linked because they are "slightly unusual", and there should be lots of encyclopædic information about them; "baking powder" is considered "less exciting". The participant would make another pass, and possible add another few links, in a real case. They would ensure that they only link a term on the first instance.

When looking at the films in which the villain appears, the participant notices that the text is identical, and that the duplicated content is "unfortunate". Because of the small change required by the task, and the difficulty of refactoring, they would just make the change in two places, but then suggest on the talk page to perform a split, much like for the game series in their off-line review. They stress the necessity of the talk page announcement: "because this is quite a large change to make, people are going to get upset if you're not careful".

6.3.6.3 Remarks

The participant considered themselves extensively familiar with the MediaWiki software, and clearly demonstrates a high level of experience with Wikipedia practices. They remarked that, especially after the mistake with 'Belgium', templates are "nasty", and the Wikipedia syntax in general is painful. They have to create workflow within their browser with multiple tabs because Wikipedia provides no support for it, even though they are often editing multiple things at once, moving content between them. The hover previews are one way in which they try to improve this.

6.4 Conclusions

6.4.1 Common observations

Behaviour	2	3	4	5	6	7
Same villain in two movies						
Edited both	✓	✓	✓			
(Would) split				✓	✓	✓
Specific and general London Underground						
Edit detailed	✓					
Edit both		✓	✓	✓	✓	✓
Edit infobox			✓			
Disambiguate Shelf links						
Found backlinks		✓	✓	✓	✓	✓
Corrected target	✓	✓	✓	✓	✓	✓
Summarise two types of train						
Rewrite in own words	✓			✓	✓	
Copy existing text		✓	✓			✓
Create category			✓	✓		
Add links to cake introduction						
Nouns, uncommon	✓	✓		✓	✓	✓
Nouns, ingredients			✓			✓
Explicitly first only		✓				✓
Check existence		✓	✓	✓		✓
Add fact to infobox for Belgium						
Added fact		✓	✓	✓		✓
Established_event instead			✓	✓		
Date format used by example		✓		✓		
Used template documentation		✓				✓

Table 6.1: Summary of participant behaviours for tasks on Wikipedia

We set out to determine the goals editors set themselves, and how they act to achieve them. To recap, table 6.1 groups the general approaches against each participant.[1]

[1]"Explicitly first only" for the cake links indicates that the participant made note that they were only linking the first occurrence of words, but no participant linked a word multiple times.

Several of the participants edited articles to correct errors they encountered while following a primary goal of looking up some information. While this is not particularly surprising for cases of simple, non-content corrections, such as markup or typographical errors, it is counterintuitive that people who are looking up information, and thus are presumably not experts in that field, will make more significant edits, such as finding and providing references. However, some participants looked up articles on subjects about which they are knowledgeable, either as a reference, or out of curiosity as to what information Wikipedia would have.

There are three ways shown that editors will select images to add to an article. They may deliberately seek to create them with the intent to then add them to Wikipedia, as with participant four's photograph. They may discover them while browsing on unrelated tasks, then decide to add them to Wikipedia, as with the pictures found by participant seven. Or, they may be editing an article, decide that it needs illustration, and search for suitable images, as with participant five. There is therefore a range of premeditation to major edits such as this; an extreme case for textual editing is the railway work described by participant seven, with preparation of an entire node in a semi-private area of the wiki, and with a local collection of resources.

Learning by example is a common practice to all of the participants, even those who are also adept at using the documentation. Editors often tried to keep their articles, and meta-activities such as edit comments, consistent with those of other editors. They were actively aware that other editors were at work, and in cases implicitly delegated tasks to whichever editor is inclined to address any outstanding issues, by leaving incomplete sections, dangling links, or marker templates (such as 'citation needed'). Dangling links also provided a common mechanism to create new pages, as the wiki has no explicit UI feature to do this.

Even relatively advanced features which do not offer additional capabilities over simpler ones, such as the 'undo' links in the history, may be used if they save the editor time.

In the tasks where participants were asked to share text between articles, most of them decided that they would use different text on the different articles, because of the different contexts. For the specific article/general overview case, they edited the information into the existing contexts with no outward consideration of synchronising the summaries. For the trains, several explicitly stated that different text was needed. Hence, there are cases where what is abstractly the same semantic information in multiple places may still not be sharable, because of differing presentation needs. Conversely, the 'villain' task shows that sharing is suitable in some situations, where there is a larger, mostly self-contained section of content. This task also highlights the need for better knowledge modelling on Wikipedia, as the current articles do not clearly divide the concepts of actors, characters, and films.

Templates were generally troublesome, even to the more experienced editors. While they would technically permit content sharing, as one participant observed, this has the detrimental property of “hiding” away the text while editing, requiring the editor to follow a possible chain of templates to find where the text they wish to change actually exists. Infobox templates made what should be a simple task of adding a statement about a property of a resource a complicated procedure which some participants could not complete without prompting. While Wikipedia, and hence the synthetic environment, currently runs on a non-semantic wiki, we must stress the risks in not breaking away from this templating-for-properties paradigm as one moves on to systems such as Semantic MediaWiki.

The general problem is that templates on Wikipedia, due to their macro limitations, are presentational, not declarative. We have covered this problem with regard to straightforward properties in section 3.4.7, but the rail network diagram activities of participant seven highlight this as a more general problem. The complexity of these templates stems from their need to specify exact layout and rendering of arbitrarily complex graphs, here composited from tiled images, when the actual semantic content is a relatively straightforward set of connections. In this case, simple text display of the relation data is insufficient: there is a more complex transform required to generate appropriate presentation. Other example problem domains are molecular diagrams and family trees. Wikipedia currently primarily uses manually-created images for the former, and the community are investigating approaches to entering and displaying the latter,[2] but all are presentational. Solving this in the general case may be impractical without providing the facility to define Turing-complete transforms, which then introduces security and performance problems.

Both in the ‘cake’ task, and in general editing behaviour, all participants felt that things that exist should generally be linked, but that there is a optimal link density to maintain. The threshold to which they would link terms varies significantly between participants, from most of the nouns, to just a few phrases (nouns and noun phrases being those most likely to be article titles). They also prioritised links differently: some chose the simpler terms; others the more obscure terms. All participants only linked a single instance of each term, and several commented explicitly on this decision.

6.4.2 Implications

We now consider how these observations apply to our proposed hypertext system, to determine the desired ordering of feature importance.

We should note the importance of keeping two common wiki features, despite our push towards stronger hypertext. First, the editors made use of ‘broken’ hyperstructure, such as empty sections and dangling links, so we should *not* attempt to prevent this, as many

[2]`http://en.wikipedia.org/w/index.php?title=Wikipedia:Family_trees&oldid=212894318`

classic hypertext systems did. This is somewhat of a unique point of wikis, in that their mutable nature means that navigating to a non-existent target can have useful behaviour: creating the node. Second, the editors often learn by example, so must be able to view the source of pages to see how some construct is achieved, even if they are not permitted to modify the source. Some wikis entangle the source view with the editing operation such that this is not possible, which then deprives the editors of this valuable source of real-world example usage.

These incomplete states are being used as a form of passive communication between editors. The message is implicit and, interestingly, the recipient is often simply the next editor to encounter the page who has the motivation and experience to act upon it. Because these incomplete states are co-ordination between editing users, they are potentially of no interest to reading users. However, the complexity with hiding them from non-editors is that, on a normal wiki, every user is potentially an editor, even if not logged in. While users which have not created accounts are potentially less likely to undertake major editing tasks (see section 5.6.3), this is heuristic at best, and may discourage editors from getting involved if only because they are not aware of the incomplete changes. The current approach of using different styles of link—by colour, in MediaWiki—has the advantage of leaving the decision, if also workload, of ignoring dangling links to the user.

The ability to edit links from any of their anchors is a relatively simple step from first-class linking, but we did not reveal any compelling evidence that there is a pressing need for this. All the participants, once they had found the functionality in the user interface, were able to use the wiki backlinks tool to find the endpoint at which the link was embedded, and correct it there. This capability may yet prove useful as wikis transition towards semantic links, as many semantic relations are meaningful in either direction (i.e. many properties have inverses), but is not currently a priority.

Level of detail, part of adaptation, may not be as useful as one may theoretically suppose. Abstractly, it would seem sensible that a low-detail version of a page could be used as a summary about that page when linking to it from elsewhere, as with the specific/general task. However, we have found that the surrounding context affects, if not the semantics of the content, the appropriate wording, such that these summaries are not particularly re-usable.

This also affects the use of transclusion with fat and computed links for aggregation. The most obvious application of this functionality in our synthetic tests would be the types of train, aggregating the low-detail summaries of each train type into a general page on the subject. However, this is also the case where we have identified that context affects the re-usability of the content.

Transclusive re-use of content in general, however, has useful cases. Content which is sufficiently self-contained, not a summary in the context of another page, is a potential candidate for sharing.

Edit-time transclusion solves one of the problems identified by a participant: templates hide the text away. This opacity then greatly limits the usefulness of templates for re-use. As such, we consider the transparency that would be afforded by edit-time transclusion worth prototyping.

The template mechanism also greatly overcomplicates property editing. Instance property editing based on class descriptions, as described in section 4.7, would provide a much cleaner interface to this. We consider this feature highly important due to the significant problems with the current approach, but note that non-research implementation work in this area is already underway in the Semantic Forms extension: see section 4.7.

The linking of terms, as stressed in the 'cake' task, is effectively a manual form of generic linking. Outside of the synthetic tasks, this was a common 'minor edit' behaviour, and as such there should be enough benefit from automating it that we consider this a strong priority to develop. However, we must be aware that the task is not as trivial as pattern matching. Editors have varying heuristics to determine if a 'manual' generic link should be applied to a given instance of a term, and while the lack of such variation in a deterministic algorithm may improve consistency, we must ensure that the link density is kept manageable by some comparable means. At least one restriction is reasonably clear: only one instance of a term per document should be linked.

The next chapter presents a model for a class of system, an open semantic hyperwiki, which we feel combines the benefits of open hypermedia, semantic web, and semantic wiki. It supports the features we have identified here as being of particular importance to improving the wiki editing experience.

Chapter 7

Model

7.1 Introduction

In this chapter, we present a model for Open Weekat, a system of the 'open semantic hyperwiki' type. It draws from both past hypermedia models, and the informal model of modern semantic wiki systems. In particular, we preserve the notion that wiki nodes (or 'components' in Dexter parlance) are parallel to semantic web resources. Because these resources are atomic (RDF cannot perform within-component addressing on them, as that is only meaningful for a *representation* of a resource), we have carefully designed our wiki model to not rely on link endpoint specifications which go beyond what can be reasonably expressed in application-generic RDF. Anything about which one wishes to make statements, or to which one wishes to link, must have a unique identity in the form of an URI, rather than some form of (URI, within) pairing.

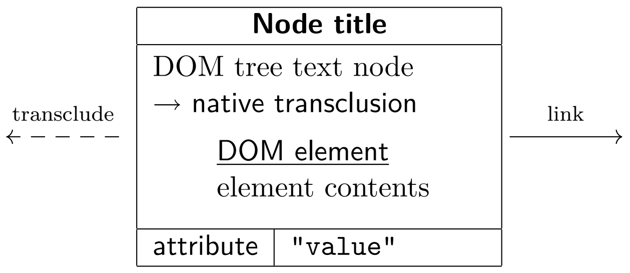


Figure 7.1: Weerkat model diagram legend

7.2 Core type: the node

7.2.1 Components of a node

Figure 7.1 shows the components of a node in the Weerkat model. (We use this diagram format, which is inspired by UML class diagrams, to depict example hypertexts throughout this document.)

Every node has a title which serves as an identifier. These identifiers may act as nested namespaces, depicted here with full stops '.'. This is useful for creating identities for content which nominally belongs within another node.

Node content is either a DOM tree of wiki markup, or an atomic object (e.g. an image binary). A notable element in the DOM tree is the 'native transclusion', which indicates that another node's content should be inserted into the tree at that point. This is necessary to support the linking behaviour described below, and is distinct from user-facing transclusion using normal links (which will be covered in section 7.7.4).

The bottom of the format shows the attribute-value pairs for the node. The domain of attributes is other nodes, and the domain of values are literals and other nodes. These are effectively very primitive semi-embedded, typed links, and are used to provide a base representation from which to describe first-class links. By 'semi-embedded', we mean that such annotations are not separable, uniquely identifiable objects as a true first-class link would be; however, they are not part of the data stream of the node content, and thus can be used even with unmodifiable or opaque binary formats. In particular, in future they should allow 'embedded' anchors to be specified for images, despite the potential use of image file formats which do not store marked regions or intrinsically support arbitrary metadata.

7.2.2 Embedded anchors and linking

Figure 7.2 shows user-level linking. As presented to the user in an example plaintext markup, the source for the Phil node would be:

```
I would call [link type=Likes to=Perl Perl] an [em elegant]
language.
```

We use edit-time transclusion to present the user with the familiar and direct model of embedded linking, but map this into an open hypermedia model. The link element, when written, separates out the link text as a separate, 'anchor' node, and is replaced with native transclusion. A first-class link is then created from the anchor node to the

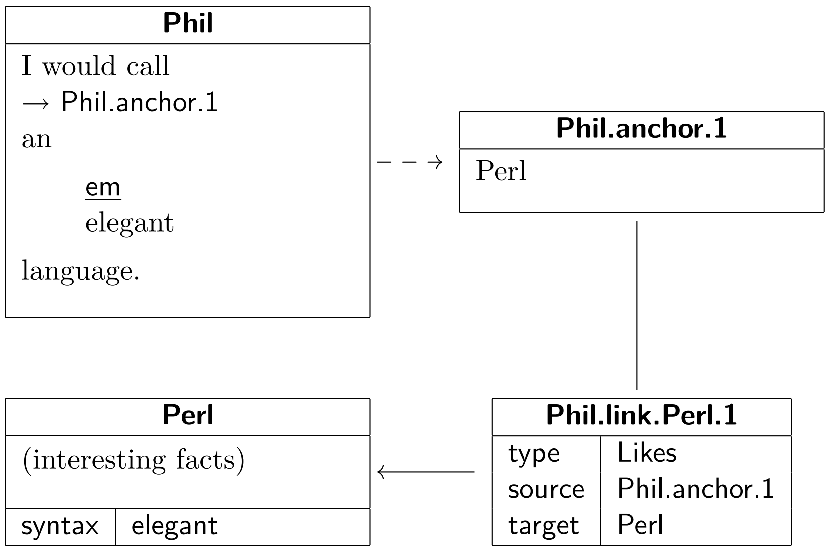


FIGURE 7.2: Weerkat model linking

link target. The identity if this link is largely arbitrary, so long as it is unique, although it may become misleading if its targets are later changed.

Native transclusion is here an optimisation for creating a named, empty anchor in the DOM, then maintaining a link which transcludes in the node of the same name. It is also considered meronymous: a link involving an anchor is considered to relate the node to which that anchor belongs. Because native transclusion is entirely implicit, only the owning node can natively transclude its anchors.

When editing the node again, the anchor is transcluded back into the node, and converted into a link element with targets derived from links affecting the anchor node. (Depending on the exact markup language used to express the DOM for editing, this may require multiple, nested link elements.)

This guarantees that each anchor has a full identity (a node title) in the system. It does not, however, immediately provide a solution to 'the editing problem'—a longstanding issue in hypertext research[60], where changes to a document invalidate external pointers into that document. The anchor names are not here used in the plaintext markup, so ambiguity can arise when they are edited. It should thus be possible to specify the anchor name (as a member of the `*Node*.anchors` namespace) for complicated edits:

```
I would call [link anchor=1 type=Likes to=Scheme Scheme] an [em
elegant] language.
```

A graphical editor could treat the link elements as objects in the document content which store hidden anchor identities, providing this in all cases.

Note that the link's properties in figure 7.2 are stored as attributes. Theoretically, in the RDF presentation described in the following section, an attribute-value pair (source, Phil.anchor.1) in the node Phil.link.Perl.1 is identical to a link of type source from the link to the anchor. However, such an approach would become infinitely recursive, as the source link's source could again be described in terms of links. The attribute-value pairs thus provide a base case with which we can record basic properties needed to describe first-class links.

7.3 Identity, meta-nodes, and RDF

It is a design goal of the Open Weerkat wiki model that the hyperstructure of the wiki is isomorphic to a useful RDF graph, in much the same vein as existing semantic wikis (e.g. Semantic MediaWiki [1].) That is, typed links between pages are expressible as an RDF statement relating the pages, and attributes on a page are statements relating that page to the associated value. The link in figure 7.2 should be presented (via RDF export, a SPARQL endpoint, or such) as the triple (Phil, Likes, Perl), with appropriate URIs. (Note that the anchor is not the subject—we follow the native transclusion back into the owning Phil node.) The attribute of the Perl node in the same figure should be presented as the triple (Perl, syntax, elegant).

For this, we grant each node a URI, namespaced within the wiki. However, 'the node Perl' and 'Perl the programming language' are separate resources. For example, the node Perl may have an URI of `http://wiki.example.org/node/Perl`. Yet a typed link from Perl is clearly a statement about Perl itself, not a node about Perl. The statements are about the associated resource `http://wiki.example.org/resource/Perl`. (This may be owl:sameAs some external URI representing Perl.)

In order to describe the node itself (for example, to express that a node is in need of copy-editing), a Perl.meta node represents the URI `http://wiki.example.org/nodes/Perl`. This meta node itself has an URI `http://wiki.example.org/nodes/Perl.meta`, and could theoretically have a 'meta meta' node. Effectively, there is an 'offset' of naming, where the wiki identifier Perl is referring to Perl itself semantically, and the Perl node navigationally; the identifier Perl.meta is referring to the Perl node semantically, and the Perl meta-node navigationally.

7.4 Representations

Each `node` URI should have a representation predicate which specifies a retrievable URL for a representation. (We do not claim to have any authority to provide a representation of Perl itself, merely our node about Perl.) For example, (wiki:node/Perl, representation, `http://wiki.example.org/content/Perl.html`). There may be more than one, in which case each should have a different MIME type. Multiple representations are derived from the content: for example, rendering a DOM tree of markup to an XHTML fragment. Hence, the range of MIME types is a feature of the rendering components available in the wiki software to convert from the node's content.

Should an HTTP client request the `wiki:node/Perl` resource itself, HTTP content negotiation should be used to redirect to the best-matching representation. In the spirit of the '303 convention' described in section 2.2.5.1, if the HTTP client requests RDF, they should be redirected to data about the requested URI: i.e. one meta-level higher. This inconsistency is unfortunately a result of the way the convention assumes that all use of RDF must necessarily be 'meta' in nature, but we have considered it preferable to be consistent with convention than to unexpectedly return data, not metadata, RDF in what is now an ambiguous case. Clients which wish to actually request the Perl node's content itself in an RDF format, should such exist, must find the correct URI for it (e.g. `http://wiki.example.org/content/Perl.ttl`) via the representation statements.

Requests to resource URIs (e.g. `wiki:resource/Perl`) are only meaningful in terms of the 303 convention, redirecting RDF requests to data about `wiki:node/Perl`. There are no representations available in the wiki for these base resources—only for nodes about them—so any non-RDF type must therefore must be 'Not Found'.

7.5 Versioning

7.5.1 Sessions

Nodes are versioned in their entirety: both content and attributes. Because links are simply nodes with particular attributes, links are also versioned. Edit-time transclusion, and the 'compilation' of links into embedded representations for editing, make it necessary to support the user modifying multiple nodes with one editing session, unlike most wikis. We shall not dwell on these points, as editing sessions covering multiple resources are common to source code version control systems, and have been previously covered in hypertext research.[74, 75, 76]

7.5.2 Revision identity

We must give consideration to the identity of versions, or 'revisions' of a node. We wish to support navigational linking (including transclusion) to old versions of a node. However, we must also consider our semantic/navigational offset: we may wish to write version 3 of the Perl node, but we do not mean to assert things about a third revision of Perl itself. Likewise, a typed link to version 3 of the Perl node is not a statement about version 3 of Perl: it is a statement about Perl which happens to be directed to a third revision of some content about it.

Node title	Content (navigational)	URI	Represents (semantic)
Perl	Text about Perl	Perl	Perl itself
Perl_3	v3 of text about Perl	Perl	Perl itself
Perl.meta	Text about text about Perl	Perl/meta	Text about Perl
Perl.meta_4	v4 of text about text about Perl	Perl/meta	Text about Perl
$\text{Perl}_3\text{.meta}$	Text about v3 of text about Perl	Perl;3/meta	v3 of text about Perl
$\text{Perl}_3\text{.meta}_4$	v4 of text about v3 of text about Perl	Perl;3/meta	v3 of text about Perl

Table 7.1: Weerkat model version identifiers

We desire three properties from a system of generating identities for old versions:

Semantic consistency. Considers the version of the content about a resource irrelevant to its semantic identity. All revisions of the Perl node are still about the same Perl.

Navigational identity. Each revision of a node (including meta-nodes) should have distinct identity within the wiki, so that it may be linked to. Intuitively, despite the above, version 3 of the Perl node is Perl_3, not Perl.meta_3.

Semantic identity. Each revision of a node (including meta-nodes) should have a distinct URI, such that people may make statements about them. (Perl_3.meta, writtenBy, Phil) should express that Phil wrote version 3 of the content for the Perl node.

We can achieve this by allowing version number specification both on the node and any meta-levels, and dropping the version specification of the last component to generate the RDF URI. Table 7.1 demonstrates this using an example syntax of subscripted numbers to indicate versioning. Should somebody wish to make statements about version 4 of the text about version 3 of the text about Perl, they could use the URI `Perl;3/meta;4/meta`. This is consistent with the 'node is resource itself; meta-node is node' approach to converting typed links into statements. Additionally, we have no need to attempt to express that Perl and Perl_3 are the same semantic resource, as this mechanism allocates them the same URI.

It should be stressed that *namespacing* components of the node identifier cannot have versions attached, as any versioned namespace content does not affect the node content. For example, $\text{Languages}_2\text{.Perl}_3$ is not a valid identifier, and would be isomorphic to Languages.Perl_3 if it were.

7.5.3 Mutability of links

Clearly, as nodes, it is very trivial to version links if the user chooses to edit them directly: say, changing their type. However, they suffer much the same problem as the embedded anchors when being edited from their 'compiled' representation in a node. If

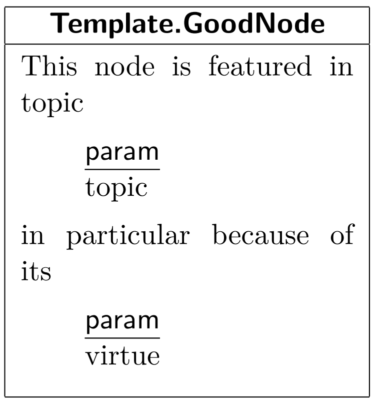


FIGURE 7.3: Exemplary parametric node

the user changes the target of an apparently embedded link from Perl to Scheme, is it still the same first-class link?

We believe that the best-defined behaviour in this case is to consider such a change the creation of a new link, and the cessation of participation in the old link. This necessarily means that a new version of the old link is created which no longer references the node. (In the interests of cleanup, if the link only has one other source or target, and has no content associated with it, it should be deleted entirely.)

This does not mean that it is impossible to change the targets of a link: such an action is possible by editing the link's node directly. Instead, we treat changing the targets of what is conceptually the same link, or *relation*, to be a minority case, with the most prominent mechanism for changing the links between pages instead creating conceptually new relationships.

7.6 Parametric nodes

7.6.1 Node identity

MediaWiki's form of transclusion, 'templates', also provides for arguments to be passed to the template, which can then be substituted in. This is in keeping with the general MediaWiki paradigm that templates are solely for macro processing of pages.

We propose a generalisation, whereby pages may be *instantiated* with arbitrary key/-value pairs. The range of our links are node identifiers, so we consider these parameters as part of the identity of an *instantiation* in a (likely infinite) multi-dimensional space of instances. Figure 7.4 shows a subset of the instance space for a node, figure 7.3, which has parameters topic and virtue. There is assumed to be an instance at any value of these parameters, although evidently all such instances are 'virtual', with their content generated from evaluating the parametric Template.GoodNode node.

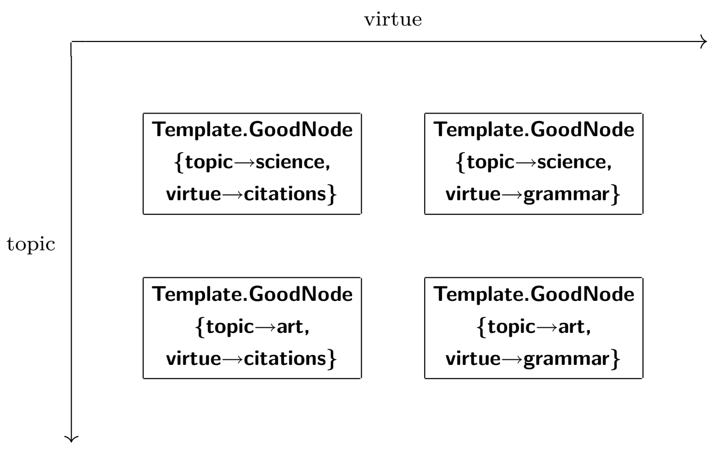

virtue
topic
Template.GoodNode {topic→science, virtue→citations}
Template.GoodNode {topic→science, virtue→grammar}
Template.GoodNode {topic→art, virtue→citations}
Template.GoodNode {topic→art, virtue→grammar}

FIGURE 7.4: Instance-space of a parametric node

It may initially seem appealing to model links to instances instead as a (*identifier*, *parameters*) pair. However, remember that our model of linking is designed such that endpoints consist solely of an identifier, so as to map neatly with the Semantic Web model that any resource worth making statements about should have identity. As such, parameters would need to be applied to the link as a whole but, even with node-namespacing of the parameters, this causes problems should a link have two endpoints to the same node. For example, a link with two endpoints to Template.Delete, with different reason values. Granting instances in-system identity is useful, as it encapsulates all necessary context into one handle.

To guarantee that all isomorphic instantiations of a page use the same identifier, parameters must be sorted by key in the identifier. For example, Locked {reason→vandalism, duration→3} and Locked {duration→3, reason→vandalism} refer to the same state of the node, yet are distinct identifiers. By mandating sorted parameters, the ambiguity is resolved, and only the latter is permissible. Note that this is orthogonal to user interface concerns—the restriction is upon the identity used by links to refer to 'this instance of this node with these parameters', not upon the display of these parameters when editing such a link. As with revision specifiers, parameters upon namespace components of the identifier are meaningless and forbidden.

Within the node's content, parameters may be used to fill in placeholders in the DOM tree. These placeholders may have default value should the parameter not be provided; and the default-default parameter is to flag an error. For example, a parameter may be used to fill in a word or two of text, or as the target of a link. User interface operations upon Foo {bar→baz}'s content, such as viewing the history, and editing, should map through to Foo, as the instance has no content of its own to operate upon.

Because we model parameterised nodes as a (possibly infinite) set of static objects with first-class identity which are simply instantiations of a general node, identifiers which do not map to a valid instantiation of a node could be considered non-existent targets.

Template.Discussions	
type	discussion
source	$param$(node).meta
target	Discuss.$param$(node)

FIGURE 7.5: A parametric, functional link

For example, an identifier which specifies a parameter which does not exist.

7.6.2 Resource identity

We must consider whether such instances are separate Semantic Web resources to each-other, and to the parametric node from which their content is derived. As with version specifiers, parameters affect the *content* of a node, not the resource which it describes. Because the Perl node represents Perl itself, it follows that Perl {bar→baz} still represents Perl. However, as with version specifiers, these node instances still have distinct identity as nodes. As Perl.meta represents the Perl node, so does Perl {bar→baz}.meta represent the Perl {bar→baz} node. Therefore, we can form a URI for a parametric node instance in exactly the same way we form URIs for specific revisions, defined in section 7.5.2. In brief, the final set of parameters are dropped.

RDF expressions of the hyperstructure should specify that parametric node instances, where used, are derivations of other nodes. For example, (Perl {bar→baz}.meta, instanceOfTemplate, Perl.meta) (remember that we are making a statement about the Perl nodes, not about Perl). This is distinct from being the instance of an OWL class.

7.6.3 Parametric links

If we also allow parameter values to substitute into the attributes of a node, we can create parametric links. Figure 7.5 shows such an example which links all meta-nodes to discussion nodes. For example, a value of Perl for node would link Perl.meta to Discuss.Perl.

These are different from generic links in that they are matching on node titles, not terms within the node content. However, if there were a function $contains(t)$ which evaluated to the set of nodes containing the term t in their content, then a parametric link with source $contains$(term) and target term would be a generic link for any term term.

7.6.4 Eager vs. lazy evaluation

The infinite space of non-link parametric node instances can be considered to not exist until they are specified as a link target, as their existence or non-existence in the ab-

Template.FancyLink	
type	Link
source	*param*(from)
target	*param*(to)
decoration	`fancy`

FIGURE 7.6: Free-variable parametric link

sence of explicit reference is irrelevant. However, parametric node instances which are links have the ability to affect parts of the hyperdocument outside of their own content and relations: this is the nature of first-class links. Hence we must consider whether parametric node instantiation, at least for link nodes, is eager (all possible instances are considered to always exist) or lazy (instances only exist if they are explicitly referred to).

As well as the previous examples, figure 7.6 highlights a case where this distinction is particularly significant. With lazy evaluation, this template could be used as a macro, in a 'classical' wiki style, to create links. One would have to create links to instances of this link, which would then cause that particular instance to exist and take effect, linking its from and to parameters.

An eager approach to evaluation would, however, treat parametric links as free-variable rules to satisfy. All possible values of from and to would be matched, and linked between. In this case, every node in the hyperdocument would be linked to every other node.

Logically, eager evaluation is more consistent, and potentially more useful: free-variable links are of little utility if one has to explicitly provide them with possible values. It has been previously covered, in the context of transcluding Wikipedia-style templates in section 3.4.7, that using macro facilities to express common semantics is poor modelling. It would be better to manually link the nodes, with a type of FancyLink which is then defined to be fancy. If there were some content provided by the Template.FancyLink template, it could still be used, but would simply display this content rather than actually functioning as a link. This is similar to the suggested use of 'infoboxes' to only query and display, rather than assert, properties of a resource in section 3.4.7.

While is is conventional on wikis to permit links to non-existent nodes, eager evaluation of parametric links must constrain itself to nodes which exist, else the set of possible parameter values will likely be infinite.

7.7 Expressing features

7.7.1 First-class, n-ary links

As shown in section 7.2.2, links are also nodes, which provides them with first-class identity. If a node has at least one source and target, it is a link.

Multiple link targets, for a navigational link, should be presented as a 'fat link' that the user may choose from. Transclusive and adaptive links modify this behaviour, as described in the respective sections below.

When expressing a link as an RDF relation between nodes, the cross-product of all sources and targets should be taken, as RDF statements may only have one subject and one object. For example, a link with sources (A, B) and targets (C, D) of type t should be expressed as a set of RDF statements ((A, t, C), (A, t, D), (B, t, C), (B, t, D)). This is the same approach that has been suggested for converting XLink links, which are an mechanism for describing n-ary hyperlinks in XML, into RDF statements [77].

Because links are a specialisation of nodes, and can be edited directly, it is possible for authors to create links which do not have any targets. We allow these links to exist as part of the 'work in progress' state of authoring, but they should not be displayed when navigating the wiki. Links without sources also clearly do not have any nodes upon which they would appear.

Conversely, wiki conventions state that, while we can detect 'dangling' links (those which point to non-existent resources), we should still display them. Navigating to a node which does not exist generally prompts the UI to encourage the user to create it. This is used, again, as part of the authoring process—creating links to nodes which the author intends to create later, as shown by experimentation in section 6.4.2.

7.7.2 Adaptive hypermedia

The standard wiki model deliberately tightly-couples content and identity in the form of nodes, as this is convenient for the majority case of writing about distinct resources. This coupling is also present in our model for the same reasons.

As such, we specify an alternateNode predicate, whose object is another node which represents a resource which is `owl:sameAs` the resource of this node. This allows there to be multiple nodes, with distinct node IDs, which can offer significantly differing representations of the same resource. For example, there may be an ordinary Perl node, and a Perl.simplified node. By allowing these as distinct nodes, rather than multiple representations of Perl alone, the former may relate to the latter: e.g. to transclude it.

Each node on may also be annotated with whatever alternation metadata is needed: for example, language.

As one of these alternation data, we suggest detailLevel. For level-of-detail alternation, it is necessary to express detail levels in some usefully consistent manner. Given the task domain is to construct documents, it is likely more useful to indicate absolute, rather than relative, detail levels: a sentence, a paragraph, a section, a whole page. Such values can also be automatically computed from the node content, avoiding the need to manually specify them. ‘Detail’ is orthogonal to the complexity of a node (e.g. its use of domain-specific terminology).

If a link target is a node with alternatives, then all of the alternatives are considered to be targets of the link when navigating. If a link’s type is a subclass of the alternation type, then only one of its targets, rather than all as a ‘fat link’, should be selected. We do not currently specify the criteria for this alternation, or how custom criteria may be specified.

7.7.3 Generic and functional links

The source and target predicates have a range of node identifiers, which makes them suitable for ordinary, static links. However, they are not sufficient to represent generic and, more generally, functional links.

For these, we specify sourceQuery and targetQuery predicates. Their range is that of literal embedded queries, which provide link endpoints. These allow for computed links, although, to avoid interdependencies, the results of a link with an endpoint query cannot be found as the results of another query.

Queries may include content matching, such that they may act as generic links. Local generic links may be expressed as queries with both by-content and by-metadata criteria, such that the latter constrains the set of nodes to be matched by the former.

Functional links are possible to some extent by the use of parametric nodes which are links with free variables. Section 7.6.3 covers some simple examples.

7.7.4 Transclusion

A link is transclusive if its type is a specialisation of transclusion. A transclusive link replaces the display of its source anchor contents with its target contents. As discussed in section 3.4.7, and unlike the ‘native transclusion’ in section 7.2.1, user-level transclusive links do not imply a part-of relation.

Transclusive links with multiple targets which are *not* also adaptive must composite all targets into one sequence, although the ordering for this is currently unspecified.

Edit-time transclusion is user-interface specific, although quite similar to the issues already covered in section 7.2.2 with the native transclusion performed by embedded anchors. For a simple, text serialisation interface, such as a web form, it is possible to serialise the transcluded content in-place with a small amount of surrounding markup; if the returned text differs, this is an edit of the transcluded node. Again, richer, graphical editors can replace this markup with subtler cues.

'Cold' transclusion, again described in section 3.4.7, is served by being able to address a specific version of a node to link to. While a link to Perl will always navigate to the newest version of the node, a link to Perl_3 will always remain a link to that specific version, regardless of future edits to the node.

7.7.5 Templating

Templating can be achieved through the use of parametric nodes and transclusion. Simple macroing functionality, as in contemporary wiki systems, is possible by transcluding a particular instance of a parametric node which specifies the desired parameter values.

It should be stressed that parametric nodes are *not*, however, a macro preprocessing system. As covered in section 7.6.4, parametric links are eagerly evaluated: i.e. they are treated as rules, rather than macros which must be manually 'activated' by using them in combination with an existing node. In general, use of macroing for linking and relations is discouraged, as it is better expressed through classes of relation.

7.7.6 Composition

The range of features in the model provide at least three distinct ways to achieve composition. Figures 7.8 through 7.10 depict these, with some of the previously discussed minutiae of linking omitted for clarity. Figure 7.7 shows the desired result of the composition.

The parametric node approach provides composition by means of substituting the parameter value into the parametric node's content. For compositions where, as with the example, small pieces of content are being composed in the context of some encompassing content, this is the cleanest approach. Note that parameter passing is by value, and that the content being passed does not have distinct identity. A variation of this for cases where sharing is desired would be to specify a link target as a parameter, thus effectively substituting a specified node and achieving pass-by-reference.

Figure 7.9 shows that a document may use multiple transclusive links to compose multiple nodes into its own content. (For simplicity of the diagrams, the links are shown embedded here, but would have first-class representations as discussed in section 7.2.2.)

This approach is useful for cases where the content being composed consists of larger, shared nodes. It is possible to interleave the transcluded content with content within the Haskell node itself. This approach is not particularly suitable where content must be included in groups, as with the example. The related contents are not kept together, as with parametric nodes, and can only be edited together when they are already used together (via edit-time transclusion).

Finally, figure 7.10 shows a composite, or 'fat', link approach. This is largely similar to the composite document approach, and edit-time link compiling would cause both to appear very similar, with the distinction that this approach only allows one type (that of the link). The multiple targets of this link could be provided by some form of query, allowing for composition of dynamically-calculated sets of nodes. At present, however, ordering is not specified for the targets of a fat link.

In terms of the semantics of inter-node relations, none of these approaches is particularly meaningful, as they are all content operations: that is to say, none of them say anything about Haskell. It is perceived as a strength of the model that there is more than one way to achieve composition, as all are well-defined, and offer greater user flexibility to choose the most appropriate approach for each specific case.

7.8 Conclusions

We can summarise and formalise the classes and predicates covered in this section as a small ontology, summarised in the class diagram in figure 7.11. Appendix C expresses the ontology in more detail, using the Terse RDF Triple Language (Turtle)[1]. Our model has be designed to consist of a very small and clean set of types, with expressive power resulting from combination of features. For example, a parametric link is literally a node which is both parametric and a link.

In the next chapter, we discuss how this model can be realised, in particular in terms of an existing, modular wiki system. We give particular attention to the efficient implementation of the rich link endpoint types we have covered here.

[1]`http://www.dajobe.org/2004/01/turtle/`

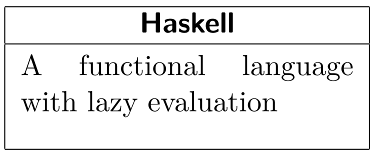


Figure 7.7: Desired composition

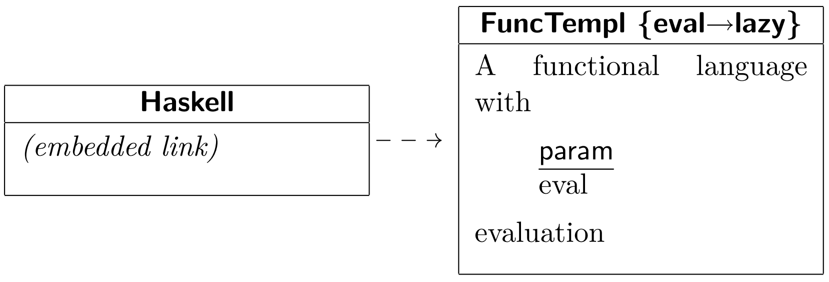


Figure 7.8: Composition by parametric node

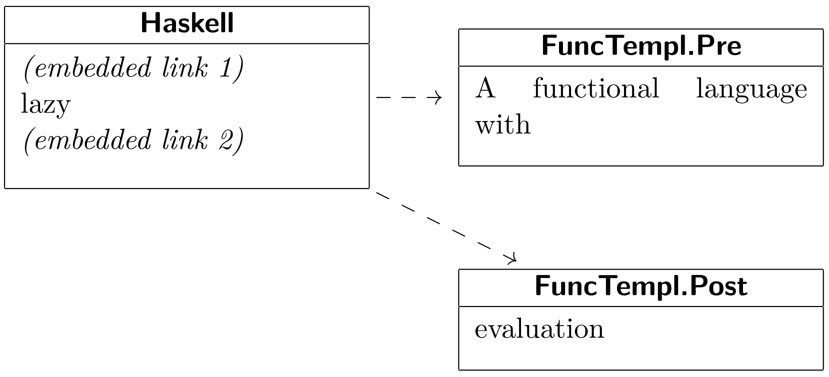


Figure 7.9: Composite document

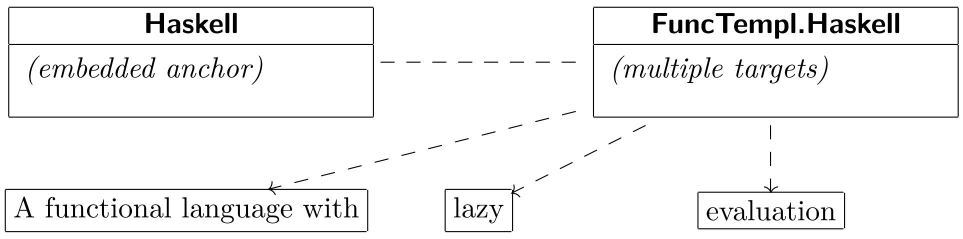


Figure 7.10: Composite ('fat') link

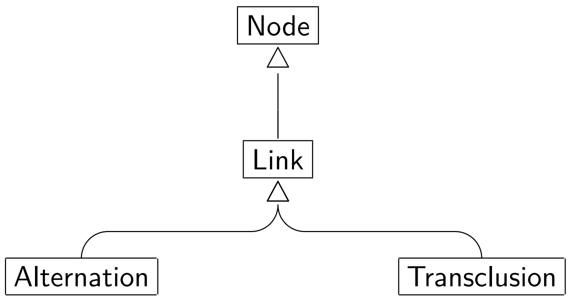


Figure 7.11: Open semantic hyperwiki model ontology

Chapter 8

System design

8.1 Introduction

We present in this chapter the existing wiki system, Weerkat, upon which we plan to base implementation of the model in prototype form. This is followed by a description of, and solution to, the issues with adding first-class links while remaining within the embedded-link paradigm of the web and wikis, and mechanisms for implementing the more advanced endpoint types in an efficient manner. Finally, we cover the changes to Weerkat which will be required to improve it into an open semantic hyperwiki.

This allows us to demonstrate the extent and depth of changes involved, even for a system designed with independent modules for flexibility. It is because of the wide scope of changes that we did not develop our prototype as an extension for the established Mediawiki system.

8.2 Existing technology

Weerkat [44] is a highly modular and extensible wiki system designed for research into the possible interactions between wiki technology and other fields. Figure 8.1 shows the high-level modules in the design. Each of these modules is defined by an abstract interface such that it can be chained and composited with other implementations. In particular, the rendering module used by the system, which is responsible for converting abstract node contents into concrete, displayable content, is the ChainRenderer implementation. This implementation does no rendering itself, but instead links together a sequence of renderers, much like UNIX-style pipes, or Microcosm filters [22].

The existing Weerkat system has trivial support for generic linking from terms to resources in an ontology browser, which is provided by a pair of 'semantic linking' renderers. The chaining approach means that neither of these has to be specific to the details

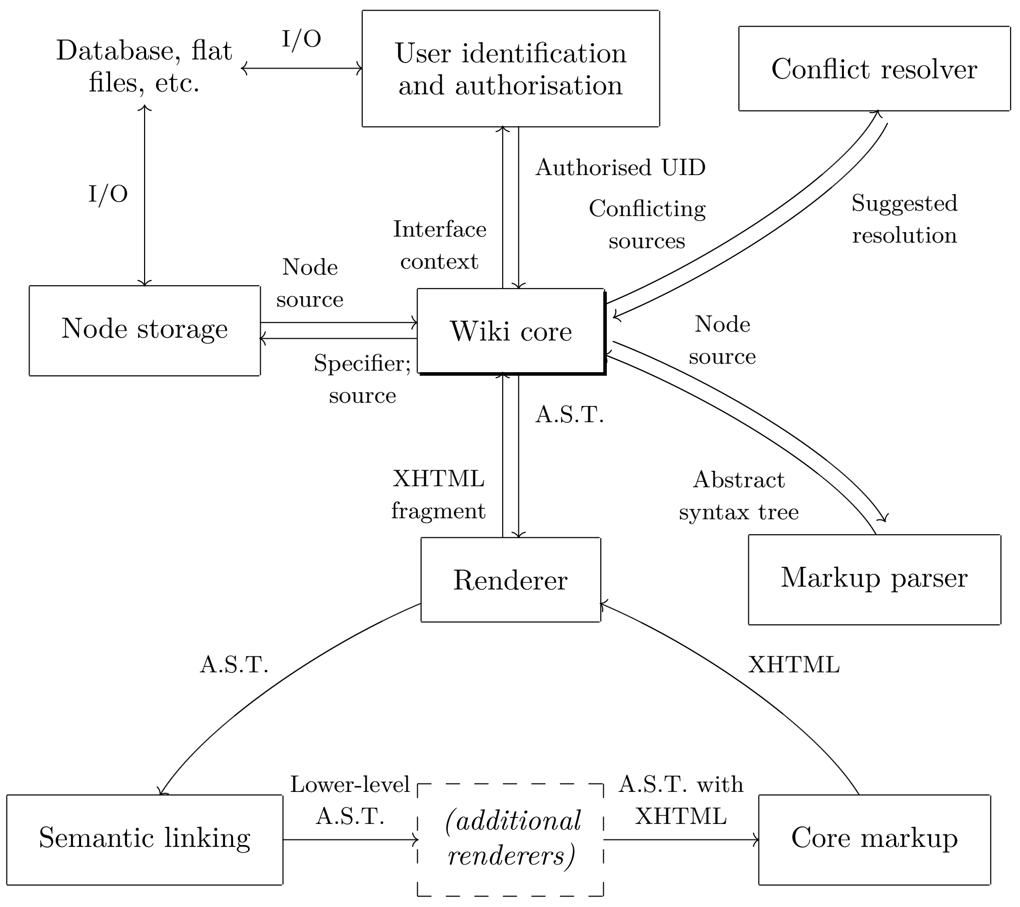


Figure 8.1: Legacy Weerkat high-level design

of the output format (XHTML). The first is responsible for generic linking, and adds abstract links from terms it identifies to resources. The second then converts any links to resources, including those which users may have explicitly defined, to abstract links into the ontology browser. It is then the responsibility of a later renderer, XHTMLFinal, to convert this simplified, 'core' markup into XHTML. It is trivial to add more functionality into the node content by inserting additional renderers into the chain, which perform similar partial conversions.

It should be stressed that the legacy Weerkat system is *not* a 'semantic wiki' in the modern sense, as the wiki structure itself does not define the ontology.

8.3 Link embedding

The linking model of legacy Weerkat is embedded, and roughly web-style. The new model is much richer, with first-class, separate links. One of the largest changes to be made to the system is to add support for this new model.

The model uses first-class links; however, the user interface will present these links as embedded for editing, and must embed them for XHTML output. Hence, we must

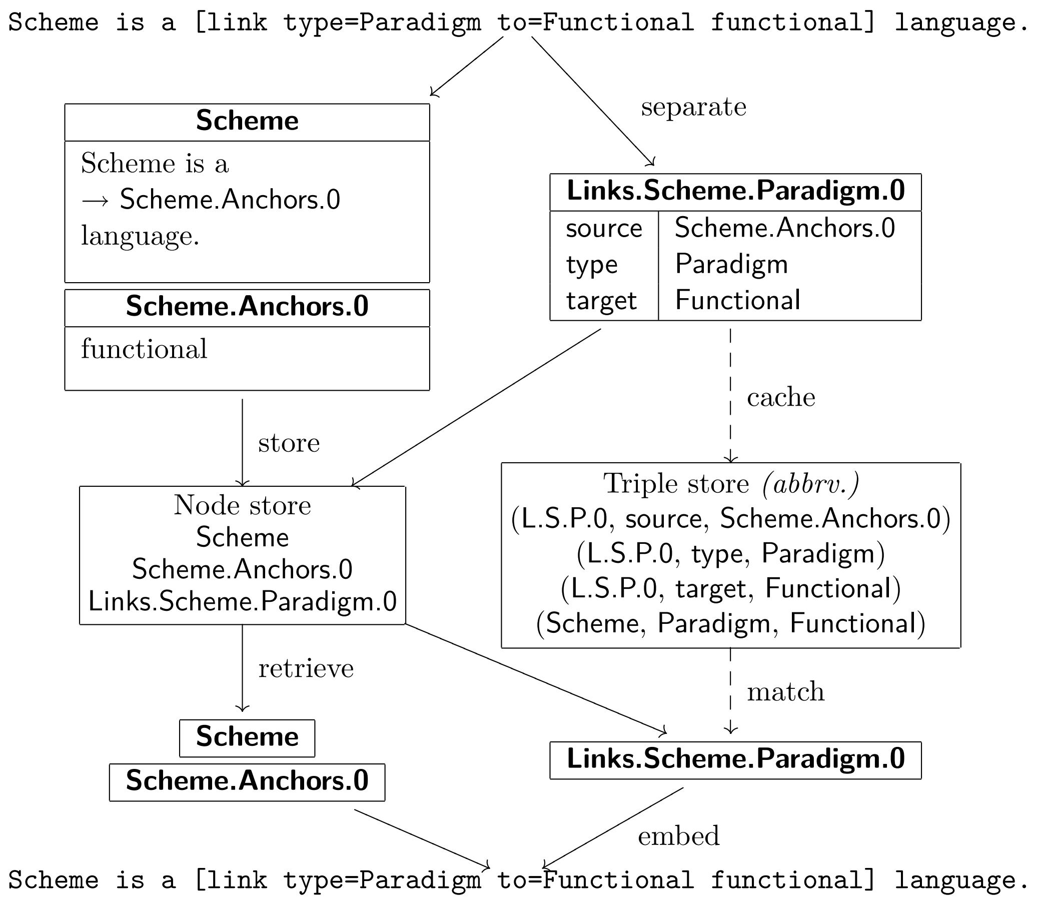


FIGURE 8.2: Separation and embedding of a simple node's source

transform links between these two forms, as suggested in Moreau and Hall, and Brown and Brown [38, 61]. Figure 8.2 shows an overview of the process for the following example.

Unfortunately, the 'compile' and 'decompile' terminology, for combining a linkbase and set of documents into a set of documents with embedded links and back, is at odds with the 'compilation' terminology in computer programming, and the process of converting embedded-markup source code into a set of abstract structures. To avoid potential linguistic confusion from 'compiling' a node structure into node source, we instead use the more direct terms 'embed' and 'separate'.

8.3.1 Separation

Let us take an example node which the user is submitting to the system, Scheme, with the following markup as content:

```
Scheme is a [link type=Paradigm to=Functional functional] language.
```

This must be converted into a structure as described in section 7.2.2, with the node text, the subordinate embedded anchor, and a link.

The embedded link must be replaced in the Scheme DOM with a native transclusion to an embedded anchor, which we shall generate an new identifier for: Scheme.Anchors.0. If the markup were to specify an anchor identifier, that would be used instead, under the Scheme.Anchors namespace. This may specify a previously-existing named anchor, in which case a new revision of it is created.

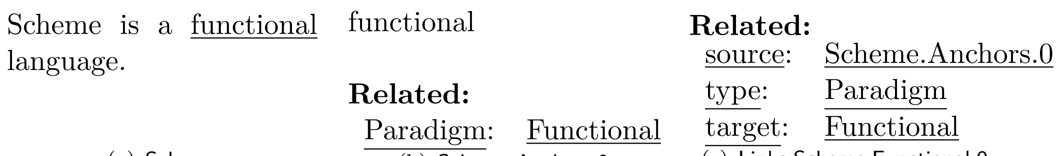
We must check to see if the type of the link is a subtype of the wiki-specified Transclusion type. If not, then the content of the anchor is the content of the embedded link: the text `functional` in this case. Otherwise, the content of the anchor is irrelevant, as it will be replaced by the transcluded content.[1] However, if the user has provided a non-empty body to the link element, this should be set as the new source text of the target node, complete with necessary recursive separation. If there are multiple target nodes, this behaviour would be ambiguous, so should be forbidden.

The link from the anchor to the embedded link target (or targets) must now be made first-class. It will have no content (unless it is a new revision of a link to which a user has added such), and types and endpoints as specified by the attributes of the DOM node. If the DOM node specifies no sources, the embedded anchor is a source. Likewise, the targets default to the anchor unless specified otherwise. This allows for both any-endpoint link editing, and the embedding of links with computed endpoints.

As no identifier has been specified in this example, we first attempt to identify it as an unmodified link: if there is a link whose set of types exactly matches the set of types in link DOM element attributes, and which matches with a source endpoint, and whose type and target endpoint sets match exactly with the link element attributes, we need perform no more action here. Else, we consider this a new link and generate an identifier: Links.Scheme.Paradigm.0. Whereas anchors are namespaced under the node they belong to, links are globally namespaced, as they can potentially exist without any particular 'owning' node.

The inclusion of link properties (which may be changed) in the identifier is a compromise to allow for human-readable identification of simple, web-style links, which are ordinarily 'changed' by being replaced (see chapter 7.5.3). For more complicated first-class links, whose sources and targets are more likely to change while conceptually remaining the same relation, users would be better to create links whose identifier does not encode this information. Note that use of the Links namespace is an organisational convenience: the system detects links by their type.

[1]Strictly, in a system permitting dangling links, this may be useful as a fallback. However, we have judged the edit-time transclusion feature, which occupies the same syntactic niche, more valuable.



Scheme is a functional language.

(a) Scheme

functional

Related:

Paradigm:	Functional

(b) Scheme.Anchors.0

Related:

source:	Scheme.Anchors.0
type:	Paradigm
target:	Functional

(c) Links.Scheme.Functional.0

FIGURE 8.3: Viewing links which apply to the whole node

The link is created with no content, but attributes which specify the source, type, and target endpoints. These are taken from attributes of the link DOM element, and an implicit source endpoint of the node in which it was embedded.

We have now separated the embedded link and created (or updated) three nodes, which should be stored as their DOM and attributes. For performance in link matching, the attributes of each node should also be expressed as RDF (see section 7.3) and cached in a triplestore. This includes any relations implied by static-endpoint links, so that query-endpoint links may operate upon them. We should also perform cleanup. Any node in the Scheme.Anchor namespace which is no longer referenced should be deleted. If this then results in links with no valid source endpoints and no content (i.e. no user has treated them as first-class), those links should also be removed. Note that these conditions will not be met by removing an embedded form of a computed endpoint link, or a target-side link, from the document source. For example, a generic link from the word 'Scheme' may become embedded during compilation, but removing it and saving the document will not delete the link.

8.3.2 Embedding

We now consider the problem of embedding these structures back to a text string of node source markup.

Serialising the node we are viewing is a relatively straightforward operation, and details are left to the markup implementation in use, as with parsing. Native transclusions are recursively inlined, although we shall first have to potentially wrap them in additional DOM elements. (It should be noted that a native transclusion which does *not* end up so wrapped will be folded back into the node's main content during the separation process.)

We must also find which links must be embedded into the node. We call this process 'link matching' and, as it is quite involved, describe it in the next section. Links which match below the level of the node being displayed are inserted as DOM link elements. Links which match the node being displayed are to be shown as separate navigation, as they apply to the visible document as a whole.

Figure 8.3 shows the effects of these rules, with DOM links highlighted by underlines, as in a contemporary user interface. Note that the link on the word 'functional' is in-line when viewed in the context of the surrounding document in (a), but migrates to document-level navigation when the anchor is viewed alone in (b). Likewise, the attributes of the link node are considered document-level when it is viewed in (c).

8.4 Link matching

To determine which links should be applied to a node (including embedded anchors), we must find those which have an endpoint which 'matches'. Links may match on either source or target endpoints, so as to allow any-anchor link editing, although the user may wish to embed only outgoing links in some cases, and the normal display of a node will likely only use this information as an out-of-document 'incoming links' sidebar. For these cases, it is sufficient merely to neglect to match the case where the node being viewed matches a target endpoint, or handle these results separately.

8.4.1 Static endpoints

The simple case is to find links which have a literal endpoint which is the current node. Assuming suitable prefixes and subtype inference, we can find such links with a set of simple SPARQL queries:

1. `SELECT ?l WHERE { ?l rdf:type link . ?l w:source node:Scheme . }`
2. `SELECT ?l WHERE { ?l rdf:type link . ?l w:source node:Scheme_5 . }`
3. `SELECT ?l WHERE { ?l rdf:type link . ?l w:target node:Scheme . }`
4. `SELECT ?l WHERE { ?l rdf:type link . ?l w:target node:Scheme_5 . }`

The first two queries find links where this node is a source; the latter two, where it is a target. We must also find links from or to the specific version of the current node, which is provided by queries two and four.

8.4.2 Computed endpoints

We consider any endpoint which is not static to be 'computed'. This includes by-query, generic, parametric, and functional endpoints.

8.4.2.1 Query

Query endpoints are handled as SPARQL queries, where the union of all values of the selected variables is the set of matched pages. For example, a query endpoint of

`SELECT ?n WHERE { ?n node:Paradigm node:Functional . }` would link from or to all functional programming languages. This kind of endpoint can be tested for a specific node via a SPARQL term constraint:

```
SELECT ?n WHERE { ?n node:Paradigm node:Functional .
                  FILTER ( ?n = node:Scheme ) }
```

If multiple variables are selected, the filter should combine each with the logical or operator, so as to retrieve any logically-sound solution to the query, even if some of the variables involved are not the node we are interested in linking with.

8.4.2.2 Generic

Generic endpoints can be implemented as a filtering step on query endpoints.[2] We define a postcondition `CONTAINS ( ?n, "term" )` to filter the solutions by those where the node n contains the given term. This postcondition can be implemented efficiently by means of a lexicon cache, from each term used by any generic link, to a set of the nodes using that term. Changes to generic links add or remove items from the lexicon, and changes to any node update the sets for any terms they share with the lexicon. If `CONTAINS` is used alone, n is implied to be the universal set of nodes, so matching is a simple lexicon lookup.

To be useful for generic linking, `CONTAINS` implies an anchor at the point of the term when it is used as a source endpoint. For example, `CONTAINS ( ?n, "Scheme" )` matches the Scheme node, but should link not from the entire node, but from the text "Scheme" within it. For user interface reasons, it is desirable to restrict this only to the first occurrence of the term for non-transclusive links, so that the embedded-link document is not peppered with repeated links. For transclusive links, however, it is more consistent and useful to match all occurrences. While transclusive generic links are a slightly unusual concept, it is possible that users will find innovative applications for them. For example, if it is not possible to filter document sources at a node store level for some reason, a generic, transclusive link could be used to censor certain profane terms.

Multiple `CONTAINS` constraints can be allowed, which require that a node contains all of the terms. Any of the terms are candidates for implicit anchors: i.e. whichever occurs first will be linked, or all will be replaced by transclusion.

[2]An alternative approach may be to assert triples of the form (Scheme, containsTerm, `term`), but this would put a great load on the triplestore for each content edit.

8.4.2.3 Parametric

We can use SPARQL variables for parametric links. Every SPARQL variable is bound to the parameter element in the node's DOM tree with the same name: variables and parameters are considered to be in the same namespace. This allows the content to reflect the query result which matched this link. If the query allows `OPTIONAL` clauses which can result in unbound variables, then they could potentially have values provided by defaults from the parameter definitions in the DOM. Default values are meaningless for parameters which appear as compulsory variables in the query, as the query engine will either provide values, or will not match the link.

Parametric links may have interdependent sources and targets, in which case they are simple functional links (the source can be a function of the target, and the target an inverse function of the source). Link matching is performed pairwise for all source and target combinations. For example, consider a link with these endpoints:

```
source: SELECT ?thing WHERE { ?thing node:Colour node:Red . }
target: SELECT ?image WHERE { ?image node:Depicts ?thing . }
target: SELECT ?image WHERE { ?image node:Describes ?thing . }
```

This would create links from all nodes about things which are red, to all nodes which depict or describe or those red things. To perform this match, we union each pair of the clauses into a larger query:

```
SELECT ?thing, ?image WHERE { ?thing node:Colour node:Red .
                              ?image node:Depicts ?thing .
                              FILTER ( ?thing = node:Scheme
                                    || ?image = node:Scheme ) }
```

A similar query would also be performed for Describes. Note that we may receive values for the variables used as source or target which are not the current node if it matches in the opposite direction. We must still check that any given result for the endpoint direction we are interested in actually binds the variable to the current node. In this example, current node Scheme is not Red, so the query will not match, and no link will be created.

The pairwise matching is to be consistent with the RDF representation presented in section 7.3, and the 'or' nature of matching with static endpoints: a link must only match the current node to be used, and other endpoints may be dangling. An alternative approach would be to create a 'grand union' of all sources and targets, such that all are required to be satisfied. Neither approach is more expressive at an overall level: with a pairwise approach, a single target endpoint can include multiple `WHERE` constraints to require that all are matched; with a union approach, independent targets can be achieved through use of multiple links (although they would no longer share the same identity). The union approach is more consistent with regard to the interdependence of variables; with the pairwise approach, one matching pair of source/target endpoints may have a different variable binding for a variable of the same name to another. However, it loses the RDF and static endpoint consistency. Ultimately, the decision is whether the set of targets is a function of the set of sources (and vica-versa with the inverse), or if it is the *mapping* of a function over each source. In lieu of strong use cases for n-ary, interdependent, parametric links (most are better modelled as separate links), we choose the former for its greater consistency, and ability for a single link to provide both behaviours.

8.4.2.4 Functional

We also give consideration to arbitrarily-functional links. These are computationally expensive to match in reverse (i.e. for target-end linking and backlinks) unless the functions have inverses. We do not currently propose the ability for users to write their own Turing-complete functions, as the complexity and performance implications are widespread.

However, we can potentially provide a small library of 'safe' functions: those with guaranteed characteristics, such as prompt termination. One such example which would be of use is a 'concatenate' function:

```
source: SELECT ?n WHERE { ?n rdf:type node:ProgrammingLanguage . }
target: CONCAT( "Discuss.", ?n )
```

This would be a link from any programming language to a namespaced node for discussing it.[3] However, it highlights the reversibility problem: the inverse of `CONCAT` has multiple solutions. For example, "ABC" could have been the result of *CONCAT*("A", "BC"), *CONCAT*("AB", "C"), or a permutation with blank strings. Hence, while it is easy to match the source, and then determine the target from this, it is not practical to start with the target and determine the source.

We suggest that any endpoint which is an *arbitrary* function of others in this manner must therefore only ever be *derived.* Matching is performed against all other endpoints, and then the functional endpoints are calculated based on the results. A link from `CONCAT( ?n, ".meta")` to `CONTAINS( ?n, "lambda" )` would only ever match as a backlink: showing that any node containing 'lambda' would have been linked from its meta-node, without actually showing that link on the meta-node itself. A link with only arbitrarily functional endpoints will never match and is effectively inert.

[3]Although it would be better yet to have an explicit 'prepend namespace' function, to avoid the details of namespace syntax, and its escape if necessary.

8.5 Required modifications

8.5.1 Initial

We first seek to qualify as an open hypermedia system—to support first-class links—and then to continue to enrich the feature-set, following the priorities discovered from experiment in section 6.4.2. The 'semantic linking' components of the legacy Weerkat system should be discarded, as they are not useful toward this goal.

Our absolute first step, therefore, must be to make the link embedding and splitting changes in the preceding section. Embedding and splitting, along with native transclusion, is specific to the DOM used by the wiki to represent nodes. This DOM *is* the abstraction the system works around, thus such behaviour belongs in the core of the system. Link matching, however, is expandable depending on the link endpoint types we wish to support. Initially, static, literal endpoints will suffice, but the experiment shows that support for generic linking will be an important development. Link matching should therefore be provided by a component, which dispatches to sub-components and unions the results, as for the rendering system.

We must also modify the node store to include attributes along with the content, and add a component for attribute caching and very simple (subtype) reasoning: this can ultimately be implemented in terms of a triple store. The node store also requires changes to store nodes as DOM trees, not markup strings, and the markup parser must add an inverse transform: serialising a tree into markup. These are important because of the node structure rewriting required in the wiki core to support first-class links, and also allow for better markup independence.

Non-native transclusion can be realised as another component in the rendering chain. Edit-time transclusion requires changes to the wiki core again to insert and detect changes to the transcluded source. Otherwise, adding support should be largely straightforward once the previous groundwork on linking has been completed.

Instance property editing is purely a user interface feature. The previous changes already provide the infrastructure to handle node attributes and relational links, so the UI must use these features to identify candidate properties by domain and generate suitable form controls based on their ranges. Provided form values can simply be set as attributes of the node, rather than requiring generation of node content in the form of relational links.

In summary, this initial work amounts to modification of the wiki core, node storage interface and implementation, and markup interface and implementation; the addition of a new component type for link matching; and a new rendering component.

8.5.2 Future

Implementing additional link matching sub-components will provide the more advanced linking mechanisms, as described in section 8.4. To allow for the implicit source anchor of generic links, the link matching component will need to be able to specify modifications to the node DOM.

To grant appropriate client and user interface abstraction, rather than the current programmatic API binding, we seek to leverage a Semantic Web parallel of the World Wide Web ‘information bus’: a data bus. As this is decidedly a future goal, the design is currently still very high-level, but it should be possible to interact between the user interface and wiki system via HTTP. Our work on identifiers in chapter 7 should allow even special operations such as archived version retrieval to be performed as semantically reasonable `GET` requests with content negotiation; modification could be performed over `PUT`. The node attribute cache could be exposed as SPARQL endpoint. These two channels would then allow standard access *and modification* to the content and metadata held by the wiki. The web user interface then becomes a front-end for this data bus: to emulate a contemporary wiki, the web user interface would be configured to work (only) with the wiki system on the same server. However, we also open the possibility that some other web system may interoperate with the wiki, which should prove highly useful for the future goal of improving the wiki to work as a distributed system. It would also permit a thick-client user interface, a ‘wiki browser’, which could avoid the problems caused by having to edit wiki nodes in a serialised form due to the limitations of the simple text fields of web forms.

There are many other improvements of varying complexity which could be made to the system, but we must be careful to keep scope manageable. The modular nature of Weerkat allows many of these to be cleanly added as additional or replacement components at a later time. For example, the current conflict resolver is very crude, and should be replaced with one which automatically merges conflicting edits as much as possible, in the same way as modern version control systems.

8.6 Conclusions

In this chapter, we have shown our plan for the embedding and separating of first-class links into and from nodes in terms of the model defined in the previous chapter. An important part of this is the process of ‘link matching’, or determining if a separate, first-class link has the current node as an endpoint. We then showed that SPARQL queries, with a custom filter function operating on hashed map lookups for generic linking, could be used to perform this matching, including a pragmatic, best-effort approach to fully-functional endpoints.

The Weerkat system we have described was designed to be modular, extensible, and suited to experimental work in the field of wikis, and we feel that this will prove a useful starting point for this more advanced prototype. Hence, we have described which components of the system need to be modified and added to in order for the system to qualify as what we consider an open semantic hyperwiki.

The next chapter covers the details of our implementation of these ideas, the resulting prototype system, and the challenges we faced during the process.

Chapter 9

System implementation

9.1 Introduction

We now describe our experience implementing the model as modifications to the Weerkat system, as described in the previous chapter.

The system was developed and hosted on a 1.8GHz Celeron laptop acting as a low-power server, hosting a mod_perl and Apache environment. While performance could be sluggish, it was usable. Additional illustrations of example use of the system can be found in chapter 10.

9.2 System changes

We introduced the existing system in chapter 8: a wiki with a highly modular design, but still of a web-based hypertext design with embedded links.

9.2.1 SEML library

As link separation and embedding, and edit-time transclusion, required the system to manipulate document structure that will be presented for an editing view, we needed a mark-up parsing module which could also serialise structures back to text. Our support for multiple link types and such also required multi-value attributes, whereas the existing document tree abstraction expected only XML-style single values. Hence, we first wrote a new parser library for an improved version of the mark-up language used by Weerkat, S-Expression Mark-up Language (SEML), to support the multiple attribute values and improved escaping semantics.

The syntax is a consistent tree structure, in contrast to the chaotic mixture of symbols used by conventional wikimarkup. It is historically derived from a combination of LISP S-expressions and XML, although its current state is not a subset of either. SEML marks out elements with square brackets, selected for the relatively rare use in written prose, and to avoid confusion with XML. Immediately within an element may exist any number of attributes, followed by content, and finally an optional guard before the element closes.

As an example, `[link to=Perl type="Written in" type=Inspired\ by Pathologically Eclectic Rubbish Lister|link]` is the syntax for a link element with attributes to with a value of Perl, and type with the values Written in and Inspired by, and the content "Pathologically Eclectic Rubbish Lister". Note that we support both quoting and UNIX-style backslash-escaping: this is applied universally, and technically element names may also contain arbitrary characters. Unicode escapes are possible via Perl-style escapes, by name, octal, or hexadecimal value. Attributes are separated from content by the first whitespace-separated token which is not a value attribute, or this distinction can be made explicitly with an unescaped pipe character, which allows allows the element content to contain leading whitespace. Finally, we have demonstrated here the optional 'guard', which is intended to offer XML's advantage of explicit closing tags to help ensure correctly matched element start and end points, without the verbosity of forcing its use in every case. As this example does not have complicated content with nested elements, its use here is for demonstrative purposes only.

The SEML library also supports autoparagraphing, or implicit para elements, through the use of LaTeX or classic wiki markup-style blank line separation. It also makes the same inline ('text-level') vs. block-level element distinction as HTML: only text-level elements can exist within an paragraph, whereas block-level elements split (automatic) paragraphs. This helps to build a document tree which is reasonably compatible with the XHTML DOM.

SEML uses a hand-crafted lexer and parser, which can be found in the Weerkat source. Appendix D shows the state diagram. Originally, SEML was to include namespacing, but this was removed soon after integration with Open Weerkat as it introduced considerable complexity for no concrete gains.

9.2.2 DOM-oriented refactoring

The first change to Weerkat itself was one of the most extensive: to re-factor the system so that the content of node revisions was a DOM tree, not a string of markup. Classic Weerkat already used a DOM abstraction, so as to be markup-agnostic, but node storage was, like conventional wikis, in text markup form. Because its HTML-style embedded links were dealt with during the rendering step, parsing wasn't performed until relatively late into handling a request, with the wiki core operating in terms of raw markup text.

9.2.2.1 Storage

First the Revision class, which represents a single revision of a node, was changed to contain an AnnotatedTree instead of a string as its document content. The Flatfile implementation of the NodeStore abstract base class was largely rewritten to deal with this change. As a result, we have now pushed the system to the other extreme: it does not deal with markup at all except for as needed format for editing. Apart from theoretically allowing truly interchangable markup languages on the same document body, this also changes performance characteristics of the wiki to optimise toward readers, rather than writers. Caching aside (which we currently do not implement, although it is low-hanging fruit for future optimisation), we avoid a parsing stage for every read, whereas most conventional wikis effectively perform multiple substitution passes over the document, transforming it into the required output without an intermediate representation, using regular expressions. We trade this for a serialisation stage to user-facing markup for every edit, which is a cheaper operation, and a one-off parse when the change is submitted.

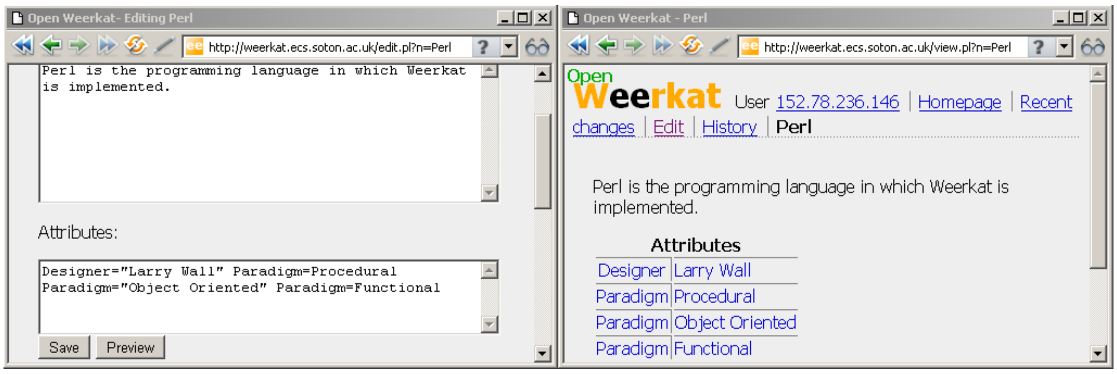
We also improved the storage layout so that arbitrary page titles are supported without lossage through an URI-encoding mechanism. Nodes exist as directories, under which lie revisions as files. The URI-encoding is deliberately slightly overeager so as to reserve a set of valid filenames which node titles will never use.

Node-granularity locks are implemented using the POSIX `O_EXCL` mechanism to safely serialise concurrent access: it guarantees atomicity of lockfile creation, although pending processes are forced to busy-wait. The store passes back an object which captures the context required to later clean up the lockfile upon its destruction, following the RAII (Resource Allocation Is Initialisation) paradigm: this is possible because Perl's reference-counting garbage collector still allows for deterministic object destruction. The core avoids deadlock by always claiming and releasing locks in an ordered fashion: node titles provide a straightforward strict total ordering.

We felt that the use of an full relational database was unnecessary, as the only index the filestore requires is (*node*, *revision*) to a binary blob (the serialised Revision object, which itself subdivides into the document content and attributes), and modern filesystems are designed for efficient name to datastream mappings.

9.2.2.2 Processing

The MarkupParser implementation was updated to use the new SEML library, and the interface updated to support serialisation. To allow the SEML library to preserve some


Open Weerkat- Editing Perl
http://weerkat.ecs.soton.ac.uk/edit.pl?n=Perl
Attributes:
Save
Preview
Open Weerkat - Perl
http://weerkat.ecs.soton.ac.uk/view.pl?n=Perl
Open Weerkat
User 152.78.236.146 | Homepage | Recent changes | Edit | History | Perl

```
Perl is the programming language in which Weerkat
is implemented.
```

```
Designer="Larry Wall" Paradigm=Procedural
Paradigm="Object Oriented" Paradigm=Functional
```

Perl is the programming language in which Weerkat is implemented.

Attributes	
Designer	Larry Wall
Paradigm	Procedural
Paradigm	Object Oriented
Paradigm	Functional

FIGURE 9.1: SEML-based attribute editing, and tabular display

of the syntactic choices made in writing the node source, document trees are allowed to contain markup-specific presentation hints. These are a largely unavoidable wrinkle in the ideal of interchangable markup if we are to avoid entirely normalising node sources, and instead allow users to maintain the flexibility of choosing between representations such as `"Semantic Web"` vs. `Semantic\ Web`.

We also renewed the element types we support to allow for the changes to links, and removal of Classic Weerkat's limited semantic features. Currently we provide a small set of useful document elements: paragraphs, headings, emphasis, and such, which loosely follow XHTML. The final set is presented in appendix E. The MarkupParser is now responsible for ensuring that the parsed DOM contains only valid, defined elements, whereas previously the markup-centric core left the task of 'cleaning' the output to the renderer.

Once the XHTMLFinal renderer (so named as it is intended to be the last item in a potential chain of renderers) was updated to allow for this new set, we were again able to view and edit pages to test the success of our refactoring efforts.

9.2.3 Node attributes

9.2.3.1 Representation

Because the original system operated largely within XHTML assumptions, the AnnotatedTree class used for internal DOM representation only supported XML-style single-value attributes. However, our model required multiple values, so that a node which is a link may have many target endpoints. Therefore we modified our implementation and expanded the API accordingly; the legacy, single-value methods operate on the first value only.

We expanded the WebCGI user interface to have a very simple SEML-based attribute editor. The attributes of the node were serialised as if they were the attributes of an element, and likewise parsed back. While this was originally intended as a stopgap for debugging, it was later employed as the 'Other' box to allow for attributes unknown to instance property editing. We also added a view-time table to display a node's attributes, which were hyperlinked by the UI code to the node of the same name. Figure 9.1 shows the attribute editing box below the main page content in the left window (which we have here temporarily reduced in height), and the display table in the right. Note that the attribute Paradigm has been given multiple values.

For storing the attributes in the Flatfile implementation, we used the Redland RDF[1] library's 'hashes' backend, with BerkeleyDB persistence. The lightweight database files are stored using some of the aformentioned reserved names in the file organisation. This also provided us with an internal SPARQL implementation, which would be required for link matching; a query() method was added to the NodeStore interface to access it.

9.2.3.2 Versioning

The version separator we use for our URIs is the semicolon. According to RFC3986 [78] (and particularly clear in previous RFC2396 [79]), this is the parameter separator, and shouldn't have any unwanted effects on URI equivalence, relative paths, or such. We are effectively adding some extra information to the path element, which is a reasonable description of a version field. Because ';' is reserved for this purpose, it is URI escaped if in a title. This behaviour is also consistent with ISO9660's handling of file versions.

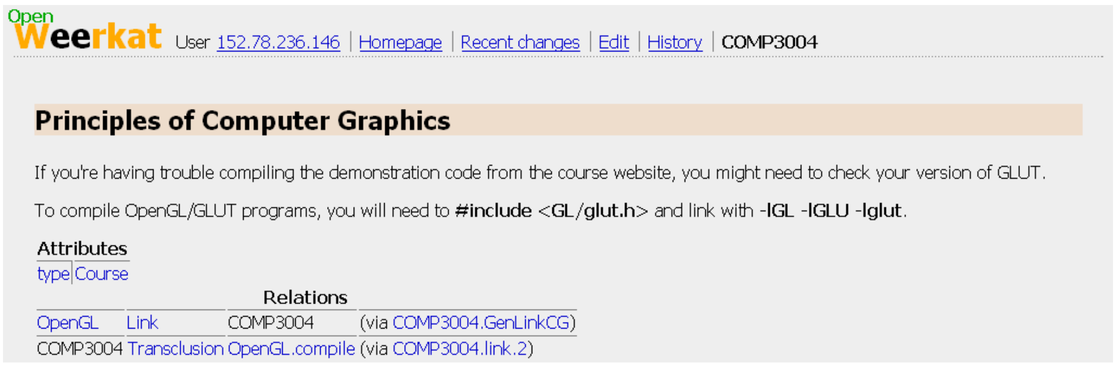
9.2.4 Link matching

We added the LinkMatcher base class, as described in the plan in section 8.4, and a Static implementation which matched literal endpoints by performing the queries for source and target attributes.

9.2.4.1 Visualisation

To debug the matches, we dumped them to a table the web view, which eventually evolved into a table of immediate node relations. This has a column for the three RDF components, and either the object or subject will be the current node (the actual information provided to the user interface is in the form of directional typed arcs between the current node and another). A fourth column, added later, names the link responsible for the arc. While this was never a formally planned aspect of the system, it was left in due to its use in development, and proved to be particularly appreciated by users in our experiments in chapter 11.

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


Open Weerkat User 152.78.236.146 | Homepage | Recent changes | Edit | History | COMP3004

Principles of Computer Graphics

If you're having trouble compiling the demonstration code from the course website, you might need to check your version of GLUT.

To compile OpenGL/GLUT programs, you will need to **#include <GL/glut.h>** and link with **-lGL -lGLU -lglut**.

Attributes

type	Course

Relations

OpenGL	Link	COMP3004	(via COMP3004.GenLinkCG)
COMP3004	Transclusion	OpenGL.compile	(via COMP3004.link.2)

FIGURE 9.2: Prototype relations table

Figure 9.2 shows the final relations table for an example node from section 10.2. The first line shows that the link COMP3004.GenLinkCG is responsible for the OpenGL node linking to this page, COMP3004. The second shows an outgoing transclusive link from this page to OpenGL.compile, a fragment of shared content.

9.2.4.2 Versioning

The system does not yet match versioned links, due to versioned predicates: the statement (MyLink_1, type_1, Link_1) is not detected by the matcher because it currently does not allow for the versioned variant of the type or Link URIs. This is because we currently work upon the internal URIs, rather than the external RDF representation defined in section 7.3. Similarly, the source and target predicates are not recognized in their versioned forms.

There are two potential short-term workarounds for this issue; for the requirements of our prototype and evaluation, we have not yet pursued either of these.

Unversioned predicates and duck-typed links. Do not add a version specifier to the predicate of an attribute nor require that a node is explicitly of Link type to act as a link. At this point, we had not assessed the wider implications this may have on the system.

Unversioned known system URIs. Do not use versioned URIs for the resources special to the wiki, such as source and Link. This has the problem of making such URIs special, exceptional cases, complicating the uniformity of the system.

9.2.5 First-class linking and transclusion

9.2.5.1 Link embedding

We added a new retrieve() method to the wiki core for use by the view() and edit() methods, which the user interface uses to get a rendered and source representation of a node respectively. Previously these methods retrieved the node source/tree directly from the node store. The retrieve method performs link embedding, as defined in section 8.3. We first changed view() to use the new retrieval routine, which allowed us to test it with a manually-created first-class link. Once edit() was changed over as well, this link appeared in the node source; this change was also needed for edit-time tranclusion.

Algorithm 2 shows the *final* algorithm, with support for transclusion and generic linking. We cover the processTreeWords method in section 9.2.6.2, and the query methods in section 9.2.5.3. In general, the procedure is to walk the document tree, replacing anchor elements with link elements (or the functionally identical trans) containing the target content, and with attributes derived from the link arcs.

9.2.5.2 Link separation

The original modify() method, which the user interface calls to submit changed source back to the system for conflict checking and storage, was written with the standard wiki expectation that the change would only involve a single node. Link and transclusion embedding breaks this expectation. Hence, we first modified the method to first batch up a set of changes (initially consisting only of the node submitted), acquire locks on each in order (to avoid deadlock), then store (commit) them all as one. Should any change in the set trigger a conflict, no storage is performed, and the modify routine returns the conflict information as before. This is not strictly atomic, as there is no way for a failed store() to roll back any previous stores; however, storage failures are at least bug/fault conditions, not possible rejections. We soon after resolved this with a new storemulti() method in the NodeStore base class, which uses a non-atomic implementation in terms of store() by default, but can be overridden for storage backends capable of transactions.

For our prototype, we have not updated the ConflictResolver class to be aware of multiple-change edits. This means that the system's handling of edit conflicts is currently suboptimal, as it ignores the ability to localise conflict problems to specific links or transcluded sections.

Algorithm 3 shows the *final* modify routine, with transclusion and generic linking support. We focus on the separation algorithm, and pass over the details of conflict detection and actual storage management. At a high level, the method walks the element tree, converting link and trans elements into anchors, while preserving their content and links

Algorithm 2 Final core retrieve method

```
function RETRIEVE(title, reduce) ▷ reduce: generate only static link attributes, for rendering; else preserve queries, for editing
    revision ← nodestore.RETRIEVE(title)    ▷ Fetch from storage
    if revision = NULL then return NULL    ▷ No such revision
    procedure MATCHTOATTRS(elem, match)    ▷ Add attributes to element based on match, also build arcs
        role, linknode ← details of match
        link ← nodestore.RETRIEVE(linknode)
        outmatches ← linkmatcher.MATCHOUTWARD(inverse role, link)
        for all outmatches do add each type, target pair to arcs for role
        if elem = NULL then return    ▷ Building arcs only
        Set id on elem from linknode
        Copy types to elem from link
        if reduce then Set inverse attribute of role (to/from) from outmatches
        else Copy to, from, toq and fromq from link, translating from source, etc.
    procedure EMBED(tree)    ▷ Embed transcluded content, natively and otherwise
        if tree is an anchor element then
            anchorid ← tree.attribute[id]
            Convert tree to empty link element
            anchor, matches, arcs ← RETRIEVE(title+anchor+anchorid)
            for all matches match do
                if nodestore.QUERYISA(match node, Transclusion) then
                    link ← nodestore.RETRIEVE(match node)
                    elem ← RETRIEVE(link target)    ▷ Recursive transclusion fetch
                    Convert elem to trans element
                    Add elem to document as sibling of tree
                else elem ← tree
                MATCHTOATTRS(elem, match)    ▷ Add attributes to selected element
                Remove self from from attributes
                elem.attribute[aid] ← anchorid    ▷ Preserve anchor ID
            if All matches were transclusive then Remove tree from the document
            else Copy anchor children into tree
        for all child nodes do EMBED(child)    ▷ Recurse down tree
    EMBED(revision.content)    ▷ Embed the revision fetched
    matches ← linkmatcher.MATCH(title)
    for all matches match do
        MATCHTOATTRS(NULL, match)    ▷ Add arcs from node-level matches
        if match has text then    ▷ Is it a generic link?
            PROCESSTREEWORDS(revision.content) lambda(word)
                if word = match text and first occurrence then
                    elem ← new link element
                    MATCHTOATTRS(elem, match)
                    Add word as child of elem; set aid attribute to _
                    return (replace word with) elem
    Filter arc names and remove duplicates
    return revision, matches, in and outbound arcs
```

Algorithm 3 Final core modify method

```
function MODIFY($title$, $newrevision$)
    Parse $newrevision.content$ using $markupparser$
    $changes[title] \leftarrow newrevision$                          ▷ Title-to-content map of changes
    procedure SEPARATE($tree$, $septitle$)                         ▷ $septitle$ used to generate IDs
        if $tree$ is a link element then $aid \leftarrow tree$.attribute[aid] or find new free ID
        if $aid$ is _ (implicit generic link) then
            Remove $tree$ from document, promoting children to siblings
        if $aid$ is valid then
            $oldlink \leftarrow nodestore$.RETRIEVE($tree.attribute$[id])
            $linkname \leftarrow$ find new free ID unless id saved
            Prepare new node with $oldlink$'s content
            $types \leftarrow tree$.attributes.[type]
            Copy $tree$ to attributes etc. to new node target attributes; also $types$
            if not $nodestore$.QUERYSUBCLASSOF($types$, Link) then Add Link type
            $changes[linkname] \leftarrow$ new link node
            $istrans \leftarrow nodestore$.QUERYSUBCLASSOF($types$, Transclusion)
            if $istrans$ then $bodyname \leftarrow$ target attribute
            else $bodyname \leftarrow$ anchor name
            if not ($istrans$ and $tree$ empty of content) then
                Prepare new node with old anchor's attributes, if any
                Copy $tree$ children into new anchor content
                SEPARATE(each child, $bodyname$)                   ▷ Separate nested anchors
                Add new node to $changes$
            if $istrans$ and anchor does not exist then
                Add empty anchor to $changes$
            Change $tree$ element into an anchor; save id
        for all child nodes do SEPARATE(child, $septitle$)         ▷ Recurse down tree
    SEPARATE($newrevision.content$, $title$)
    for all $changes$ $change$ do Check user has permission to make $change$
    for all SORT($changes$) $change$ do Claim NodeStore lock for $change$
    Check for conflicts and unlock/return resolution if any
    $nodestore$.STOREMULTI($changes$); unlock; return SUCCESS
```

attributes as further changes to other nodes. Note that we keep the *content* of link elements, but replace their attributes with those of the embedded link, while conversely preserving the attributes of anchor and transcluded nodes, replacing their content with that which was embedded. Line 24 deals with the requirement that for a link to match, its endpoints must exist, and a new and embedded transclusive link would otherwise not create its anchor in that document, as the transclusion code instead redirects changes to its target.

With the old embedded linking support removed from the XHTMLFinal renderer, we now had a system with working first-class links.

9.2.5.3 Subclass inference and transclusion

Because our model uses Transclusion as a subtype of Link, we needed very simple subClassOf inference. Technically, we use wk:subClassOf, due to the implicit namespace of all node names in the system, but this is considered to be the same as its RDFS namesake.

We first built a simple abstract representation of SPARQL queries, and updated the LinkMatcher and NodeStore to use it, so that queries could be programatically constructed and manipulated rather than using string operations. In future, this should prove useful when combined with a suitable parser for handling our extensions such as CONTAINS in combination with arbitrary queries, as described in section 8.4.2.2.

Following this, we added queryIsA() and querySubClassOf() methods to the NodeStore, which test if an instance and a class are of a given class, respectively. They have base implementations which work in terms of simple queries which do not require the underlying triplestore to perform inference. As shown in pseudocode in algorithm 4, the former is implemented in terms of the latter, which is effectively a breadth-first search.

Algorithm 4 NodeStore base Is-A and SubClass-Of query

```
    function queryIsA(resource, class)                    ▷ Is resource a class?
        types ← map                                       ▷ Find explicit types
            lambda(binding) binding[t]                    ▷ Unwrap query result
            Query(resource, type, ?t)
5:      return querySubClassOf(types, class)

    function querySubClassOf(types, class)    ▷ Is one of types a subclass of class?
        while types non-empty do
            if types contains class then
                return true
5:          types ← map                                   ▷ Find immediate subtypes
                lambda(type) map
                    lambda(binding) binding[t]            ▷ Unwrap again
                    Query(type, subClassOf, ?t)
                types
10:     return false                      ▷ Ran out of types: cannot be of class
```

It was at this point that the link responsible for an arc matching was reported by the core node retrieval and view methods for debugging, which added the 'via' column to the relations table. With this, we were able to determine that Link subtypes were now matching correctly as links. Transclusion followed quite simply due to previous separation/embed work in retrieve() and modify().

9.2.6 Generic linking

9.2.6.1 Word dictionary

To implement generic linking, we began by implementing a simple general purpose Dictionary class in terms of tied Perl hashes: a simple persistent word-to-object mapping, again backed by BerkeleyDB and POSIX file locks. We extended the NodeStore API again to include a searchWord() method which returns the set of nodes containing a given word, and provided a Flatfile implementation using the new dictionary class.

This does come at a cost: initially the Flatfile would, upon storage, first scrub the dictionary of words from the previous version of the node, affecting speed. Later we changed this to store versioned node URIs, instead slowly growing the dictionary as a storage cost, although currently computational scalability of multiple dictionary values for the same key is linear. Theoretically, this permits versioned searching to be implemented at a later date.

9.2.6.2 Tree text map

To populate the dictionary with words when nodes are saved, we wrote a utility function to walk document trees and perform operations upon them: processTreeWords. Strictly, we call a provided closure upon each word, and the return value of the closure can be used to specify a replacement, which will be useful later. If the replacement is in the middle of a DOM text node, it is split and the preceding and following text become siblings of the replacement. Functional programmers may recognise this as being a specialised form of the map function of some languages, which operates on lists.

9.2.6.3 Generic LinkMatcher

Until this point, the embedding and separation code was not strictly using the link matcher fully: upon a match, the routines would directly use the to attribute of the link. To rectify this, we added a matchOutward() method, which asks the LinkMatcher implementation "given this link, where does it point?" As with match(), this takes an arc role bitmask, and matches include direction information, so it can also return the link's source endpoints. In addition, link matches were expanded to specify the text they matched upon, if generic.

For efficient matching of generic links, we wanted to create indexes from words to links which involve them. There is one Dictionary for words used in source arcs, and one for target arcs. For example, a generic link named ExampleLink with a sourceQuery of `CONTAINS('Perl')` would be present in the source-role index as a mapping from `Perl` to ExampleLink.

To create these, we added a postStoreHook() method to the LinkMatcher class, which the Generic implementation uses to spot `CONTAINS` query endpoints and add them to the appropriate index. This does unfortunately affect one of the properties of the Weerkat system: until this point, it was largely possible to change the components loaded on a system at any time. The requirement to build an index upon storage hooks, however, currently requires that for the generic link matcher to function correctly, it must have been in use continually for the whole lifespan of the wiki content. This problem is not insurmountable, and could be resolved by performing a full index rebuild when the generic link matcher is (re)added to the configuration.

When the wiki core queries the generic link matcher for links for a document, it must first fetch it (conversely, the static link matcher works entirely on the RDF metadata), and then use the text tree map routine to check each word against the word-to-link index for the queried endpoint role. This means that the per-display cost of generic links is proportional to the length of the document, but this optimisation only works for links whose queries are purely `CONTAINS`. Fully functional links would have to each be tested, causing the system to scale with the number of links with query endpoints. Outward matching scales with the number of query endpoints on the link being considered, as it is simply a searchWord() call on the node store to look up nodes containing the word queried for.

To place links in the document where generic links match, we needed to create temporary anchors to work as link endpoints. This was facilitated by our earlier text tree manipulation routine (section 9.2.6.2), by providing it with a closure which replaced the first instance of the word with a new sub-tree representing a link anchor, and from then on acted as a no-op.

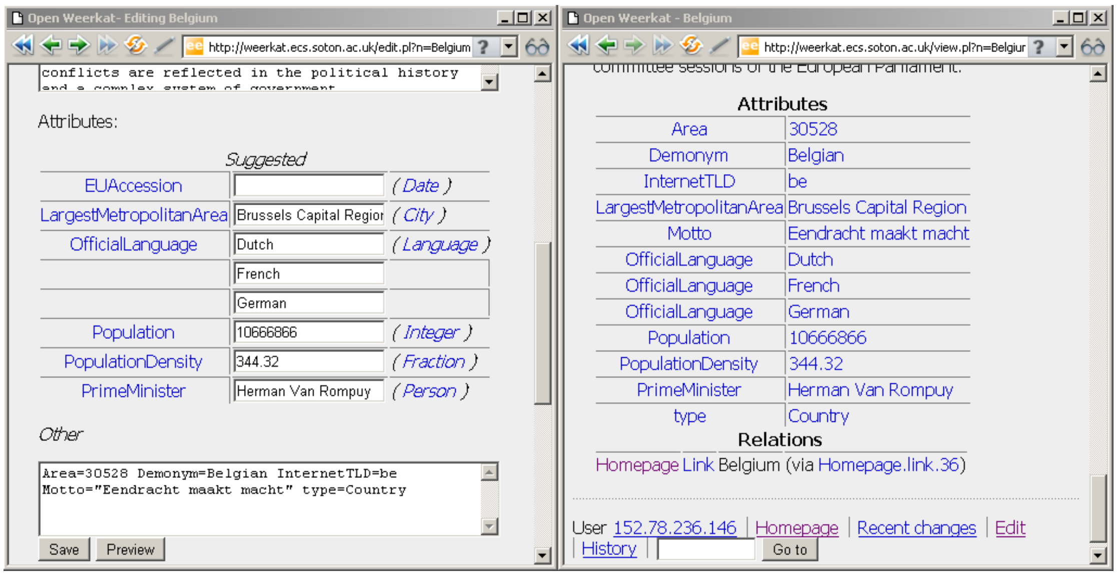
We gave these anchors an identifier of '_', to allow link separation algorithm know that they are implicit, and not to store them back into the document. This is currently quite aggressive, and the system ignores all changes to generic links via their endpoints. Once done, generic links began to function in the system.

Once we had updated the retrieve() and modify() algorithms to use matchOutward(), generic links also functioned backwards.

9.2.7 Instance property editing

To implement instance property editing, we first defined the information that the user interface required to provide an appropriate form; given a node of type T:

- The suggested predicates of that type, i.e. (?*predicate*, domain, T).
- The range of each predicate P, i.e. (P, range, ?*range*).


Open Weerkat- Editing Belgium

http://weerkat.ecs.soton.ac.uk/edit.pl?n=Belgium

```
conflicts are reflected in the political history
```

Attributes:

Suggested

EUAccession		(*Date*)
LargestMetropolitanArea	Brussels Capital Regior	(*City*)
OfficialLanguage	Dutch	(*Language*)
	French	
	German	
Population	10666866	(*Integer*)
PopulationDensity	344.32	(*Fraction*)
PrimeMinister	Herman Van Rompuy	(*Person*)

Other

```
Area=30528 Demonym=Belgian InternetTLD=be
Motto="Eendracht maakt macht" type=Country
```

Open Weerkat - Belgium

http://weerkat.ecs.soton.ac.uk/view.pl?n=Belgiur

Attributes

Area	30528
Demonym	Belgian
InternetTLD	be
LargestMetropolitanArea	Brussels Capital Region
Motto	Eendracht maakt macht
OfficialLanguage	Dutch
OfficialLanguage	French
OfficialLanguage	German
Population	10666866
PopulationDensity	344.32
PrimeMinister	Herman Van Rompuy
type	Country

Relations

Homepage Link Belgium (via Homepage.link.36)

User 152.78.236.146 | Homepage | Recent changes | Edit | History |

Figure 9.3: Prototype instance property editing, and matching view

We added a properties() method to the Wiki core which would provide this information for a given node. (Note that our simple queryIsA() subtype inferencing is unsuitable to use here, so subclasses will only work if the underlying triplestore provides inference. We could resolve this by exposing and using the internal entire class set involved as part of the queryIsA() algorithm, at a performance penalty.)

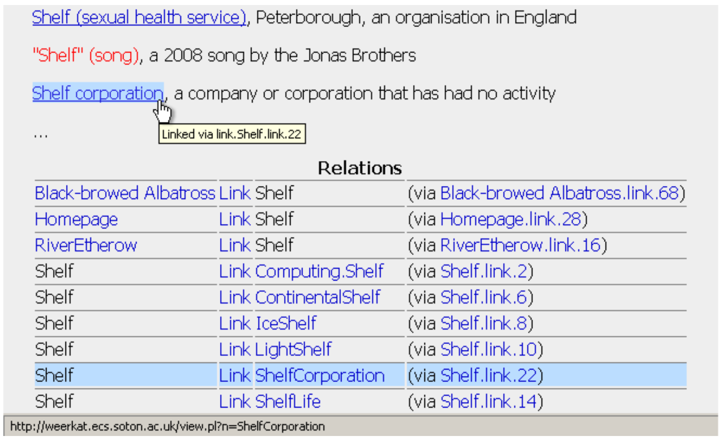
We updated the editing user interface page to use this method, and generate a table of form fields as a result. Attributes are mapped into this form by preference, and remainders use the existing SEML-serialised 'Other' box. Upon form submission, the two sets are merged: there are no conflicts because we allow multiple values.

Figure 9.3 shows this using a reduced form of the attributes for Belgium. Note that the page view, right, contains a mixture of the 'suggested' and 'other' attributes in the editing display, left: the distinction exists only when editing.

9.2.8 Usability tweaks

9.2.8.1 Cross-link highlighting

The relations table contains a 'via' column which names the link responsible for the relation, and link anchors in the document have a title XHTML attribute which does the same, which appears as a tooltip in most browsers. To strengthen the reporting of this relation, we added a JavaScript-based cross-highlighting cue. The XHTMLFinal renderer adds URI-encoded, namespaced CSS classes to each XHTML (embedded) link encoding the Weerkat (first-class) link responsible. Mouse enter/exit events which call highlight/dehighlight functions add and remove a 'highlight' CSS class to all document



Shelf (sexual health service), Peterborough, an organisation in England

"Shelf" (song), a 2008 song by the Jonas Brothers

Shelf corporation, a company or corporation that has had no activity

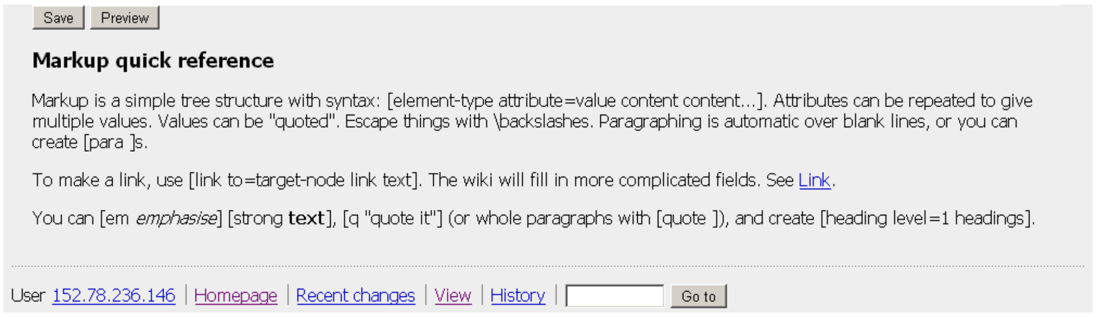
...

Relations

Black-browed Albatross	Link Shelf	(via Black-browed Albatross.link.68)
Homepage	Link Shelf	(via Homepage.link.28)
RiverEtherow	Link Shelf	(via RiverEtherow.link.16)
Shelf	Link Computing.Shelf	(via Shelf.link.2)
Shelf	Link ContinentalShelf	(via Shelf.link.6)
Shelf	Link IceShelf	(via Shelf.link.8)
Shelf	Link LightShelf	(via Shelf.link.10)
Shelf	Link ShelfCorporation	(via Shelf.link.22)
Shelf	Link ShelfLife	(via Shelf.link.14)

http://weerkat.ecs.soton.ac.uk/view.pl?n=ShelfCorporation

FIGURE 9.4: Prototype link cross-highlighting



Markup quick reference

Markup is a simple tree structure with syntax: [element-type attribute=value content content...]. Attributes can be repeated to give multiple values. Values can be "quoted". Escape things with \backslashes. Paragraphing is automatic over blank lines, or you can create [para]s.

To make a link, use [link to=target-node link text]. The wiki will fill in more complicated fields. See Link.

You can [em *emphasise*] [strong **text**], [q "quote it"] (or whole paragraphs with [quote]), and create [heading level=1 headings].

User 152.78.236.146 | Homepage | Recent changes | View | History | Go to

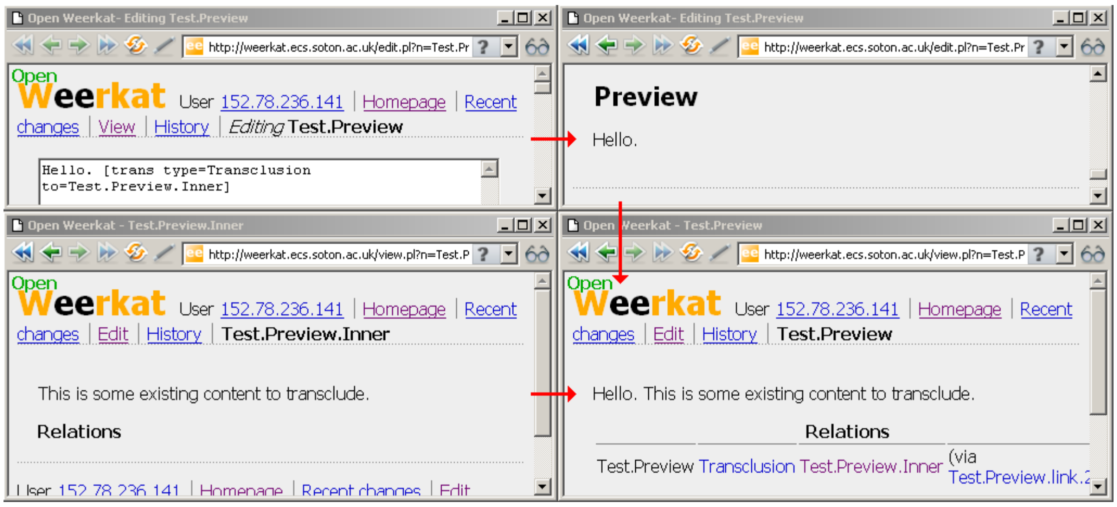
FIGURE 9.5: Markup quick-reference at the bottom of an edit page

elements with the given link class respectively. This is shown in figure 9.4, along with the link tooltip and browser status line. The highlighting appears the same should the user instead hover over the row in the relations table.

9.2.8.2 Arc filtering

The final evolutionary step of the relations table was to filter and sort the arcs which it displays. We remove the anchor-specific part of the node identifier for source arcs, to better emulate our intended RDF model, and make backlinks more readable: this changes 'Phil.anchor.1 Link Perl' to 'Phil Link Perl'.

The embedding process which generates arcs is subject to creating duplicates due to its recursive nature and the way it derives arcs both from links and node-level attributes. Hence, we filter these out, as they do not convey any useful information. The user interface also applies by-role and alphabetical sorting to the table.


Open Weerkat- Editing Test.Preview
http://weerkat.ecs.soton.ac.uk/edit.pl?n=Test.Pr
Open Weerkat
User 152.78.236.141 | Homepage | Recent changes | View | History | *Editing* Test.Preview
Preview
Open Weerkat - Test.Preview.Inner
http://weerkat.ecs.soton.ac.uk/view.pl?n=Test.P
User 152.78.236.141 | Homepage | Recent changes | Edit | History | Test.Preview.Inner
Relations
User 152.78.236.141 | Homepage | Recent changes | Edit
Open Weerkat - Test.Preview
User 152.78.236.141 | Homepage | Recent changes | Edit | History | Test.Preview
Relations

```
Hello. [trans type=Transclusion
to=Test.Preview.Inner]
```

Hello.

This is some existing content to transclude.

Hello. This is some existing content to transclude.

Test.Preview Transclusion Test.Preview.Inner (via Test.Preview.link.2

FIGURE 9.6: Demonstration of preview limitations in Open Weerkat

9.2.8.3 Markup quick-reference

Weerkat features a small paragraph below the editing controls which summarises the syntax and vocabulary needed to write wiki nodes. Given the importance of learning by example identified in our experiment in chapter 11, we updated this to reflect the revised SEML syntax, new element set, and new capabilities of Open Weerkat. It conveys a core subset of the markup, and is shown in figure 9.5.

9.2.8.4 Base types

Prior to our experiment setup, and of general use to any Open Weerkat install, we populated the wiki with node content for the system types, such as Link. These nodes can be found in appendix F.

9.3 Technical challenges

9.3.1 Preview

Due to time constraints, the current preview mechanism is largely unchanged from the original wiki design. The user interface takes the submitted page source, and manually runs it through the markup parser and rendering pipeline. This is no longer sufficient for two reasons: first, pages need to go through the separation and embedding stages to be correctly transformed for rendering. The preview partially works as-is because the previously embedded content is preserved unchanged, but new embeds such as in-line transclusion will not take effect. Second, edits are no longer guaranteed to only affect the node being edited.

The first problem is demonstrated in figure 9.6. The new page, top-left, is written to transclude some content from an existing page, bottom-left. However, this text does not appear in the preview, top-right, since the preview does not run through the full embedding and separation process. After saving, the view then performs all the correct transformations and the page appears as intended, bottom-right.

This latter problem can quickly become very complicated when the user has made changes within in-line transclusions: it is necessary to overlay an entire transient changeset over the current node store state for the embed/separate routines to work with. It may be wise to postpone working on this issue until versioning-related future work has been considered, as better changeset support would help immensely here, effectively treating the preview as a private branch.

There are also implementation quirks with preview and instance property editing, because the preview stage does not transfer the whole type information across, and the user interface cannot query the Wiki core for it again, as it may have changed. This manifests as a harmless loss of range type hints, and could be solved by preserving the state in hidden form fields.

9.3.2 Limits of XHTML and linear markup

An obvious limitation of XHTML affecting the system is that XHTML links can only have one target endpoint. It is impossible to represent fat or nested links, and they can only be made navigable by laying application-specific user interface on top. Even without XHTML serialisation, we also do not support overlapping links at all due to the tree structure of the document representation.

Linear serialisations of the model, such as XHTML or editable text markup, have difficulty gracefully handling multiple separate links matching on the same endpoint. To avoid duplicating the anchor or transcluded content, and thus both significant confusion and the risk of conflicts with the edit itself, all links must be merged into a single element. It is not acceptable to nest them, as this is ambiguous with the case where an anchor or transcluded node itself contains a link spanning the entire document. Hence, it is necessary to keep separate within the same link element the type and endpoint information for more than one link. The current syntax does not permit this, and allowing it would add significant additional user-facing complexity while using text markup for editing.

Multiple source anchors matching the same anchor of mixed transclusive and non-transclusive types are also a problem, if somewhat of a pathological case.

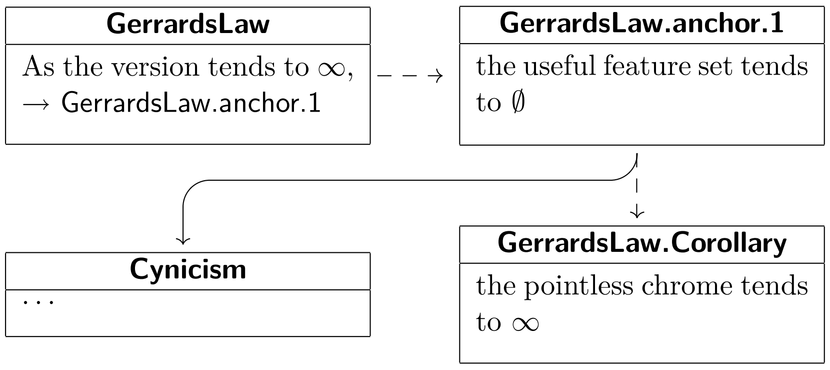

GerrardsLaw
As the version tends to ∞,
→ GerrardsLaw.anchor.1
GerrardsLaw.anchor.1
the useful feature set tends to ∅
Cynicism
· · ·
GerrardsLaw.Corollary
the pointless chrome tends to ∞

FIGURE 9.7: Example of regular and transclusive link sharing source anchor

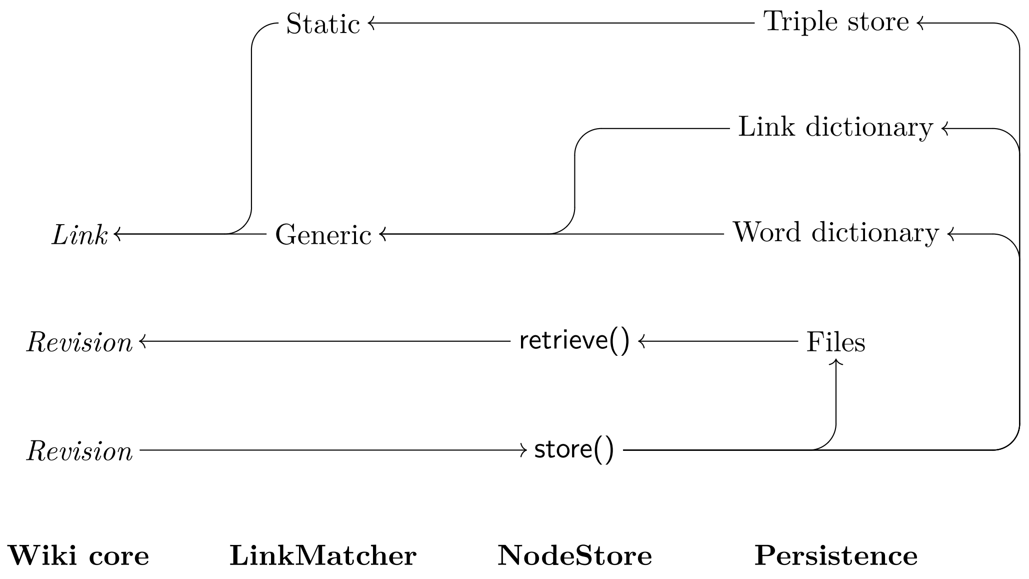


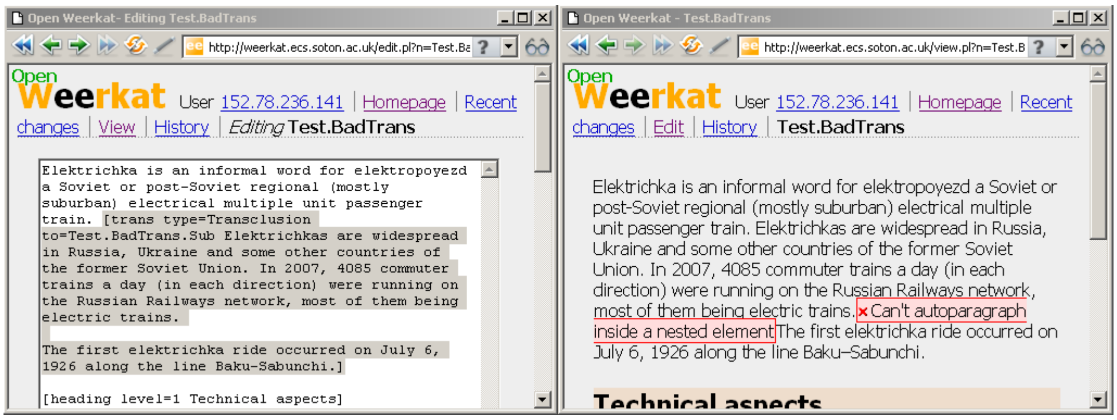
FIGURE 9.8: Data flow complexity introduced by Generic linking

Consider figure 9.7, in which the anchor in GerrardsLaw is linked non-transclusively to Cynicism, and transclusively to GerrardsLaw.Corollary. (For conciseness, we do not show the intermediate first-class links in this diagram; the anchor is, of course, transcluded natively.) There is no correct way to display this document as a linear sequence: if displaying 'featureset' clause, the transclusion has not been correctly represented; if displaying the 'chrome' clause, the regular link has not.

The conclusion we would draw from this is that links should not be able to share source anchors, at least if of mixed transclusive type; while this constraint is specific to an endpoint's role, the same can be said of navigability and content replacement. This does not prevent fat links, as we still allow multiple targets on the same link.

9.3.3 Generic linking

Generic linking, and functional links in general, add a lot of complexity to the system, and this complexity is not easily localised into a single module. Support for it is required



Open Weerkat- Editing Test.BadTrans
http://weerkat.ecs.soton.ac.uk/edit.pl?n=Test.Ba
Open Weerkat User 152.78.236.141 | Homepage | Recent changes | View | History | *Editing* Test.BadTrans

```
Elektrichka is an informal word for elektropoyezd
a Soviet or post-Soviet regional (mostly
suburban) electrical multiple unit passenger
train. [trans type=Transclusion
to=Test.BadTrans.Sub Elektrichkas are widespread
in Russia, Ukraine and some other countries of
the former Soviet Union. In 2007, 4085 commuter
trains a day (in each direction) were running on
the Russian Railways network, most of them being
electric trains.

The first elektrichka ride occurred on July 6,
1926 along the line Baku-Sabunchi.]

[heading level=1 Technical aspects]
```

Open Weerkat - Test.BadTrans
http://weerkat.ecs.soton.ac.uk/view.pl?n=Test.B
Open Weerkat User 152.78.236.141 | Homepage | Recent changes | Edit | History | Test.BadTrans

Elektrichka is an informal word for elektropoyezd a Soviet or post-Soviet regional (mostly suburban) electrical multiple unit passenger train. Elektrichkas are widespread in Russia, Ukraine and some other countries of the former Soviet Union. In 2007, 4085 commuter trains a day (in each direction) were running on the Russian Railways network, most of them being electric trains. ✗ Can't autoparagraph inside a nested element The first elektrichka ride occurred on July 6, 1926 along the line Baku–Sabunchi.

Technical aspects

FIGURE 9.9: Attempted transclusion of a region spanning tree branches

in the core embed/separate methods, to handle implicit anchors; in the storage layer, to maintain a word dictionary (although this is a requirement shared with full-text search), and to maintain indexes for efficient lookup of relevant links (implemented as a hook in the LinkMatcher); in the link match type, to preserve the information of which content matched upon; and finally the actual link matching implementation itself. Figure 9.8 shows how the generic link matcher fits into the system data flow for storing, retrieving, and matching nodes and links. If the system is to support generic linking at all, the maintenance cost of its complexity must be paid even when not working on the relevant LinkMatcher, and likewise the performance costs are paid without a global configuration flag to disable them. As noted in section 9.2.6.3, the generic link matcher can also not be easily 'hotplugged' into a wiki without a setup stage to build initial indexes.

Implementing generic links has raised considerable scalability concerns. Generic links use a word-to-link index to aid scalability. Arbitrary functional links cannot use this, beyond a simple set listing which links are functional: every link in this set must be tested for matches upon every retrieval and modification. This problem is not easily resolved with view caching, and in fact the existence of functional links greatly affects the usefulness of caching in general: cache validity is affected should either a node change, or a link affecting that node changes. Functional links greatly complicate the task of determining which node caches should be invalidated when the link is changed, and can create interdependencies between nodes where changing one should also invalidate the cache of others.

9.3.4 Arbitary tranclusion

The tree structure of documents forbids links which overlap or span different elements, and this includes transclusions. However, as we were to discover when participants tried it in our experiments in chapter 11, there are use cases for, for example, transcluding out the last sentence of one paragraph, and the first of the next. Figure 9.9 illustrates this problem: on the left, the attempted separation of sections of adjacent paragraphs by inserting the transclusive link markup (highlighted); on the right, the result, with error that a paragraph break could not be implied inside the element.

We consider an initial approach to this problem based around ‘sticky elements’ in the concluding remarks of section 11.4.3, given the context of our experiment. In short, when an edit-time transclusion splits a block-level element such as a paragraph, the system must preserve somewhere the information that when transcluded back, the content should merge back into that element at that edge, rather than form a child tree. However, this is a complex problem, especially when the two edges of the transclusion are at different tree depths, and when it is reused across dissimilar documents, so future development is needed.

9.4 Availability

Open Weerkat is licensed under the GNU Affero General Public License version 3, with some components under the 3-clause BSD license, and can be downloaded, with source code, from `https://forge.ecs.soton.ac.uk/projects/weerkat/`.

The source distribution includes an installation script which will copy the required Perl modules and web pages to a given prefix. A sample Apache configuration for correct mod_perl configuration, and use of virtual hosting, is also provided.

Due to the widespread nature of the changes required, the test suite has lagged behind development, and many of the tests no longer use correct API or are even applicable. For this reason, it is omitted from the current distribution.

9.5 Conclusions

As noted, the hardware platform for Open Weerkat development and evaluation was significantly underpowered by modern standards. We feel that proper hosting would help improve performance to within responsive levels. The larger performance spikes are because of a lot of ‘low-hanging fruit’ optimisation which has been omitted for now in the interests of first making the system function correctly under the time constraints. For example, after separation, we push out changes to every link, every anchor, and every transclusion, without checking to see if they have actually changed, which also wastes space and creates versioning noise.

We again restate the maintenance costs of generic linking, as support for it was one of the most complicated tasks involved in implementation, and the resulting concern about the practicality of arbitrary functional links in a wiki context.

On a software engineering note, we found refactoring the system to allow for these changes—in particular the type of node content changing from a string to a DOM object—to be remarkably awkward because of the use of a dynamically-typed language. This is true of all popular web development 'scripting' languages, such as Perl, PHP, and Ruby, by definition, but highlights the utility of a 'stricter' language as projects grow. Many of the early system bugs could have been automatically detected by simplistic static typechecking, which was simply not available. Unfortunately, stricter alternatives such as Java and C# have their own type system, deployment, and modelling issues; Perl's multiparadigmatic approach was invaluable for blending an object-oriented overall system structure with functional programming tools such as our text tree map function.

Chapter 10

Applications

10.1 Introduction

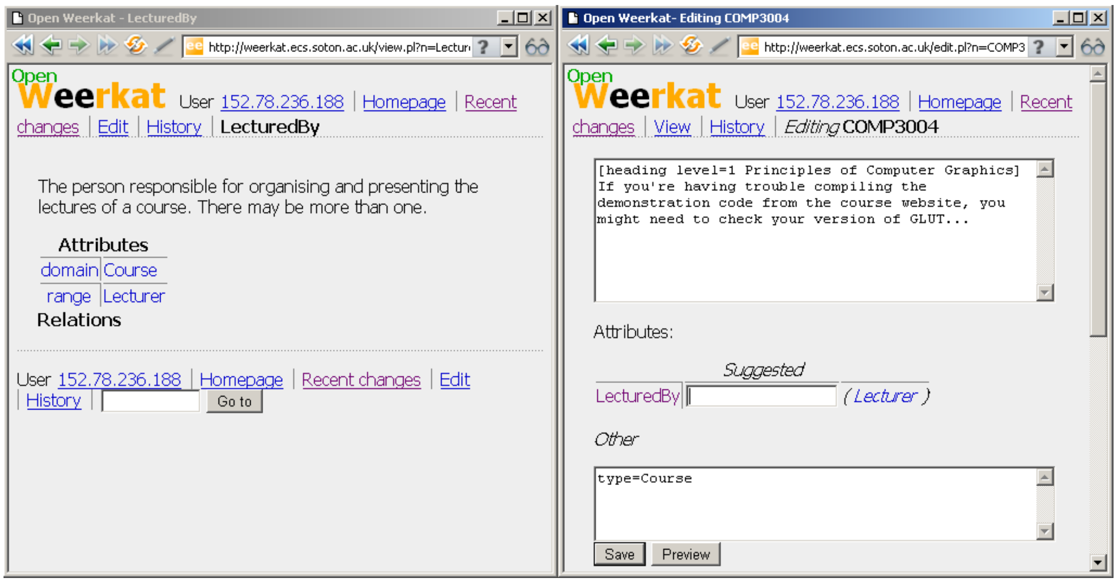
In this chapter, we present some hypothetical use-cases of the system in greater detail. We do not expand upon use in encyclopædic contexts, as we feel that these have been adequately covered by the preceding chapters as our main focus: in particular our comparisons with and experimental studies concerning Wikipedia.

10.2 Enterprise knowledge: University course notes

Wikis have been used to allow undergraduate students to collaborate on notes and advice for their courses. At Southampton, we have historically used 'classic' wiki software, such as UseMod and MediaWiki for this purpose. Weerkat's features offer some advantages over these systems, enabled by its first-class linking capability.

The principle type of data in the wiki is the course. Courses have some common attributes, such as a set of lecturers who run the course, and a year and semester in which they are run. By defining these attributes with appropriate domains and ranges, we can then use Instance Property Editing to encourage and assist in providing this metadata for courses with consistent predicates.

Figure 10.1 demonstrates this in the prototype system. In the left window, we have defined a LecturedBy node, which we shall use as an attribute key. We have simply created this node and set two attributes: Course and Lecturer do not, in fact, even exist for this example. In the right window, we show an editing view of the COMP3004 node, which has been defined in a previous edit to be of type Course. (For technical reasons, the prototype cannot act upon changing type information within the same edit.) As a result, the system now prompts this second editor with possible attributes with a domain of Course: namely, our LecturedBy node, and the expected range.


Open Weerkat - LecturedBy

http://weerkat.ecs.soton.ac.uk/view.pl?n=Lectur

Open Weerkat User 152.78.236.188 | Homepage | Recent changes | Edit | History | LecturedBy

The person responsible for organising and presenting the lectures of a course. There may be more than one.

Attributes	
domain	Course
range	Lecturer

Relations

User 152.78.236.188 | Homepage | Recent changes | Edit | History | Go to

Open Weerkat- Editing COMP3004

http://weerkat.ecs.soton.ac.uk/edit.pl?n=COMP3

Open Weerkat User 152.78.236.188 | Homepage | Recent changes | View | History | *Editing* COMP3004

```
[heading level=1 Principles of Computer Graphics]
If you're having trouble compiling the
demonstration code from the course website, you
might need to check your version of GLUT...
```

Attributes:

Suggested

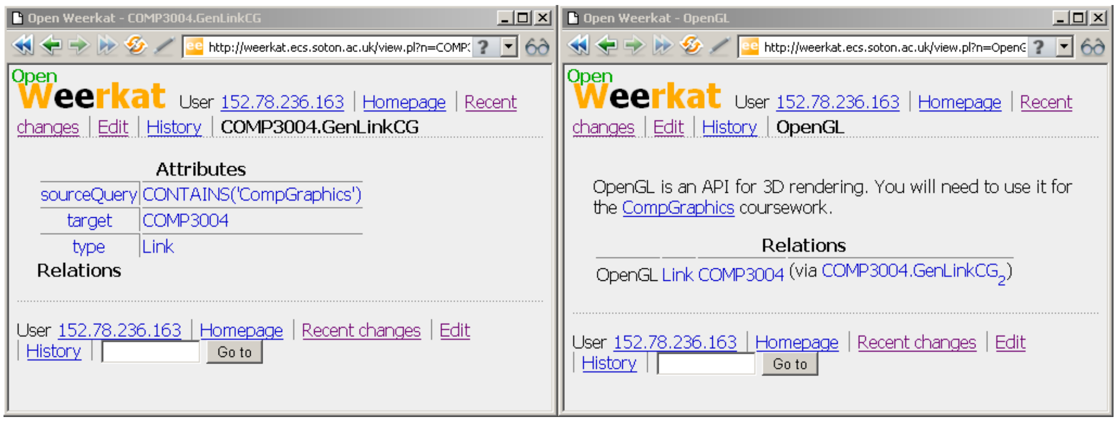
LecturedBy | *(Lecturer)*

Other

```
type=Course
```

Save Preview

FIGURE 10.1: Course Notes: Instance Property Editing


Open Weerkat - COMP3004.GenLinkCG

http://weerkat.ecs.soton.ac.uk/view.pl?n=COMP:

Open Weerkat User 152.78.236.163 | Homepage | Recent changes | Edit | History | COMP3004.GenLinkCG

Attributes	
sourceQuery	CONTAINS('CompGraphics')
target	COMP3004
type	Link

Relations

User 152.78.236.163 | Homepage | Recent changes | Edit | History | Go to

Open Weerkat - OpenGL

http://weerkat.ecs.soton.ac.uk/view.pl?n=OpenG

Open Weerkat User 152.78.236.163 | Homepage | Recent changes | Edit | History | OpenGL

OpenGL is an API for 3D rendering. You will need to use it for the CompGraphics coursework.

Relations

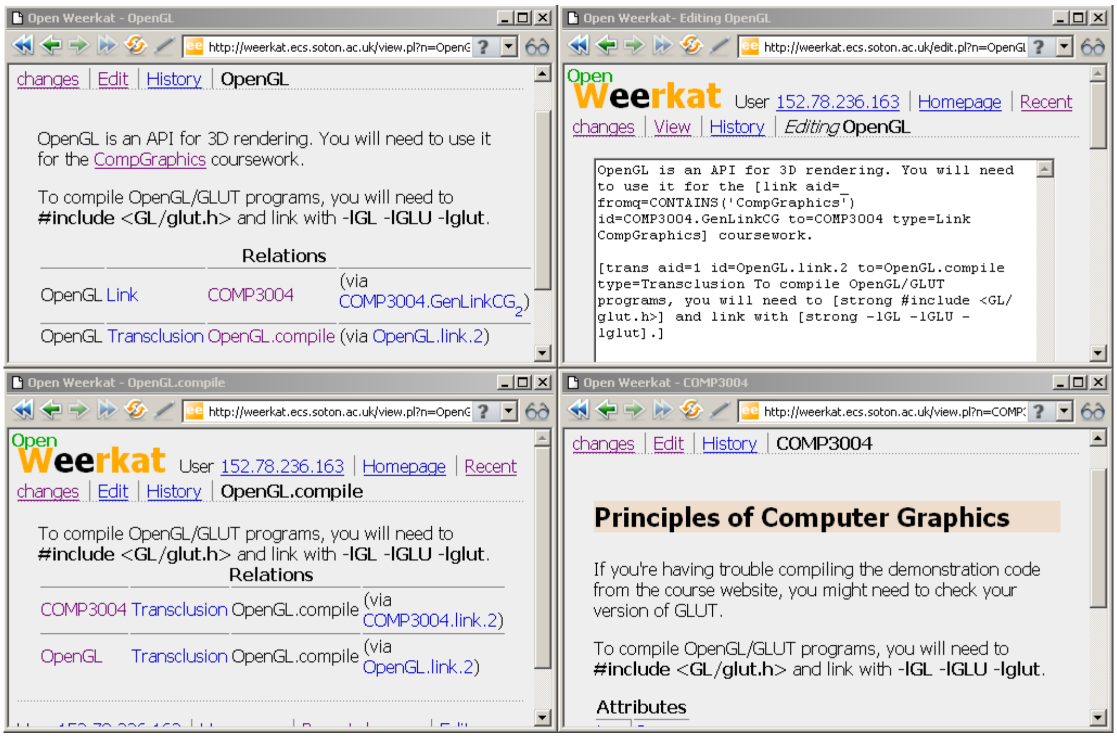
OpenGL Link COMP3004 (via COMP3004.GenLinkCG$_2$)

User 152.78.236.163 | Homepage | Recent changes | Edit | History | Go to

FIGURE 10.2: Course Notes: Generic Linking

Courses have canonical identifiers in the form of course codes, which are suitable for use in node titles and URIs. However, they also have plain text names, and quite often these names are subject to a degree of inconsistency through everyday conversational permutation. For example, COMP3004 is known as 'Comp. Graphics', 'Introduction to Computer Graphics', and 'Principles of Computer Graphics'. Generic links provide a way to link from these disparate strings to the single identifier throughout the entire wiki. This is easier than creating individual links as desired from occurrences of the title, as it avoids the need for the user to first find or recall the opaque course code: a single mapping is created in the system from the human name to the node title.

In the Open Weerkat prototype, we created a generic link for the compound word "CompGraphics", as content search is currently single-word. This is shown in the left window in figure 10.2. Again, all we have done is create a node and set some attributes.


Open Weerkat - OpenGL

http://weerkat.ecs.soton.ac.uk/view.pl?n=OpenG

changes | Edit | History | OpenGL

OpenGL is an API for 3D rendering. You will need to use it for the CompGraphics coursework.

To compile OpenGL/GLUT programs, you will need to **#include <GL/glut.h>** and link with **-lGL -lGLU -lglut**.

Relations

OpenGL Link	COMP3004	(via COMP3004.GenLinkCG$_2$)
OpenGL Transclusion OpenGL.compile (via OpenGL.link.2)		

Open Weerkat- Editing OpenGL

http://weerkat.ecs.soton.ac.uk/edit.pl?n=OpenGL

Open Weerkat User 152.78.236.163 | Homepage | Recent changes | View | History | *Editing* OpenGL

```
OpenGL is an API for 3D rendering. You will need
to use it for the [link aid=_
fromq=CONTAINS('CompGraphics')
id=COMP3004.GenLinkCG to=COMP3004 type=Link
CompGraphics] coursework.

[trans aid=1 id=OpenGL.link.2 to=OpenGL.compile
type=Transclusion To compile OpenGL/GLUT
programs, you will need to [strong #include <GL/
glut.h>] and link with [strong -lGL -lGLU -
lglut].]
```

Open Weerkat - OpenGL.compile

http://weerkat.ecs.soton.ac.uk/view.pl?n=OpenG

Open Weerkat User 152.78.236.163 | Homepage | Recent changes | Edit | History | OpenGL.compile

To compile OpenGL/GLUT programs, you will need to **#include <GL/glut.h>** and link with **-lGL -lGLU -lglut**.

Relations

COMP3004	Transclusion	OpenGL.compile	(via COMP3004.link.2)
OpenGL	Transclusion	OpenGL.compile	(via OpenGL.link.2)

Open Weerkat - COMP3004

http://weerkat.ecs.soton.ac.uk/view.pl?n=COMP:

changes | Edit | History | COMP3004

Principles of Computer Graphics

If you're having trouble compiling the demonstration code from the course website, you might need to check your version of GLUT.

To compile OpenGL/GLUT programs, you will need to **#include <GL/glut.h>** and link with **-lGL -lGLU -lglut**.

Attributes

FIGURE 10.3: Course Notes: Transclusion

Because these are system attributes, our prototype installation has them defined with domains and ranges, which means that editing them is assisted by Instance Property Editing for nodes of type Link.

In the right window we can see the OpenGL node, which talks about the COMP3004 course using this informal term. The generic link matches the text and causes it to be linked to the course page, as can be seen in the relations table.

Several courses overlap in the technologies they cover in their syllabus and coursework. For example, both 'Database Systems' and 'e-Business Techniques' involve using the MySQL database; both 'Principles of Computer Graphics' and 'Interactive Entertainment Systems' involving using OpenGL with GLUT; and many involve the process of uploading and setting permissions on files uploaded to the student's personal webspace. In the standard wiki, this has lead to a degree of redundancy, as it is undesirable to completely indirect the instructions fully. The node regarding the shared technology is oft highly detailed and extensive in task coverage—for example, the OpenGL node covers topics from getting OpenGL example programmes to compile, through principles and basic operation, to lists of online resources and gotchas—which results in links to it thus being surrounded by an ad-hoc simplification of the pertinent task to avoid sending the user to a node which will likely just overwhelm them. Instead, Weerkat allows for transclusion to share this simplification between every course that needs it by creating it as a node itself, possibly also used by the fully-detailed OpenGL instructions. The

OpenGL node can continue to contain sections on many tasks, but each section can in fact be a transclusion of a node on how to, for example, compile an OpenGL programme. This section can then be broken down into a simple command list/reference, and then the explanatory prose covering the principles and caveats involved. The former can be transcluded into course pages as a summary on how to get started.

Figure 10.3 shows transclusion being used to share brief instructions for compiling GLUT programmes being shared between two pages. The text was original written in the OpenGL page, and placed in the body of a `trans` element with a target of OpenGL.compile. The top-left window shows the node being viewed, where the effect of this element is invisible; the top-right shows it being edited, where in-line transclusion shows both the content and the transclusive link responsible for its presence.

The bottom-left window shows the OpenGL.compile node which was created as a result of 'pushing out' the content as a transclusion target. It is by giving this shared text a full node identity that we are able to transclude it into the COMP3004 node, bottom-right. We simply wrote the link markup without any content (`[trans to=OpenGL.compile type=Transclusion]`) to create the link. Users now editing the COMP3004 page will find that the shared content appears in its source, just as for the OpenGL page, and that it can be edited from either.

10.3 Software development: SDL documentation

The development and documentation of software often requires a structured but flexible knowledge repository, and wikis have been used for this task. We focus here on the Simple Directmedia Layer's hypertext documentation project, the SDL Doc Wiki[1].

Documentation wikis are an interesting variation because there is an external organisation source: the source code of the program or library already has strong structure, authoritative domain experts in the form of the developers, and in some cases embedded documentation. The SDL Doc Wiki follows this by giving each function in the SDL API its own wiki node. On top of this, they add manual indices with fairly strong categorisation: lists of functions pertaining to audio, to video, etc.

Function documentation often contains self-contained code samples with enough surrounding context to explain the function's role, and in some cases to compile as a complete program. This naturally leads to a degree of duplication: `SDL_SetVideoMode` is demonstrated not only in its own page, but in `SDL_VideoModeOK` with a different example. The page for `SDL_UnlockSurface` does not show an example, instead directing the reader to its complement, `SDL_LockSurface`. Transclusion could allow these examples

[1] `http://www.libsdl.org/cgi/docwiki.cgi`

to be shared amongst the applicable pages, and promoting them to nodes in their own right also allows a given example to be linked to in, for example, a mailing list discussion.

Functions have common properties, as can be seen from the consistent structure of their nodes. In particular, most function pages have a 'requirements' section at the bottom containing an table with the header to include to declare the function, the API version in which it was introduced, and the required library file to link against. These properties could be handled via node attributes and instance property editing, encouraging their presence for all nodes, and also making the information available in a machine-processable form.

Finally, function names are reliably unique identifiers (they should never need context to disambiguate, at least in a C API with no namespacing), and it is quite likely that a user may wish to view documentation for functions which are referenced in the description or examples of others. This makes them strong candidates for generic linking, although we note that 'all substrings of a certain form to the node with that string as a title' is closer to the 'classic' wiki CamelCase form of generic linking that the explicit-link form in our prototype. As it stands, either each function would need its own generic link from its name to its node (which is clearly an undesirably nontrivial manual process), or the prototype would need to be expanded to implement parametric links as per the model. It would then be possible to specify a link from `CONTAINS(?n)` to `?n`. (Of nodes containing the word $?n$, link to $?n$.) A more advanced case may wish to pattern-match $?n$ to ensure it is an `SDL_...` function as an addition filter.

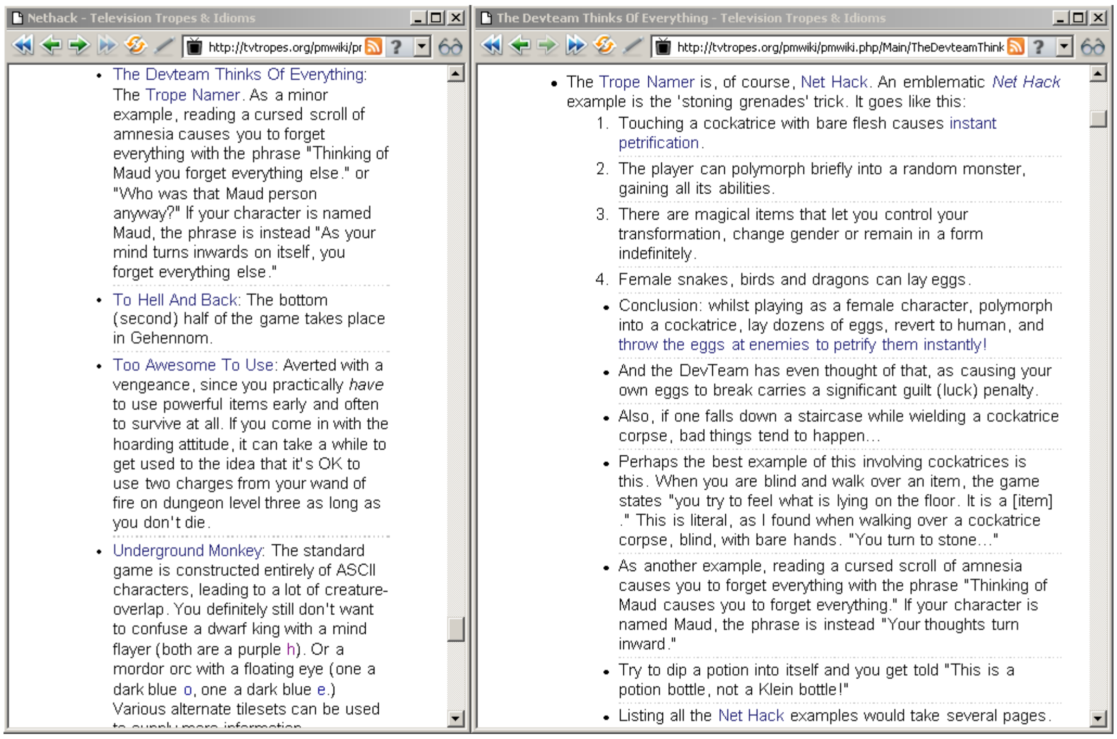
10.4 Community: Tropes in media

TVTropes[2] is a communal-effort website documenting the use of literary devices and conventions in films, books, games, and other media. Each trope has a node with a description and a list of examples in works; each work has a node with a description and a list of the tropes it demonstrates.

Once based on PmWiki, they now run on a system so heavily modified that it is effectively bespoke. While it has domain-specific features, the core of the wiki functionality is effectively a 'classic' wiki with untyped, embedded links. In particular, unlike MediaWiki, but like early wikis like MoinMoin, it performs CamelCase-style generic title linking.

The TVTropes system *does* have limited support for node types: nodes can be specified to be one of an enumeration of types such as 'work', 'trope', or 'contributor'. They also have an 'index' system, which allows a node to be placed in an arbitrary number of sequences of nodes; these sequences are used in a manner similar to Wikipedia categories.

[2]`http://tvtropes.org/`

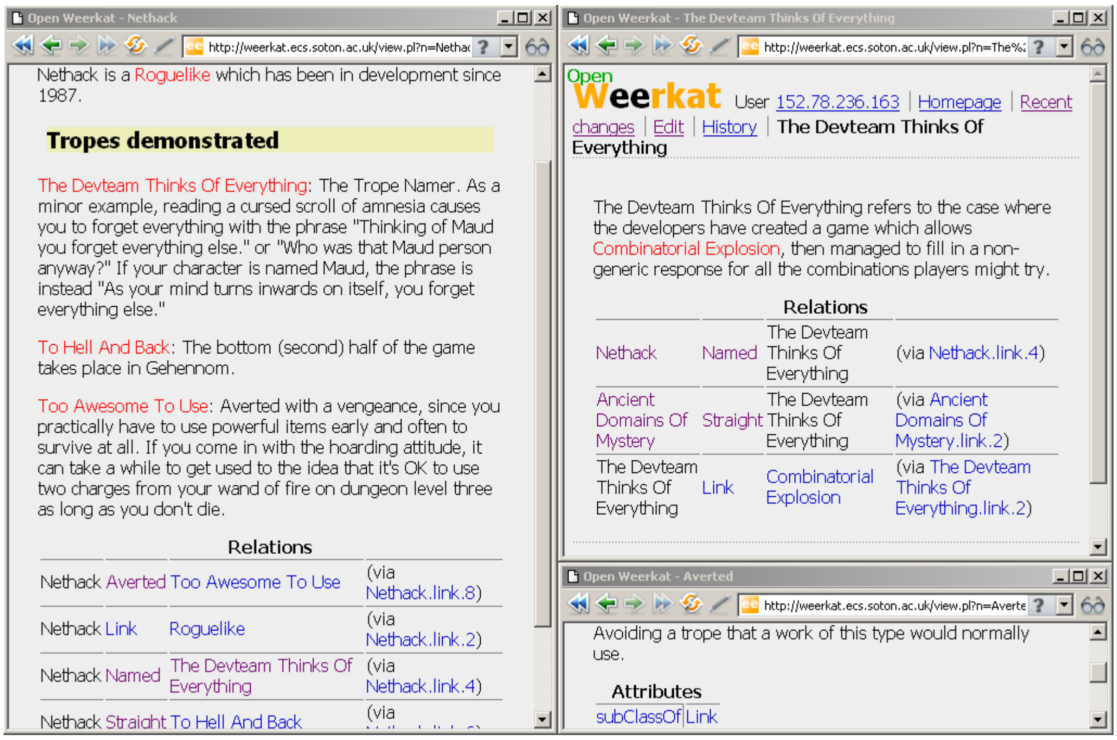

Nethack - Television Tropes & Idioms
http://tvtropes.org/pmwiki/pr

- The Devteam Thinks Of Everything: The Trope Namer. As a minor example, reading a cursed scroll of amnesia causes you to forget everything with the phrase "Thinking of Maud you forget everything else." or "Who was that Maud person anyway?" If your character is named Maud, the phrase is instead "As your mind turns inwards on itself, you forget everything else."
- To Hell And Back: The bottom (second) half of the game takes place in Gehennom.
- Too Awesome To Use: Averted with a vengeance, since you practically *have* to use powerful items early and often to survive at all. If you come in with the hoarding attitude, it can take a while to get used to the idea that it's OK to use two charges from your wand of fire on dungeon level three as long as you don't die.
- Underground Monkey: The standard game is constructed entirely of ASCII characters, leading to a lot of creature-overlap. You definitely still don't want to confuse a dwarf king with a mind flayer (both are a purple h). Or a mordor orc with a floating eye (one a dark blue o, one a dark blue e.) Various alternate tilesets can be used

The Devteam Thinks Of Everything - Television Tropes & Idioms
http://tvtropes.org/pmwiki/pmwiki.php/Main/TheDevteamThink

- The Trope Namer is, of course, Net Hack. An emblematic *Net Hack* example is the 'stoning grenades' trick. It goes like this:
  1. Touching a cockatrice with bare flesh causes instant petrification.
  2. The player can polymorph briefly into a random monster, gaining all its abilities.
  3. There are magical items that let you control your transformation, change gender or remain in a form indefinitely.
  4. Female snakes, birds and dragons can lay eggs.
  - Conclusion: whilst playing as a female character, polymorph into a cockatrice, lay dozens of eggs, revert to human, and throw the eggs at enemies to petrify them instantly!
  - And the DevTeam has even thought of that, as causing your own eggs to break carries a significant guilt (luck) penalty.
  - Also, if one falls down a staircase while wielding a cockatrice corpse, bad things tend to happen...
  - Perhaps the best example of this involving cockatrices is this. When you are blind and walk over an item, the game states "you try to feel what is lying on the floor. It is a [item]." This is literal, as I found when walking over a cockatrice corpse, blind, with bare hands. "You turn to stone..."
  - As another example, reading a cursed scroll of amnesia causes you to forget everything with the phrase "Thinking of Maud causes you to forget everything." If your character is named Maud, the phrase is instead "Your thoughts turn inward."
  - Try to dip a potion into itself and you get told "This is a potion bottle, not a Klein bottle!"
  - Listing all the Net Hack examples would take several pages.

Figure 10.4: TVTropes: a work and a trope in their system



Nethack is a Roguelike which has been in development since 1987.

Tropes demonstrated

The Devteam Thinks Of Everything: The Trope Namer. As a minor example, reading a cursed scroll of amnesia causes you to forget everything with the phrase "Thinking of Maud you forget everything else." or "Who was that Maud person anyway?" If your character is named Maud, the phrase is instead "As your mind turns inwards on itself, you forget everything else."

To Hell And Back: The bottom (second) half of the game takes place in Gehennom.

Too Awesome To Use: Averted with a vengeance, since you practically have to use powerful items early and often to survive at all. If you come in with the hoarding attitude, it can take a while to get used to the idea that it's OK to use two charges from your wand of fire on dungeon level three as long as you don't die.

Relations

Nethack	Averted	Too Awesome To Use	(via Nethack.link.8)
Nethack	Link	Roguelike	(via Nethack.link.2)
Nethack	Named	The Devteam Thinks Of Everything	(via Nethack.link.4)
Nethack	Straight	To Hell And Back	(via

Open Weerkat - The Devteam Thinks Of Everything
http://weerkat.ecs.soton.ac.uk/view.pl?n=The%2
Open Weerkat
User 152.78.236.163 | Homepage | Recent changes | Edit | History

The Devteam Thinks Of Everything

The Devteam Thinks Of Everything refers to the case where the developers have created a game which allows Combinatorial Explosion, then managed to fill in a non-generic response for all the combinations players might try.

Relations

Nethack	Named	The Devteam Thinks Of Everything	(via Nethack.link.4)
Ancient Domains Of Mystery	Straight	The Devteam Thinks Of Everything	(via Ancient Domains Of Mystery.link.2)
The Devteam Thinks Of Everything	Link	Combinatorial Explosion	(via The Devteam Thinks Of Everything.link.2)

Avoiding a trope that a work of this type would normally use.

Attributes

subClassOf | Link

Figure 10.5: TVTropes: a work and a trope in Open Weerkat

While the index system is technologically comparable to Memex trails, most are sorted alphabetically, and thus are not intended to be read in sequence as a narrative.

The TVTropes community are effectively developing a folksonomy through their organisational efforts, although their system has limited capability to model this. In prose, they define sub- and super-trope relations, as well as tropes which are similar, contrasting, and inverse. The relations between works and tropes can also be fairly rich, again in prose. They comment upon whether a work 'plays a trope straight' (uses it conventionally), or deviates somehow: it may 'avert' it (simply not use a trope which would be expected), 'subvert' it (build audience expectation, but ultimately do something different), 'defy' it (make a point of not following a trope), 'invert' it (do the exact opposite), and so forth. There may be categorisation of examples within a trope node, but this is rather ad-hoc, and depends on the use of plain-text headings. A common grouping is works by media (e.g. literature, film, live-action TV), but some pages may be grouped by specialisations of the trope (e.g. 'Living Toys' adds an outer categorisation of good toys vs. evil), and others unsorted entirely (at least until an editor is inclined to change this).

Figure 10.4 shows the game Nethack (left) and the trope 'The Devteam Thinks Of Everything' (right; herein 'Devteam' for brevity) in the TVTropes system. Note that that Nethack uses the Devteam trope is recorded twice in the wiki: once in each node. Worse still, list items are often expanded to clarify the example, or expand upon how the work and trope relate. The details of the case where the player is called 'Maud' while reading a cursed scroll of amnesia show the overlap and inconsistency this can lead to. There is no automatic assistance to synchronise examples between works and tropes: it is common for a work to list a trope without being listed as an example on the trope's node, or vica-versa.

The limitations of a classic, embedded links wiki mean that this is the only practical way to model their data. Figure 10.5 shows a mapping of a subset of the Nethack example to our prototype system. Because first-class links allow for easy bidirectional navigation, the relations table automatically 'fills in the gaps' between works and tropes. Typed links using the informal TVTropes ontology allow the relations table to be more informative, and also keep the data machine-processable. We show one of our link types in a small window bottom-right. Note that it is a subclass of the Link type, allowing it to be used as the type of links without the system also implying an explicit type of Link. Once more, this is simply a new node with an attribute, subClassOf, set. Typed links could also be used to explicitly relate tropes which each-other, such as inverses.

We have chosen here to use an organisation where one type of node—works—holds the example prose, while the other—tropes—is covered by the relations table. However, should an example of a work using a trope be particularly noteworthy or merit extended discussion, it can be shared between the two contexts using transclusion. This allows the node prose to be reduced to just the examples which are most informative or interesting without compromising on recording all the work/trope relations.

The TVTropes system uses CamelCase-style generic title linking, which can be reproduced with our prototype but, as with the SDL function case, only by creating a generic link for each title. Given the different body structure of TVTropes articles to the encyclopedic style which informed our 'only link first instance' approach, we may also need to reconsider the matching algorithm for this context. Because the text is not a continuous prose, but a list, it can be useful for titles to be repeatedly linked wherever they appear. This can be achieved by modifying the GenericLinkMatcher module, and could even become a configuration option. It is also worth considering that these links would increase 'noise' in the relations table, by populating it with additional untyped links amongst the work/trope relations.

Works have common properties, such as media, publisher, genre, and year, but such information is not always present on TVTropes pages as it does not aim to be an artistic database. Instance property editing would help with capturing this information for other purposes—for example, TVTropes *does* implicitly contain the media of each work by virtue of the subheading categorisation within each trope page—but currently the prototype has no mechanism for such automatic reordering of relations or compositions based on related metadata.

10.5 Conclusions

We have listed here a small set of hypothetical, non-encyclopædic use cases grounded in real-world systems, along with explanation and exploration of how Weerkat's current feature set can be used to help these organisations manage their knowledge stores.

In contrast to this expert use of the system, the next chapter revisits the micro-scale wiki experiment to assess how effectively untrained users new to the software can use it to complete tasks.

Chapter 11

Experiment III: Evaluation

11.1 Introduction

An experiment was carried out to evaluate the effects of redesigning the wiki concept in terms of open semantic hypermedia on its core usability and expressiveness.

Our hypothesis was that, generally, task performance would improve; we cover the specifics in the following section. We also anticipated that the additional syntactic complexity would prove to be a minor but pervasive impediment.

11.2 Procedure

To allow a direct comparison against a regular wiki system, we based our procedure tightly upon that of the self-report section of Experiment II, described in chapter 6. The same environment was recreated in the Weerkat system as closely as possible, given that its significantly different markup and architecture made this a manual process.

We shall now recap the participant tasks, along with the expected changes to the participant's solution when faced with the new system capabilities.

Edit description of same villain in two movies. The text is clearly duplicated, although not quite consistent. We expected participants to tackle to this redundancy with transclusion of the content from a new node dedicated to the villain.

Add fact to specific article, and to summary in general article. This task covers level-of-detail with information redundancy; given our earlier results, we thus didn't expect Weerkat to trigger different behaviour to MediaWiki. Participant preference for context-specific text would render transclusion unsuitable. The domain is the London Underground.

Refine links pointing to the disambiguation node for 'shelf'. With MediaWiki, this task involved finding the backlinking (reverse link navigation) capability, then finding and editing the embedded links. Weerkat makes these reverse links more prominent, but also increases the verbosity of embedded links; we expected participants following this approach to offset initial success from the former by increased difficulty due to the latter. A faster and easier, yet more advanced, approach is possible by directly editing the link—accessible from the relations table—rather than finding its embedded version.

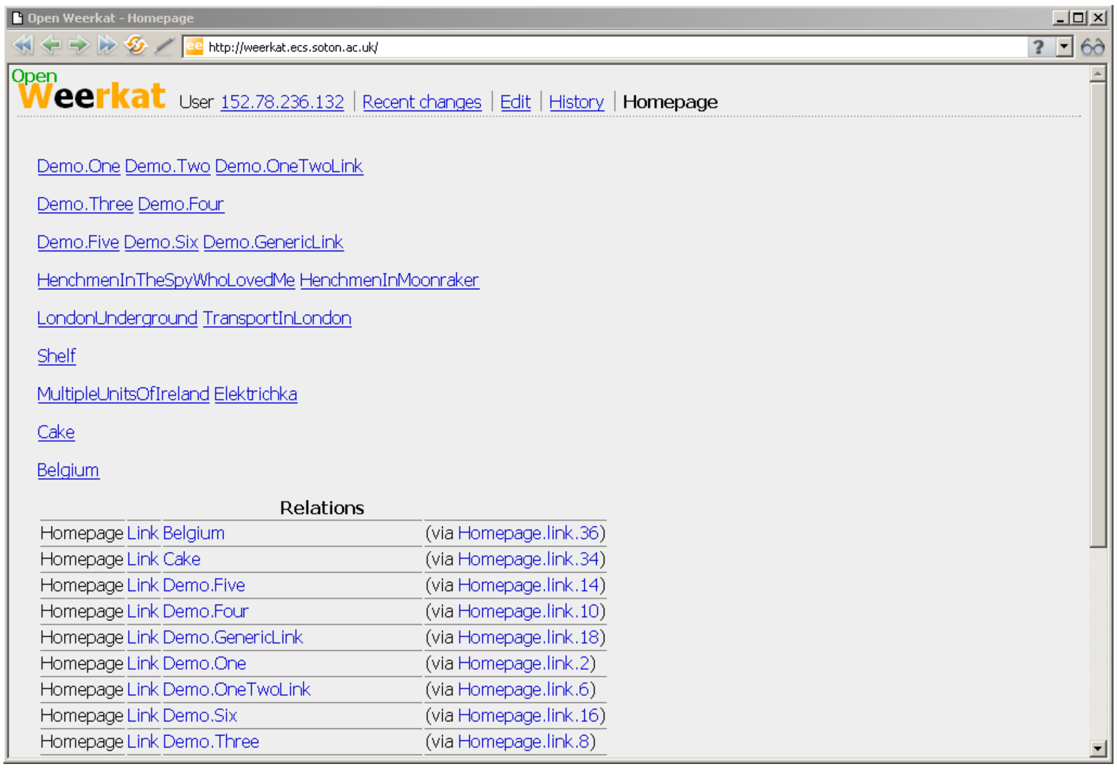
Create page summarising two other articles about type of train. As an aggregation of summaries task, transclusion again surfaces as a possibly useful tool; however, our previous experiment led us to hypothesise that the context-dependent nature of summaries may lead participants not to use it. The domain is a kind of train called 'multiple units' across different countries.

Add links where appropriate to plain text of 'cake' article. Generic links are a potential solution to this task, although the isolated nature of the tasks does not particularly encourage it. This task *does* test straightforward inline link creation, which we expected to be slightly hampered by the more complicated syntax.

Add fact to attributes of 'Belgium' article. As opposed to the MediaWiki infobox, we used the attributes of the article. Because appropriate Instance Property Editing metadata was provided for the attribute in question (the date of EU membership), the system provided a form field to prompt for it. Hence, we expected this task to prove relatively straightforward. As a minor complication, the investigator stated the date in prose format ("25th March 1957"), to see if the system adequately prompted the participant to use the desired YYYY-MM-DD format.

The script and questionnaire for this experiment are provided in appendix B. In particular, each session began with the investigator showing a quick demonstration of first-class links and their embedded forms, transclusion, and generic linking. The pages for these demonstrations were linked from the Homepage. Since there was no week of editing to discuss, the total session was limited to the same half-hour span as the on-line reporting of the previous experiment.

Figure 11.1 shows the homepage of the prototype system configured for the experiment, for comparison with the Mediawiki system in figure 6.1. Figures 11.2, 11.3, and 11.4 show the pages used for the demonstration given by the investigator. The first showed a first-class link with no embedded form, and that it appears in the Relations table in both directions. The second demonstrated transclusion, including edit-time transclusion allowing the content to be edited in-place, and that this change would then affect the transcluded Demo.Four page. The third demonstration showed generic linking, including that the generic link appeared in the source of the page so that the reason that the word is linked can be seen.



Open Weerkat User 152.78.236.132 | Recent changes | Edit | History | Homepage

Demo.One Demo.Two Demo.OneTwoLink

Demo.Three Demo.Four

Demo.Five Demo.Six Demo.GenericLink

HenchmenInTheSpyWhoLovedMe HenchmenInMoonraker

LondonUnderground TransportInLondon

Shelf

MultipleUnitsOfIreland Elektrichka

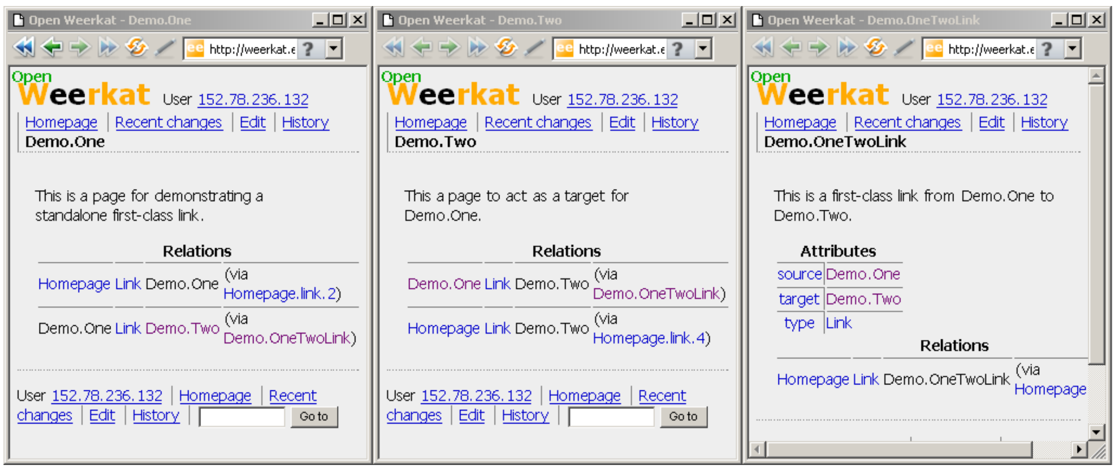
Cake

Belgium

Relations

Homepage	Link	Belgium	(via Homepage.link.36)
Homepage	Link	Cake	(via Homepage.link.34)
Homepage	Link	Demo.Five	(via Homepage.link.14)
Homepage	Link	Demo.Four	(via Homepage.link.10)
Homepage	Link	Demo.GenericLink	(via Homepage.link.18)
Homepage	Link	Demo.One	(via Homepage.link.2)
Homepage	Link	Demo.OneTwoLink	(via Homepage.link.6)
Homepage	Link	Demo.Six	(via Homepage.link.16)
Homepage	Link	Demo.Three	(via Homepage.link.8)

Figure 11.1: Open Weerkat prototype set up for experiment



Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History Demo.One

This is a page for demonstrating a standalone first-class link.

Relations

Homepage	Link	Demo.One	(via Homepage.link.2)
Demo.One	Link	Demo.Two	(via Demo.OneTwoLink)

User 152.78.236.132 | Homepage | Recent changes | Edit | History |

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History Demo.Two

This a page to act as a target for Demo.One.

Relations

Demo.One	Link	Demo.Two	(via Demo.OneTwoLink)
Homepage	Link	Demo.Two	(via Homepage.link.4)

User 152.78.236.132 | Homepage | Recent changes | Edit | History |

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History Demo.OneTwoLink

This is a first-class link from Demo.One to Demo.Two.

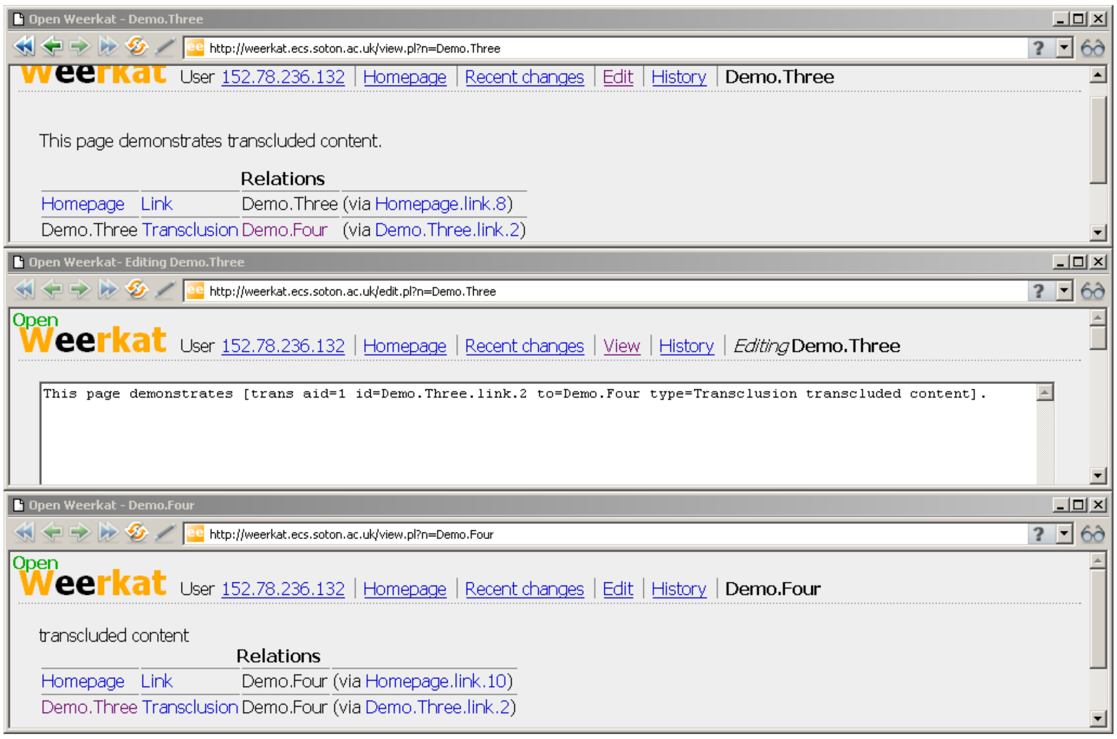
Attributes

source	Demo.One
target	Demo.Two
type	Link

Relations

Homepage	Link	Demo.OneTwoLink	(via Homepage

Figure 11.2: Open Weerkat first-class link demonstration


Open Weerkat - Demo.Three

http://weerkat.ecs.soton.ac.uk/view.pl?n=Demo.Three

User 152.78.236.132 | Homepage | Recent changes | Edit | History | Demo.Three

This page demonstrates transcluded content.

Relations		
Homepage	Link	Demo.Three (via Homepage.link.8)
Demo.Three	Transclusion	Demo.Four (via Demo.Three.link.2)

Open Weerkat- Editing Demo.Three

http://weerkat.ecs.soton.ac.uk/edit.pl?n=Demo.Three

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | View | History | *Editing* Demo.Three

```
This page demonstrates [trans aid=1 id=Demo.Three.link.2 to=Demo.Four type=Transclusion transcluded content].
```

Open Weerkat - Demo.Four

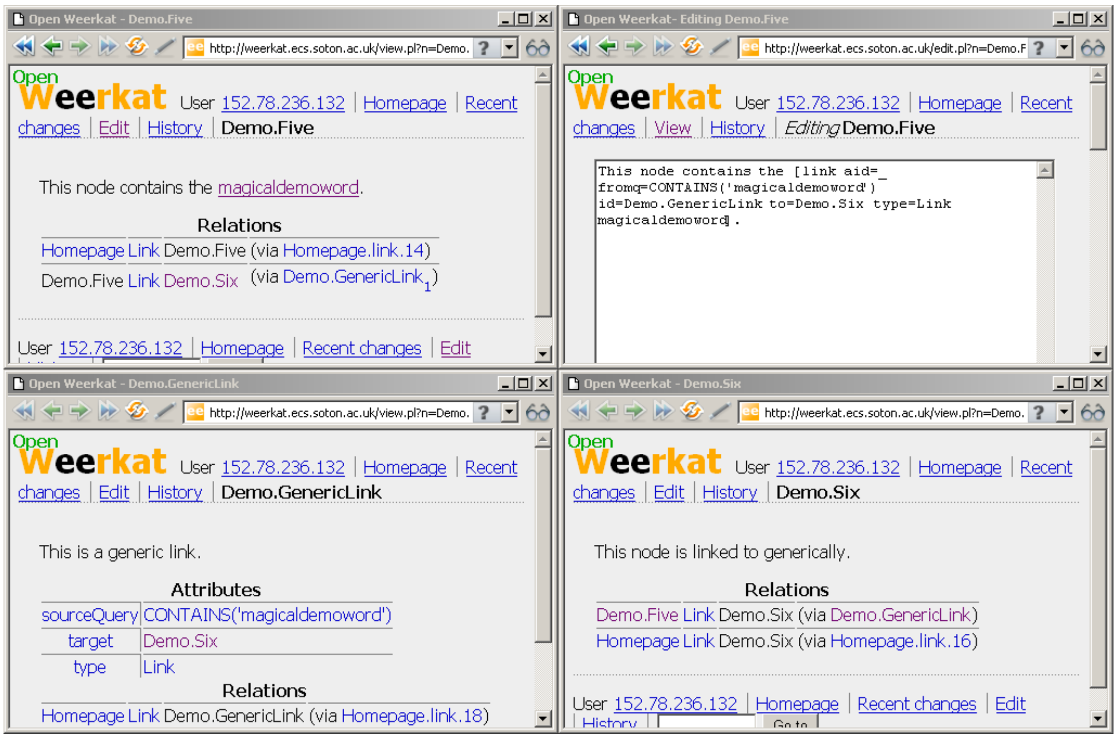
http://weerkat.ecs.soton.ac.uk/view.pl?n=Demo.Four

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History | Demo.Four

transcluded content

Relations		
Homepage	Link	Demo.Four (via Homepage.link.10)
Demo.Three	Transclusion	Demo.Four (via Demo.Three.link.2)

Figure 11.3: Open Weerkat transclusion demonstration


Open Weerkat - Demo.Five

http://weerkat.ecs.soton.ac.uk/view.pl?n=Demo.

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History | Demo.Five

This node contains the magicaldemoword.

Relations		
Homepage	Link	Demo.Five (via Homepage.link.14)
Demo.Five	Link	Demo.Six (via Demo.GenericLink$_1$)

User 152.78.236.132 | Homepage | Recent changes | Edit

Open Weerkat- Editing Demo.Five

http://weerkat.ecs.soton.ac.uk/edit.pl?n=Demo.F

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | View | History | *Editing* Demo.Five

```
This node contains the [link aid=_
fromq=CONTAINS('magicaldemoword')
id=Demo.GenericLink to=Demo.Six type=Link
magicaldemoword].
```

Open Weerkat - Demo.GenericLink

http://weerkat.ecs.soton.ac.uk/view.pl?n=Demo.

Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History | Demo.GenericLink

This is a generic link.

Attributes	
sourceQuery	CONTAINS('magicaldemoword')
target	Demo.Six
type	Link

Relations		
Homepage	Link	Demo.GenericLink (via Homepage.link.18)

Open Weerkat - Demo.Six

http://weerkat.ecs.soton.ac.uk/view.pl?n=Demo.

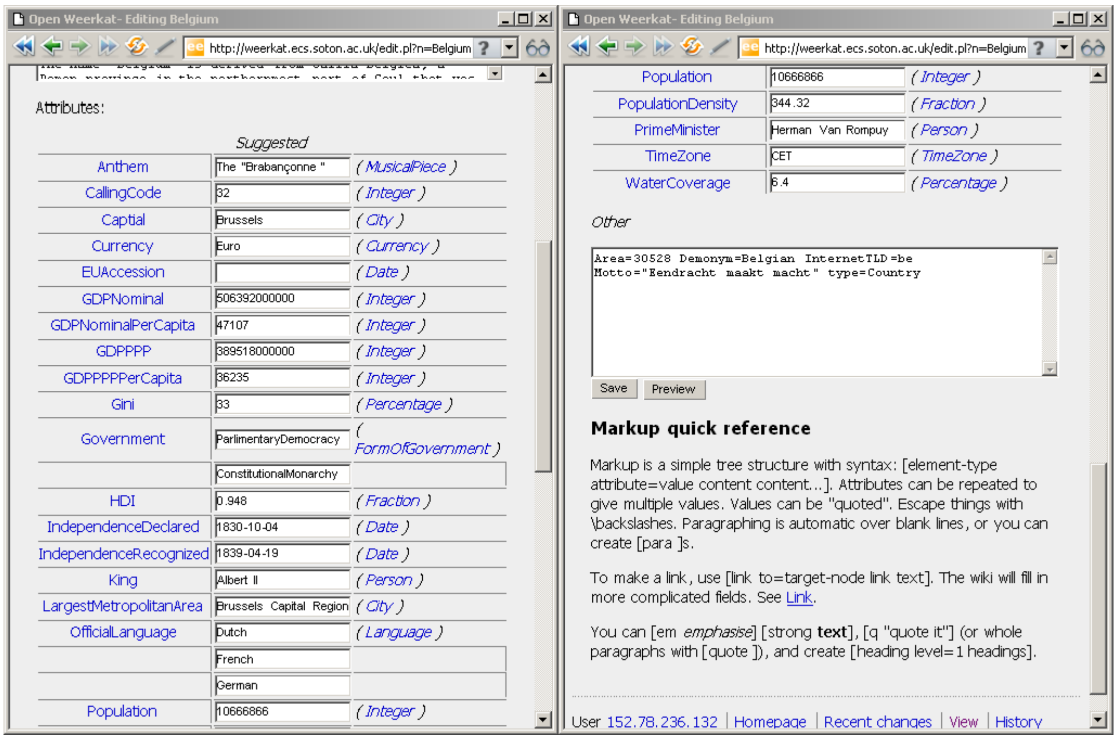
Open Weerkat User 152.78.236.132 | Homepage | Recent changes | Edit | History | Demo.Six

This node is linked to generically.

Relations		
Demo.Five	Link	Demo.Six (via Demo.GenericLink)
Homepage	Link	Demo.Six (via Homepage.link.16)

User 152.78.236.132 | Homepage | Recent changes | Edit | History | Go to

Figure 11.4: Open Weerkat generic link demonstration


Open Weerkat- Editing Belgium
http://weerkat.ecs.soton.ac.uk/edit.pl?n=Belgium

Attributes:

	Suggested	
Anthem	The "Brabançonne "	(*MusicalPiece*)
CallingCode	32	(*Integer*)
Captial	Brussels	(*City*)
Currency	Euro	(*Currency*)
EUAccession		(*Date*)
GDPNominal	506392000000	(*Integer*)
GDPNominalPerCapita	47107	(*Integer*)
GDPPPP	389518000000	(*Integer*)
GDPPPPPerCapita	36235	(*Integer*)
Gini	33	(*Percentage*)
Government	ParlimentaryDemocracy	(*FormOfGovernment*)
	ConstitutionalMonarchy	
HDI	0.948	(*Fraction*)
IndependenceDeclared	1830-10-04	(*Date*)
IndependenceRecognized	1839-04-19	(*Date*)
King	Albert II	(*Person*)
LargestMetropolitanArea	Brussels Capital Region	(*City*)
OfficialLanguage	Dutch	(*Language*)
	French	
	German	
Population	10666866	(*Integer*)
PopulationDensity	344.32	(*Fraction*)
PrimeMinister	Herman Van Rompuy	(*Person*)
TimeZone	CET	(*TimeZone*)
WaterCoverage	6.4	(*Percentage*)

Other

```
Area=30528 Demonym=Belgian InternetTLD=be
Motto="Eendracht maakt macht" type=Country
```

Save Preview

Markup quick reference

Markup is a simple tree structure with syntax: [element-type attribute=value content content...]. Attributes can be repeated to give multiple values. Values can be "quoted". Escape things with \backslashes. Paragraphing is automatic over blank lines, or you can create [para]s.

To make a link, use [link to=target-node link text]. The wiki will fill in more complicated fields. See Link.

You can [em *emphasise*] [strong **text**], [q "quote it"] (or whole paragraphs with [quote]), and create [heading level=1 headings].

User 152.78.236.132 | Homepage | Recent changes | View | History

FIGURE 11.5: Full IPE listing for Belgium page

For a sense of scale, figure 11.5 shows, split across two panes, the full vertical height of the Instance Property Editing table for the Belgium article. Note the blank field for EUAccession, the hyperlinked Date range type next to it, and that some attributes have multiple values, and thus rows.

11.3 Results

Results are presented in the order in which the interviews were performed as for experiment II. Where participant IDs overlap with the previous experiment, the participant is the same (and has consented to being identified as such). While this does imply a degree of ordering bias, it is worth remembering that the previous experiment deliberately involved participants which had been selected to be experienced with the system, which is itself widely used. Even if they had been exposed to Weerkat first within the context of these experiments, they have all used MediaWiki previously.

All of the participants were familiar with Wikipedia disambiguation pages.

11.3.1 Participant nine

For the ‘villains’ task, the participant noted immediately that the character description was duplicated, but “slightly different” in each. They initially suspect that there may be sharing, yet “the transclusion isn’t quite the same as I saw before”. Looking at the source, they determine that they’d have to change it in both, but “in a Wikipedia sense... there’d be a separate page on it and I’d... add it to the main article”. After clarifying that this was a real example from Wikipedia, the participant studied the nodes serving as an example of transclusion. They stated that they were “not sure where is best to put the information to transclude it from... whether to have another page that I put that information in, and transclude it into both, or whether to have it one of these and then transclude it to the other”. After noticing that the transclusion example transcludes the whole page, they decide upon the former approach: a separate page. They create a dangling transclusive link with content of the single sentence stating the fact they have been asked to add, and expect that “possibly it will fill it in, if I understand the linking”. They edit it again to “see if the link has updated in the source”, and note that the link has gained some IDs. The participant puts an empty-body transclusion in the other node, “to see if that transcludes it across”, or if it “sets [the shared node] as empty”. The system does the former “which [was] what [they were] *hoping*, but not necessarily *expecting*... excellent, made some sense, excellent”.

The participant again planned to use a dangling link, this time from the homepage, to create the new page for general Multiple Units. They decided to “see what happens if I don’t put any text in; see if it reuses the name of the link”. The prototype system does not, so they noted that they “have an invisible link”, and edit it to have text. After studying the specific train articles, they decide to “transclude the first paragraph”, and do so by editing the pages to make those paragraphs the body of a transclusive link to new pages. They added a brief introduction and a pair of headings to their general page using the markup quick reference, questioning the use of the “standard formatting”, rather than a more conventional wiki syntax.

While transcluding, they observed that they found it strange that they were specifying the target with the `to` attributed, when “mentally I’m transcluding *from* at the moment, because I’m not putting any content here”. At one point they wished to cancel an edit, but it was pointed out that the browser back function works for this (the interface also provides a link back to the view, although it is not labelled ‘cancel’).

They were ultimately successful after correcting a misremembered target page name. While doing so they noted that the relations table showed the target even though it did not exist, and that the system prompted to create it when clicked: “that’s good”. They decided to actually change the name they gave to the transcluded paragraph, and realised that this left an orphan copy of the content with the old name. Should the summary need to be changed, the participant recognised that the modification could be made in the new, general page, and made a test edit to confirm this. They commented that they "like the fact that the text is in both, but if I change it in one, then it knows where it got that from so it goes back and changes it".

For the 'Belgium' task, the participant immediately edited the page and noted the "nice little editing area for that, but I don't know how to add one". They commented on the attribute format in the 'other' box, but "don't understand why they're there rather than in this attributes list", and quickly determine that the node source contains only the content. Shortly after, they noted the existing field, and decide to look at the rest of the list for a formatting cue. Having found one, they also noted the link after the field (the type), and followed it, finding that it confirms the format they now intended to use. Having saved, they curiously click on the new date value in the attributes table, and discovered that the page for that date doesn't yet exist. Despite having completed the task, they expressed interest in "what the list of other things at the bottom is for", and moved the date out of the table and into the other box, "to see whether that then fills [the form field] in, or whether it just sits there". They noted that it had, indeed, migrated back to the table. After prompting to try looking at the attribute node, they note the domain and range, and ask themself if Country is defined: "Country's just a description I guess". The investigator pointed out that Belgium had a type, and this matched the domain; attributes which don't match up this way end up in the 'other' box. The participant understood and commented that they "like that".

The participant found the pages with ambiguous links immediately through the relations table, went to one, searched for 'Shelf', and read the surrounding text for context. After looking at the page for the suspected kind of shelf to confirm that it was the correct disambiguation, they edited the ambiguous node and, despite commenting on the "lots of links", found and corrected the link quite quickly. They checked that the link target was now correct by following it, then used the relations table to get back from the specific shelf type to Shelf, following a link in reverse. On the Shelf page, they then checked that the previously ambiguous node is no longer listed as an incoming link. They perform the same process for the other ambiguous page: for this instance, the text of the link to the correct type of Shelf is different from the page name. They noted the page name for the correctly disambiguated shelf in the browser status bar, as the relations table was scrolled offscreen at the time.

Asked about their reaction when editing the first, larger, page, they comment that "it was quite difficult, and there were a lot of links, and because they're quite big they take up a lot of space, so as you're scanning it it then interrupts where you are in the text, because you have to scan through the link, and then carry on with reading the text…there was one bit…a big section where there were lots of links, one after another, and it was quite difficult to read that". Prompted for another way to tackle the problem, they "guess [they] could have gone to the…node page that exists in-between"; after asking themself how to get there, they checked the relations table. Knowing that the target text takes you to that page, they tried "this via part", and note "there's the source, and the target, I could have changed it there. That makes sense." They commented that they had forgotten about this by the time they reached this task, and that while "it does make a bit more sense to—makes it simpler to *find* what you're changing there", "I now have to remember to come down and look at the relations table to find that link, whereas I'm looking at it *here* [in the body text] and my instinct is to work on this bit, to make the change." They suggested an "edit version" of the page, where the link in the body text "gave me a link *to* the page and a link to the link. . . it might keep that in my mind to be able to go to that link in-between". They concluded that they may remember in future, but the feature is out of the normal workflow. When the tooltip that shows the name of the link while hovering over an anchor was pointed out to them, which they had triggered several times so far, they stated that they "hadn't noticed it; don't normally get one; hadn't really looked at it or read what it was".

For the London Underground, the participant added the fact to the history section, then used the relations table to navigate to Transport in London, because they'd "used it a few times [and] now I know it's there". It was "only a guess" that the pages may be linked, and they correctly determined that the link was from Transport in London to London Underground. They compared the summaries in the two articles to see if they were the same, and concluded otherwise. They decide to use transclusion to share the sentence containing the fact, but are "troubled a little bit" that they're "transcluding a *sentence* of text, and that that will create a node in-between and [they] need to *name* that node"; that "conceptually it seems weird having a whole page essentially just for this one sentence of text". They deliberately left content in both new transclusive links to see what happened, hypothesising correctly that the latter save would update the former. They noted that this case has caused a slight grammatical problem: they've actually shared a clause, and it has a capital letter because it is the start of the sentence in one page, which is incorrect in the other.

While adding links to the Cake article, the participant wondered aloud if they kept needing to specify the link type, then noted that the markup summary tells them that they did not. They noted that the example given there (`[link to=Target link text]`) is "slightly confusing", as the word 'link' is used both as the element name and some example anchor text. They also commented on the colouration of links; specifically that the relations table does not follow the red-for-dangling convention of the body text, which was explained as a technical limitation.

After prompting, the participant could not think of another way to add all the links, having done them manually; after asking if they'd consider generic links, the investigator recovered the example. The participant asked if the `CONTAINS` query was the only one available, and if you can only use text, not regular expressions or such. They also asked if this capability was planned to be expanded, before suggesting "a simplified version where you just, rather than having a sourceQuery, it' just `contains=`, and then you put your word in, and then tell it what to replace that with". They questioned the scope of generic links, guessing correctly that they affect all pages.

They then wanted to create a generic link, and pondered how. Choosing to create one for the word 'food', they looked back at the example and noted "it's just a node with some attributes, so I *could* edit the cake page and do all that inline I guess". While navigating between pages for the task and examples, they commented that the relations table is faster than the body text because it's a alphabetically-sorted list, rather than prose. They attempted to create the generic link inline as planned, but had problems because of the difference between attribute names for nodes and for inlined links. They also considered it a "strange thing" that they were supplying both a `CONTAINS` clause *and* some content: "I don't know whether to put that still within the link, or outside of the link, or what". The system created nested links as a result, having both a generic link and a literal anchor. The participant noted the duplication in the relations table, and also the version subscript for one of them, but was unsure why: they guessed that the word 'food' might have been recursively replaced with links. They concluded that generic links are "a little bit tricky because, if you can't add them in the page—they're a useful extra, administrative tool—I guess you could let users have them but don't tell them much about it; it should be a very advanced feature that someone should sit there with. . . a generic link interface that they can add a bunch to".

After the investigator stated that they created their generic link example as a standalone page (rather than inline), the participant wanted to try the same. They created a new page via the address bar again, and entered the attributes to create a link from the world and to the Transport in London page as a test, initially omitting the type. They asked if they have to tell it that the node is a link; why the system did not know. "What would have a source and a target that isn't a link? I think that makes sense." They noted on the Cake page that only one of the ands had become a link "because it's following the Wikipedia sense of only defining it the first time, which is sensible, otherwise you'd have ands all over the place". They followed it to Transport in London and check that that had many incoming links in the relations table. They temporarily experimented with removing the type again, which triggered a bug in the prototype causing the link to affect the page body but not show in the relations table. Finally, they noted that Transport in London links to itself now, which "made sense" because it contained the word and.

11.3.2 Participant eight

The participant added the fact about being the oldest metro "directly" into the London Underground article, and then went back to the homepage and used the relations table there to navigate to the Transport in London article. Having found the appropriate section there, they commented that they "were not sure exactly how to do it but [they'd]. . . transcribe the content into here", and asked if that was possible. Told that transclusion was only possible for whole nodes, and they they would have to break up the content, they proceeded to do so with minor guidance.

For the Multiple Units task, the participant was initially unsure how to create a new page, and decides to do so by editing the URL in the browser address bar. After being directed to the markup quick-reference to answer their question about headings, they add such for the two countries. They created 'main article' links under these headings, copying one to create the other. The syntactic error caused by a typo ("unexpected space in the value of. . . ") was found and corrected quickly.

For the disambiguation task, the participant spotted, under the outgoing links, that the relations table also showed incoming links, and opened the offending articles in browser tabs. For the smaller article, they search in page to find the context, copy the correct type of Shelf's article name from the address bar having opened it in a tab, and attempt to search in page in the edit view. The browser used for the test does not, however, support searching within text areas, so they find the link manually and correct the `to` attribute, after confirming that this was the correct and sole change required. They then confirm this by hovering over the view of the saved page and seeing the link target in the browser status bar. They do the same for the larger article, about an albatross, although with slightly more difficulty because upon editing they "can't remember exactly where [the link anchor] is".

While the system was saving the larger Shelf change, the participant inquired if there was "a typically better way of doing that", as their reaction to the edit view of the large node was "argh, words". The investigator confirmed that there was another way, and the participant guessed that "if it's a first-class link, could I have gone to that node and edited the link?" Asked how they would *find* the node for the link, they initially consider from the large, albatross article, before wondering if it would be reachable from the Shelf page, which was confirmed as correct.

The participant immediately edits the Belgium article, expecting to see the infobox in the source of the page. They note that there are attributes at the bottom of the edit page, and an 'other' box which does not have the same content as the table. They ask themself "is this [the attribute to add] already in the table", but quickly decide "it doesn't seem to be", and spend quite some time scanning up and down the page. After prompting that the list is sorted alphabetically, the participant realised that there was a blank row for the attribute, and entered the value. They asked "is there any easier way to add *other* attributes. . . because that's what I was looking for originally; I didn't realise that this was already in the table", and it was explained that there's no way to add rows to the table, but the 'other' box accepts attributes using the markup syntax. The participant questioned if "it gets this list from the type or something", and was shown an attribute node, with domain and range; they correctly "assume. . . every page just has a default type attribute, and it's value is then used to populate the rest of these attributes".

As before, the policy for links on the Cake article followed was "nouns, mostly". The participant asked of links "if I don't put text, will it just use the name of the article... or is that just required?". After linking some nouns manually, they were prompted to think of any other approaches, and quickly considered the "magic word" demo (generic linking). They ask if there's a way of making them, or if they have to create a node, and do the latter, naming it Food.GenericLink. They fill in the attributes, save, and plan to look at the Food page to see incoming links to tell if it worked. Looking at the Cake article, however, distracts them as they had already manually linked the word "food", and the system had put a link within a link. Because the prototype handles overlapping links poorly, they decide to "get rid of that whole thing [nested link markup] and just put the word 'food' in", which replaces it with just the generically-linked version, as expected.

The participant spots the duplicated henchman in the film articles, and decides to create a separate page for him, possibly transclude it back. They ask if they have to create a "particular kind" of page to use as a transclusion, and add a heading and the required content to the new page. Upon prompting, they state they would copy-paste some content from the other articles in a real situation, and ask if they could transclude just "particular portions" of a page. Being told that they cannot, they remove the heading from their page, and decide to transclude it (containing only the new fact) into the other two pages.

As final commentary, the participant suggested that the system could do with some "nice interface", like source highlighting. They suggested that when you hover over a link, "some ability to edit in place", while still in the page view. This was triggered by the editing the larger of the shelf pages: "when you have just like *bam!* loads of text it's difficult for people to pick out the syntax". Finally, they note that they had studied a hypertext course last year, and consider that before that they didn't know that links could exist outside of a document (first-class): it's not a feature people are aware of.

11.3.3 Participant ten

Participant ten started on the 'cake' task, and used the quick reference to link nouns. They stated that without it they'd "have been a bit lost I think: I'm not an expert at wikis", and commented on how Wikipedia allows links to have the same target and text. When prompted to consider other approaches, they suggested using an editor with regular expressions, or a small script, to process a list of words: it was "definitely a mechanical process". After redemonstrating generic links, they decided that they wouldn't want to create a link for each word, and noted that generic links would affect every article, not just this one. While making straightforward text edits for the London Underground task, they comment that generic links "surely won't work with everything, because it'll depend on context... sometimes you'll want 1863 to point to the *year* 1863, and sometimes it'll just be a *count* of something".

For the train task, they used the 'go to' box to create a new page, which they populated with a link to each country. They spent some time looking over the two pages to try to understand the domain, and commented that "this is getting fairly ugly, syntactically". Prompted not to worry about quality of writing, they copied and pasted a sentence from each country. Asked what would happen if one then needed a correction, they start "it would have been nice if I could have... " before remembering transclusion. With a recap of the demonstration, they understand that you would break out the shared part into a separate document, and then correctly surmise that you would transclude this into the general page.

The participant handled the shelf disambiguation task using the relations table, finding the links quite quickly and editing in place. They attempted to search-in-page on the larger ambiguous node, but again the browser does not support this. Although having completed the task, they comment that they're "slightly tired of doing it this way, so I'm going to check more carefully that there isn't a better way to do it". The investigator suggests 'disambiguating' the Homepage's link to the Shelf page, if the participant wants to try something. They clicked the word 'Link' in the relations table, but are surprised and confused by the result. Asked what they think they've found, they're not sure, but guess that it may be "links between links". Having been explained that this is the type of links, the participant determined that this was not what they wanted, backed up, and tried the 'via' column instead, which took them to the desired link. They clicked the target Shelf, but because they were viewing, this returned them to the Shelf page. They also clicked the predicate target, but realised that that took them to a page "telling me what a target is". Finally they attempted to edit the link, changing the target, and confirm that this has worked by looking at the now 'disambiguated' Homepage.

For the 'Belgium' task, the participant immediately edited the page and skimmed up and down the editing form. They experimentally clicked on an attribute, but found an undocumented one. They asked if they needed to "create an attribute of 'EU Membership Date', that things can use in general? Because there doesn't seem to be any way to just add something to that table. And that would make sense, because you're going to want to be able to use it for other countries as well." At this point, they noted the 'other' box, and added the fact as `AccessionDate="25 March 1957"`, having noticed that the motto attribute value used quotes.

After clarifying that the system had not made it sufficiently clear, the investigator pointed out the EUAccessionDate attribute in the table, and asked if the participant wanted to know why some were in the table, and some in the 'other' box. The participant did, and observed that “surely you’re going to want to use ‘Demonym’ for every country”, an attribute which, for the test data, was not given domain metadata. They then proceeded to add this domain and range metadata to Demonym such that it would appear in the table, although they initially expected “an interface for it”, and momentarily were unsure about its type. Editing Belgium again, they note that there is now a form field for it, “and it’s filled it in, and it should be removed from [the other section] I’m guessing—very neat!”

The participant spotted the duplicated character between the films, and decided that “he needs his own article really”. Having determined that the two articles are not “word-for-word the same”, they state that they would difference and merge the two in a real wiki, but for now assume that they’d done that and cut one of the sections, replacing it with the henchman’s name. They then pasted this into a new page, Jaws, the henchman. They look back at the transclusion demonstration as a reminder, and note that “they all had IDs, but I’m guessing that was added *after*”. After initially missing the type, and triggering a bug which hides the heading, they succeeded. They were surprised to see the content back in the page source when editing, but considered it “neat”, and would have done the other page the same way.

Overall, they commented that the system was “fairly easy to use once I’d got into it”.

11.3.4 Participant three

This participant was more knowledgeable about hypertext systems, and noted some points during the demonstration before the investigator could: the directionality of the links, that the embedded and first-class representations are of the same link so changes ‘propagate’ between them, and that edit-time transclusion changes are pushed back to the original, which was “very nice”.

First was the henchmen task. They noted the text differed beyond the first paragraph, and determined that it wasn’t “cleverly transcluded or anything, just part of this node, so I’d have to change it in two places”. Using the quick reference, they surrounded a section of text as the body of a transclusive link, and noted that this section had “links in already, which have anchor IDs, but I’m assuming that if I leave them there the system will fix them, cause they’re now part of the new node”. They saved this change, noting that it should look the same, but now the section of text was a separate node. Finding their new node in the relations table, they visited it and checked it contained the text. They then approximated merging the text from the other film into their henchman node, deleted it from the film, and checked that the links in the description still worked. They created an empty transclusion in the other film, expecting “the system [to] put in the rest of the text for me”. Noticing that below their shared section one film’s page had some additional prose about the character which they preferred, they edited the film and moved it into the transcluded section, which "should edit the transclusion and therefore it'll edit the other node as well". They reloaded the other film and the henchman node to confirm; it's "got everything. So that's exactly what I'd have wanted. It's got the text in both." They also clarified that they only share a subset of all the text about the henchman between the two films because the remaining text appeared to be talking specifically about their appearance in each film.

The participant made straightforward links to cake ingredients, using the quick reference to spot how to link to a page with a space in the title. They noted that they "can't use wiki text like I'm used to where I just put square brackets" (the MediaWiki link-to-anchor-text shortcut), or at least that the quick reference didn't show how. They considered if there were "any other links I'd do anything interesting with", but decided not. Prompted to consider generic links, the raised the issue of scoping: "'sweetening agent', it's pretty obvious that if you write those words, it means exactly that. I mean *sugar* you could generically link, but then it can mean other things." Generic links hadn't occurred to them because they hadn't used them, but they decided that they could have "easily" done so in "lots of these cases", so tried one. They looked at the example node for generic linking, and noted the instance property mechanism: "there's a list of attributes where it knows what I think I want, and an other box where I can type markup". While changing an ordinary literal link for sugar, in its embedded form, to a generic link to it, they noticed that the first-class view of the generic link has a target, whereas the embedded markup uses to. They paused to consider if that meant sourceQuery was right, and the investigator pointed out that the inline form is fromq. Again, creating a generic link inline linked it twice, which the participant noted. They viewed the link, and noted that it had a source *and* a sourceQuery: "so it's created a link that is both generic and specific", which was correct. They removed the source, and tried to return to the Cake article, but realised that you can't click on a `CONTAINS` query in the relations table, so went via the Homepage. This fixed the link, and they noted the subscript '2' in the relations table, correctly identifying it as the version. They then decided to add more instances of the word "sugar" to the text, "to see what would happen, because obviously the real cake node with all the text in would have it in"; they capitalised one.

This revealed a bug with the prototype, which incorrectly triggered the generic linking twice because it did not correctly identify that it had already linked to "sugar" due to the capitalisation change, yet still case-insensitively applied both instances of the link to the first instance of the word. Combined with the previous problem of a mixed generic/specific link, this caused some confusion as the participant tried to help spot the bug, and the investigator moved on to the next task.

The participant again chose transclusion to create a generic summary about the train articles. They used a dangling link from the Homepage to create their new page, to which they added headings and some basic text. They revisited the transclusion demo for a reminder, and noted again that they're going to need to make the summary of each country's trains a node itself, which will need a name. While looking at one of the country-specific nodes, they realized that "the text of this means it can't just be transcluded directly because it talks about the contents of the page". They started to break off two sentences either side of the troublesome self-reference as MultipleUnitsOfIreland.Summary.1 and MultipleUnitsOfIreland.Summary.2, using the namespacing, before realising that they didn't want the second of these for the summary. Saving, they noted that it "should look exactly like it does now—and it does, but I can see the transclusion at the bottom in the relations".

For the other country, they wanted the last part of the introductory paragraph, and the following paragraph, so constructed a transclusive link spanning the line break. The system complained (this breaks the tree model), and the participant "was worried something like that would happen": they removed the error from the markup, and put explicit paragraphing inside the transclusion. This caused 'deep paragraphing' errors, as they'd created a paragraph in the middle of an automatic paragraph, which caused confusion. After some contemplation, they went to the transcluded node via the relations table, and cleaned that up to be just two paragraphs. They then used the history to revert the country-specific node to before they made this change, then added an empty transclusion to the corrected summary. The paragraph split was lost, but the text was intact, so the participant finishes off transcluding the summaries into the general trains page. Asked about if somebody needs to make a correction to the summary, the participant notes that "they can edit it wherever they see it, and it will change in both, and when they edit any node, they'll see that it's a transclusion and that will make them aware that it will change in other places. So they can go to the transclusion and follow the relations to find out where it's gonna change if that's going to be a problem."

Upon seeing the Belgium node, the participant noticed the attribute and asked themself if they were "properties of Belgium, or the properties of *the node* Belgium". They decided that they were the former, as "if this were a link, they'd be properties of the link"; because nodes and links are different objects, "that's fine, it is actually conceptually logical. Which is nice." They edited the page and noticed the repeated attribute Language, and that "I want to add another one, obviously those boxes are all already there" (and full). They spotted the existing field for the fact to add and entered the date, having spotted the format in other date fields. Viewing the saved page, and wondering how the system knew which attributes to show, they clicked EUAccession in the attributes table. The participant determined that "it knows it's a property because it has a domain and a range. That's interesting." They noted that the date value was a link to a node that doesn't exist, "but it knows [that] is a date, because that's specified *in the node* EUAccession, which is very neat". They expected Belgium to have a class of some kind defining the attributes, and looked at Country, its type, but this only contained a short text about countries. With prompting, they realised that attributes are found by their domain, and experimented with this by changing the domain of EUAccession to another value. At first they didn't expect this to work: "presumably it won't let me because things are using it". Upon editing, they see that it has moved out of the table, into 'other', "because these are things that it doesn't have by virtue of its class, they're just other properties that anything can have. That's very interesting. That's cool."

The participant again considers transclusion for the London Underground task, because "the text in Transport in London is really summarising the other node, so it could do that by transclusions if it were the right thing to do in each context". They transclude out the sentence from London Underground about the newly-added fact, and add an empty transclusion to the other page. They shared just this single sentence "because the text of the other page is already a better summary in [its] context".

The shorter of the ambiguous Shelf pages is found and corrected as with the other participants, edited inline after using the relations table. The participant got the page name for the specific shelf type from the browser address bar while hovering over a link to it. Upon seeing the larger article, they attempted to find where the link to shelf is in the prose by using the relations table cross-highlighting, but it didn't not fit on the screen at the same time as the right paragraph. They clicked to edit the page and noted that "this is rather complicated—it's just occurred to me I could probably looked at the link itself, couldn't I". They found it in the relations table, and edited it, also considering that "this is only a specific link: [it's] only got that one anchor, so it can't have changed it anywhere else". Clicking the source afterwards did not take them back to the previously-ambiguous article as planned, however: they realised that it showed them just the contents of the anchor within that article, so they used the browser back button instead. They checked that the text was the same, but the link was now correct in the relations table and browser status bar when hovering. Finally they looked at the relations table on Shelf to confirm that it no longer had incoming links.

As concluding remarks, they wondered if there was a way to have partial paragraphs to solve the transclusion-across-boundaries issue: "it's a problem with transclusion as because you're often trying to address across the boundaries of elements, but it would be really annoying to have *two* separate transclusions just because they're in two paragraphs". They found the system "awesome: I love the fact that you can create relations just by writing text in, and that you can give something a property just by creating a node which describes that class". They contemplated whether there's a hierarchy of typing, so that one could specify a EuropeanCountry, and it would have subclass-specific properties, and inheritance: "the expressivity of the linking structure implies to me that it would work if I knew how, because it can do every other thing".

11.3.5 Participant four

During the generic link demonstration, participant four realised that if you were to remove the embedded form of a generic link, but leave the actual word in place, the link would come back.

The linked the word "food" from the Cake article because of the bidirectional linking provided by the relations table: "that means that when you get food, you'll be able to see cake as well, so that's potentially useful". After prompting, they may have considered generic linking as something that "would work too".

They created the general train article by creating a dangling link from one of the specific trains, to which they then added some lead-in prose, and headings by copy-and-pasting them. They looked for transclusion information in the markup quick reference, but resorted to the demonstration pages instead, copying the link markup. The participant was going to make up an ID, but as this was a new link they were prompted that the system will do this automatically if the ID is simply omitted, to avoid problems. They are stalled for a while looking back at the demonstration: "so *potentially* I'm not exactly sure *how* I control the link on [the transclusion target node], it looks like it *just* puts the text in". They determine that "maybe I want a separate node that's just for the [shared] text". Uncertain from which end to 'start' the transclusion, they put empty transclusion markup to a dangling common endpoint in both the general and country-specific nodes "so that they basically share content", possibly expecting an error. Specifying anchor IDs by copying link markup caused the system to become slightly confused. They spotted the link to their shared node in the relation table, and followed it, cutting and pasting from the other articles into the common one. The participant then asked which order to save pages in, and was reminded of the demonstration that transclusions 'push through' changes. This mattered because they had left the word "something" in the body of their transclusive links from copying from the demonstration page. They decided to save the merged text first, which caused it to become overwritten when they saved a "something", because "it pushed them through". After some conflict resolution, however, they re-saved their merged text, and reloaded the other pages, noting that "it's pulled that information back in; I think the two original pages *look* identical to how they were". Should someone need to make a correction, the participant said that they should look at the transclusion link and make an edit there.

Faced with the Belgium task, the participant declared that "I'm not familiar with how this works so I'm just going to edit, jump in, and copy the markup". Looking at the input boxes, they commented that "this is different to Wikipedia; this is interesting". They quickly spotted that there was already a textbox for the desired field, and that other dates had the ISO format, so entered the value. They also noticed that the word 'Date' was a link: "I'll have a quick look at that but I suspect this will. . . so it tells me how to format it, good. Then I'm confident that it's correct."

After checking the London Underground page for “anything weird like transclusions”, the participant added the fact as text to both nodes. They had contemplated using transclusion for a while, but the markup was no longer in the clipboard, nor was it in the quick reference.

They likewise made the change to the henchman in two films by editing both pages. The participant stated that if they “were to be more of a curator”, they would have pulled out content that’s the same; “the Wikipedia way to do that would be to write a bot”. They noted that the text was “identical in parts”, so one could “write a little script that went through all the nodes and found similar text”. Asked to expand on that, they suggested “a helper application that presented me with a gazetteer that I could choose from”. It would perform fuzzy text matching, and as an example it might report “‘*I found five Jawses, do you want to... transclude into a separate node and combine them?*’ And then I would be like ‘yes’ for these two, and then ‘no’ for the film ‘Jaws’ ”. In this case, they or their tool would “curate” a piece of text about the henchman Jaws in general as its own node, then transclude that into both films.

The participant used the relations table to get to the ambiguous shelf nodes, and also to get the names of the different shelf types. They made inline edits, and tested success using the relations table: “*presumably*, if I go back to the shelf page... then the relations will have disappeared hopefully; so there’s only out relations, and in relations are the Homepage, so that’s good”. When asked for their reaction to the source of the larger of the pages, the participant said “there was a lot of text, and the markup for each link is—really it’s quite verbose... in MediaWiki, if for example the link name is the same as the text you can—all you have to do is put some brackets around it... and so that, you can actually read that inline”. Their suggestion for another way to make the changes would be for search-in-page to work for editing.

11.3.5.1 Remarks

This participant made an interesting suggestion about generalising the problem of there being duplicated content on the wiki *at all* via ‘fuzzy’ techniques to try to identify it, so that it can then be unified by the provided hypertext structures. Such functionality is outside of our scope, but a possible future research area for ‘tidying’ large-scale wikis.

11.3.6 Participant eleven

The participant used the quick reference to link a few nouns in the cake article, testing using preview after the first to check they were doing it correctly. Unwilling to mechanically link the whole paragraph, they were prompted to consider an alternate approach, and tried generic linking. They first looked in the quick reference then, finding it omitting this, at the demonstration pages, where they copied the embedded form of the generic link.

There was some confusion about the target of the generic link, which seemed to be based on uncertainty on where the link's embedded form should be added: "How do I know where it's going *to*? Cause don't you want the other page, say Food, to be the page that adds to this page?" When editing the article to add the generic link, they commented upon the system-added identifiers for their existing regular link, but then replaced this with a pasted generic link for which they provided an explicit ID. They initially also replaced the content word with the copy-pasted example, but spotted this mistake in preview. However, upon saving, the link vanished, as they had not removed the special reserved anchor ID the system uses to mark temporary generic link anchors. They initially suspected that the problem was that the target node did not exist until this was pointed out to them as a quirk of the system. Upon retrying, this time removing the IDs "because presumably it makes one", and commenting upon how that would be more consistent, the link worked.

After editing the Shelf disambiguation page to note that the links there "look correct", the participant used the relations table to find the shorter, river article. They determined the correct target from context, then copied the embedded form of the link from the Shelf source into the river page. Commenting on how that was "presumably the traditional way of doing it", they theorise that "there is a slightly easier way of doing that, because that's just a link, with an ID, which I can go and edit". They clicked the link name in the relation table for the other ambiguous link, wondering where it would go to, then the article itself for the context to determine the target. Editing the first-class representation of the link, they paste the correctly disambiguated node identifier into the attribute table, having copied it from the address bar. They click the link source, "hoping to get back to the page from there, because ti said source", but this takes them to the anchor alone. The participant realises "there's no way of getting back from that anchor to the page itself", but expected to be able to because "surely the anchor has a relation to the page". Having got back via the specific Shelf type, they did state of first class linking that they "liked that feature".

With the villains task, the participant identified that the common section had been copy-pasted, and that there were differences which would have to be resolved. They decided "to include that, because it's the same article on both pages", but needed to be directed to the transclusion example to remember how. They cut the section out of one article and pasted it into a new page, created by editing the address bar. The participant copied and pasted the link from the example nodes, removing the IDs, and saved after preview didn't work. To add the transclusion to the second film, they re-edited the first, but observed that "the transclusion now includes all of that [content], so it's not as easy to copy the link", although it would work. They commented on race conditions, and were told that the system performs locking to detect them. Finally, they changed a required fact in the shared character article, then checked that this change was reflected in both films.

The participant assumed, for the Belgium infobox task, that "attributes, presumably, are just triples, included from somewhere". They saw the attributes form in the edit view, and wondered how to add a row, assuming that they had to use the 'Others' box. They added 'EUAccessionDate', using a date format they had seen in one of the other values, and saved. Noting the alphabetical sorting of attributes, they found their addition, and commented that "that was fairly easy; painless". The investigator commented on the table/'Other' distinction, and that the participant had previously used the table to edit the attributes of one of the Shelf links. Asked if they knew where there was a distinction, the participant guessed that the table attributes were "links, that are put in somewhere else, between pages and things, like on the Albatross (Shelf task) page. . . so you've got a root node relation or something, [and] you can presumably put these in somewhere else." Prompted to look at one, they noted that "it's an attribute; how do you *create* one?", to which the investigator showed that it was a regular node and, after some confusion, the type/domain matching.

For summarising the Multiple Units, they postponed immediately creating the general node, as they said "we want to create some form of relation between these things; see you've made me think in terms of your wiki now as to what you would do, because obviously you do want some relations between these things". They looked at the Shelf page for inspiration, because "it has all the links". Eventually they create a new page, MultipleUnits, which links to both specialisation pages, and look at the relations table. They refresh the specialisation pages and look at their relations tables, noting "we now have a relation".

To add the summaries of the specialisations, they ask if they can transclude part of a page. Told otherwise, they are reminded that they did effectively do that for the villians task. They remembered that they had split off the shared subsection, about a character, into an article, but questioned why you would want to do that in this case; "can't you have subaddressing"? "You would have to create a page that was just a summary of [the specialisation] page, and then include it in this page and the [general] page; and that sounds like a lot of work". The participant went on to explain that they wanted sub-component addressing because "clearly on these pages you've got this first introduction paragraph before the first title and most Wikipedia pages have these", and they wanted to "transclude that exact section" without having to think up a name for a new nodes, create it, copy-and-paste the content, save, edit the other page to remove the content, and set up the transclusive link. They followed this process for one of the specialisations, and the investigator prompted them to think of a way to avoid the explicit page creation and copying. After guesses at using generic linking, or sub-component addressing by section heading, the participant was shown that the transcluded node can be created by the 'push changes' effect of edit-time transclusion.

They commented that "you've done it backwards. . . you've created the page by telling it 'this is the content to be on that page"', and that they had meant something like that by naming a section. The participant reasoned about the Semantic Web meaning of links in terms of triples, but there were terminological difficulties, for example with 'id' in the XML within-component sense.

They added the fact about the London Underground directly to both that and the Transport In London articles, but consider that "it's a fact, so technically it should go in an infobox"; also because it is not subject to change. They would have liked to have been able to transclude or share the attribute value between the pages. They also decided to go ahead and unify the summary text between the articles using transclusion "backwards", as they had been shown in the previous task. When creating the transclusive link in the section node, they deliberately provided the node's content to see how the system would handle a new anchor with new content for an existing transclusive link, and found that it overwrote. They initially suspect that the other text was lost, but look at the history for the transcluded node and find it intact, from which they retrieve and merge it. While not an edit conflict (there was no concurrent editing), they point out that two editors here could have inadvertently specified the same link target and "been editing the same page but there's no warning about that". (This is a risk which only exists at link creation, and is at least partially because in-lined content loses derived-from version information in the current text serialisation.)

Concluding, the participant stated that they "like the system: in terms of those people who are thinking semantically about things it's a very good way to see a wiki going". They felt that the wiki would help for a project they work on, as the contributors "*constantly* edit our wiki pages and try to organise the wiki and I have a feeling that some of these things would help us out greatly because it would get rid of a whole lot of these duplication of data". "Because templates are all well and good but they're not brilliant for just being able to say "include this here include that there", which this seems to be better at doing that". They also liked having "all the relations between everything", and that links were first-class, "because then you can re-use them, and I liked that as well"; a "nice piece of work".

11.3.7 Participant seven

Participant seven grasped immediately that "links are pages as well" during the demo, and described first-class link nodes in terms of intermediate database tables for many-to-many relationships. They also recognised during the transclusion demo that editing the inlined version would affect the transcluded page.

The participant was familiar with disambiguation pages, and looked for a "what links here" feature, as with MediaWiki, before spotting the relations table. They appreciated that the relations table cross-highlighting worked in both directions. They looked at the ambiguous 'river' page, and copied the link *text*, not target, from the Shelf page for the type of shelf it should link to. The participant found the link manually within the river page after finding that the browser would not search within text boxes, and pasted the link text as the destination. They were concerned about whitespace, noting that it appeared to be used as a delimiter, but took no action about this at first. However, they were also confused by the markup for the link type and text, `Type=Link Shelf`, and changed `Shelf` there "for good measure". Because the markup was now of the form `[link to=Shelf, West Yorkshire type=...]`, the system interpreted it as a default-type link to a node Shelf, (note trailing comma) with a link text of "West Yorkshire. . . ", which was not the intended effect. The participant determined that they needed to quote the attribute value, and sought an example (they did not notice the quick reference throughout the experiment, possibly because of their Wikipedia experience, which uses this space for copyright notices). They left the type and tried again, creating a link to the incorrect but intended target Shelf, West Yorkshire; they also tried momentarily changing the type to this value. Prompted to consider what they had copied, the participant realised that the target page title was not the link text from the Shelf page, and corrected their mistake. In the process, they "got" the markup, and realised that the `Shelf` at the end of the element was the link text, not part of the previous attribute (type), so left it alone. After previewing, and testing the link destination, this was saved as correct. They noted that it showed in the relations table outgoing, "which is nice", and hypothesised and confirmed that it was now no longer an incoming relation on the Shelf page.

Moving on to the next ambiguous page, the participant lamented the absence of an "edit section" feature, made a mental note of roughly where the ambiguous link was in the text, and reluctantly went to edit the whole page. Frustrated at the browser's inability to search in textboxes, the investigator prompted the participant to consider if finding the link there was necessary. The participant decided to be "stubborn and determined" and found the embedded link by comparing to the view of the page in another tab, and updated it. After this, they contemplated that links are bidirectional, and that "there's a link in the middle" which they could have edited, but it "didn't occur to [them] at the time". They wondered how to get at the "link in the middle", as clicking the endpoints in the text takes them to the target page, before considering the relations table. The participant clicked the word 'Link' in the table, which was the *type*, and was confused to encounter documentation about links: "I was hoping to see some page names". Backing up, they eliminated the columns for the source and target, and quizzically tried the "via" column, which yielded the expected result. They determined that they could see a source and target, and could have edited this instead.

Looking at the two films with the shared villain, the participant checked for "templates or anything going on, because that might be a way to reuse [text], because I don't really want to edit two things, to add the same information in". Although not quite the same, they contemplated creating a new page for the villain, although they wouldn't necessarily do that on Wikipedia. Here, however, they were "bold" and created the new page by editing the browser address bar "because that's how some other wikis work", having not seen a "new page" button. They moved the section from one film, leaving the character heading on the original page, because it "doesn't really make sense" on the character's own page, before changing their mind. The participant suggested "a nice thing to have" would be to replace the HTML-style `[heading level=2]` markup with nested headings with implicit level, so that transcluded pages would get appropriately sub-levelled headings by context—"that would be quite cool". Momentarily distracted by the idea of generically-linking the shared character's name, the participant sought the transclusion example and copied the embedded link into one of the film pages. They "[didn't] want to specify what the text of that [target page] is", and tried leaving the link element empty, in the "hope it doesn't delete it all". Having asked if they could just ignore the IDs, they were told to remove them: "I wasn't sure what to do about those". (Previous participants had already shown that this creates confusing hyperstructure, and the investigator was wary of running out of time.) Despite some minor formatting losses (bug), the transclusion worked, and that participant was "pleased with that". They edited the film page again to copy their link to the other, suspecting that "the text is going to be enormous in here now". It was, and they "[could] see how it could be useful to edit in place. . . as a convenience", but found the whole of the character page being transcluded to be overwhelming, and "kind of [wished] that [they'd] copied and pasted that sort-of 'raw link' that [they] had before [the body was inlined]". They instead took the head of the link, leaving the body, and copied that to the other film, replacing the text there (they would have merged in a real scenario), which worked.

Faced with the Multiple Units task, the participant created a new page with a heading for each country specialisation, and looked for suitable summary text. They noted that "normally when I'm doing summary paragraphs they wouldn't be identical to the paragraphs from the page; so I *could* use transclusion here, because I'm copy and pasting, but I'd normally try and keep the text more succinct in the summary pages, in which case because the text is different I can't use transclusion". This given, they copy-and-pasted some text from each specialisation, and saved to checkpoint their progress before setting up links. They momentarily contemplated putting the links in the headings, but "that could get interesting, because it's already in a tag" (element), which was "daunting". They found a link from Shelf to base theirs upon, "because I can remember that I used a link, and it didn't have the same text on it"; while they felt they understood it, re-using an example was easier. For the link to the second specialisation, they tried leaving the link element body empty, to see if the system would imply the target's title as anchor text. When this failed, they correctly assumed that the system interpreted it as a link [anchor] with no text, "which is not particularly useful"; "probably not something you'd normally want". They did not notice that it had still affected the relations table until prompted. After correcting the link, they were happy to confirm their hypothesis that refreshing the specialisation pages now showed the incoming links: "it's a bit like 'What Links Here' in Wikipedia, but on the same page".

For the Belgium attributes task, the participant remembered that they had seen an attributes field that they previously mistook for an edit comment field. Looking at the table and 'other' box, they determine that attributes "look like they're sort-of real things as well", before finding that the required attribute was in the table with a blank value, "which [was] good". They started typing the date value in the (incorrect) format spoken by the investigator, before considering that they should find out the desired format, and uncertainly clicking the 'Date' (attribute range) link next to the field. This documented the desired format, which they considered "excellent". With the page saved, they experimented with clicking the value, giving the page about that day: "excellent". They also noted that the link colour hinting for page existence was wrong in the attributes table, a known limitation.

Previously confused by apparent attribute values without keys when editing, the participant looked again, and eventually determined that they were multiple values: "a bit more formatting and I would have got that". They contemplated how to add another value, as there was no 'add' button, and they did not suspect that comma-separation would work, before trying the 'Other' box, not expecting the merge to work. They were surprised when this succeeded. Prompted to explore why some attributes were in the table, and others not, which the participant was initially curious about, they clicked another attribute key. Reading it, they stated that it was not a "regular page" (encyclopædic article), and understood the domain and range predicates. Reminded that Belgium had a type of Country, the domain here, they worked out how the system performs the mapping to suggest attributes. They "*might* have expected" to see, on the Country page, a list of the properties mapped this way "because this is almost like a template for what would go in your infobox for a country".

The participant edited both the LondonUnderground and TransportInLondon articles to add the required fact. The investigator stating that they had copy-pasted text for the audio recording, the participant explained that they "can't give [that fact] it's own article, because it's a single statement".

For the 'cake' task, the participant immediately thought of generic linking, although this because they wanted to try something they had been shown but hadn't yet used. Initially they were going to link "sweetening agent", "because that's a very sort of scientific thing isn't it", but the investigator intervened to observe that only single words work for now to save time. They instead created a page for Gluten, and edited at the generic linking example. Instance-property editing was unhelpful here: the participant was hoping to get a box with all of the attributes, which they could copy, then paste into a newly created link. However, because the attributes were in a table, this wasn't easily possible.

They created a new GlutenLink page, and first set up the type as Link, after which they previewed. This somewhat unexpectedly provided the instance-property editing form for their link (“lovely”) into which they copied the attributes from the example. They noted that the example link had content text—“why would you put text on a link?”—but determined that they did not need to reproduce that. Refreshing the Cake article, they saw that their generic link had worked, and was in the relations table.; they intuited after a little initial surprise that the subscript ‘2’ indicated the version.

In conclusion, the participant “liked the ideas of the being able to edit templates (transcluded pages) inline”, but felt that it needed to be optional, or only “if it’s more than five words or something”, “because when I did the [villain] thing it put the *whole* [transcluded article] in the article I was trying to edit”. They didn’t want this because it was “too much” text, and they wanted “to see the article I’m editing, I don’t want to necessarily see entire other articles if I’ve transcluded them”. Although the view contained this text, and the article previously contained it, “while [they were] editing it [they] think that just got inconvenient at the time”.

They liked being able to see outgoing links on the current page in the relations table, and also that they didn’t have to click “What Links Here” to see incoming ones: seeing both directions together “*in a table* is quite useful”. However, the current table is “a bit *raw* at the moment because these links are all very cryptic underscore 2 things” that the participant suspected would confuse users.

Finally, they liked the instance property editing: “it was nice to be able to just instantly see what I had to type in... it was all there, it knew it was a Country: I could just go straight in and edit it, and not have to go fishing about and reading template documentation that’s just plain text, which is what Wikipedia would be”.

11.4 Conclusions

11.4.1 Common observations

As with table 6.1, table 11.1 groups the general approaches against each participant.

Participants were generally positive about the system and willing to explore its functionality. They demonstrated some of the same usage patterns as we discovered in chapter 6. Most significantly, they still learnt by example, seeking previously encountered uses of a feature to derive from in favour of writing markup from documentation. Some participants made use of the quick markup reference under the page for this purpose, as it took the form of self-describing examples. Participants still previewed intermediate stages of page creation, and this stresses that the preview feature, which is no longer completely

Behaviour	9	8	10	3	4	11	7
Same villain in two movies							
Edit both					✓		
Transclude section	✓		✓	✓		✓	✓
Transclude just fact		✓					
Specific and general London Underground							
Edit both			✓		✓	✓	✓
Transclude section						✓	
Transclude just fact	✓	✓		✓			
Disambiguate Shelf links							
Found backlinks via relations table	✓	✓	✓	✓	✓	✓	✓
Edited embedded form of link	✓	✓	✓	✓	✓	✓	✓
Edited first-class form of link unprompted			✓	✓		✓	
Considered first-class link as alternate approach	✓	✓					✓
Summarise two types of train							
Transclude parts	✓			✓	✓	✓	
Copy parts		✓	✓				✓
Add links to cake introduction							
Tried/wanted empty, default-text links	✓	✓	✓	✓			✓
Considered generic linking unprompted							✓
Considered generic linking as alternate approach		✓			✓	✓	
Rejected generic linking as unsuitable			✓				
Tried to create generic link embedded	✓			✓		✓	
Corrected embedded generic link to work				✓		✓	
Created generic link in first-class form	✓	✓					✓
Add fact to infobox for Belgium							
Added fact	✓	✓	✓	✓	✓	✓	✓
‘Other’ box instead of IPE			✓			✓	
Sought to add row to IPE table		✓	✓			✓	✓
Date format used by example	✓		✓	✓	✓	✓	
Date format used by range documentation	✓				✓		✓
Explored range/domain of attributes	✓	✓	✓	✓			✓
Non-task specific							
Used markup quick-reference unprompted	✓		✓	✓		✓	
Complications from link IDs in source					✓	✓	✓

Table 11.1: Summary of participant behaviours for tasks on Weerkat

functional in the prototype after the extensive changes required, is important, even if it is not novel.

The relations table was frequently used, and participant reaction to it was positive. It expedited navigation when the desired target was already known, which may be attributable to its increased link density and alphabetical, rather than prose, ordering. Participants also seemed confident in their understanding of the hyperstructure presented by the table, and all understood the cross-highlighting between table rows and link anchors in the content. Its primary problems are that it cannot cross-reference if the anchor and the table do not fit within the browser window without scrolling, and that the columns other than source and target are not sufficiently intuitive. Even participants seeking to reach a first-class link could be led astray by the column full of the repeated text ‘Link’, which is actually showing the types of each link. There is also a navigational issue in that there is no easy way to get from an anchor to its parent page, which means following the source attribute of a first-class link is currently of limited use. The tooltip when hovering over an anchor, stating the link ID responsible, was largely ignored, and could not be used to *reach* the first-class link anyway, as tooltips are not interactive. This made it largely useless.

The heavier link markup required by the system to track versions of links over time and properly synchronise anchors, with identifiers, was somewhat problematic. While it did not entirely obstruct any participant, it did increase the ‘bulk’ of page sources, which hindered the task of scanning through them, and was occasionally confusing. A recurring problem is that *users copy embedded links* as part of the process of creating new markup by deriving from previous examples. Identifiers complicate this process, as they must be removed to create new links, else the results can be highly confusing: the participant instead changes the previous link, and may assert duplicated anchor IDs. Some participants wanted per-section operations, such as sub-component addressing by section boundaries, and Mediawiki style section editing: the latter would partially help by restricting the scope of source to seek through when finding an embedded link.

While first-class linking does not come naturally to all participants, once intending to make use of it, everybody was able to modify a link in its first-class form to point to a different target. To some extent, we expect this to be a simple inertia/experience issue: first-class links are virtually non-existent in current popular, public hypertext systems, and greater exposure to the feature should naturally lead to its increased use.

Transclusion saw use by every participant, and some realised that the edit-time ‘push back’ effect meant it could be used to ‘mark out’ sections of a page by splitting them into separate, transcluded nodes. Response to content-sharing was generally positive, although the approach of creating the transcluded page manually could become cumbersome. Shorter content fragments were more disputed: while one participant created a shared page consisting of a single sentence, others were uncomfortable with the idea of creating pages which were not standalone to some degree. In some cases, smaller fragments caused problems by spanning branches of the document tree, which did not fit into the hierarchical DOM used for node content.

The instance property editing mechanism works, and was also viewed positively by participants. However, it does not distinguish empty fields well enough, as some participants missed them amongst the populated ones. A contributory factor could be because the editing view of attributes is similar to the normal view, except that the latter naturally does not show missing attributes. Participants wanted to be able to add rows to the table, although some determined that this could be done indirectly via the ‘Other’ box.

An unexpected cost of the IPE interface is that is complicates the derivative editing process: attributes which are in a table are very awkward to copy into a new node.

Generic linking was more problematic. The natural approach for most participants to create a new link is to write it inline, as is the model for HTML-derived systems (like Wikis). However, this does not work very well for generic links: conceptually, they cannot be placed somewhere in a document, as their endpoints are determined dynamically. In the case of this system, this caused problems where hybrid links were created with both generic and literal endpoints: this is very confusing when the literal endpoint splits out as an anchor subnode, and the generic link creates virtual anchors to match upon content. When participants created generic links directly as first-class entities, they worked as expected. The concise, element-style vs. extended, predicate-style terminology for link attributes was also problematic here, as participants had to map between the dissimilar `sourceQuery` and `fromq`.

11.4.2 Comparisons to Mediawiki

Compared to Experiment II, participant performance has improved. Every participant was able to complete every task, and what problems were encountered were either due to exploratory use of advanced features, or bugs caused by the immaturity of the prototype. We were surprised to note the extended use of transclusion, beyond that suggested by the previous study.

11.4.2.1 Shared villain (Jaws)

As a large, potentially standalone section used by two articles, we expected transclusion to be a natural solution, and participants used it. With Mediawiki, this was not a practical solution, as it would have involved creating the character as a template, also concealing it from the edit views of the film pages.

11.4.2.2 General and specific (London Underground)

We did not expect the system to perform significantly differently here, as participants for Experiment II largely agreed that the different contexts required different texts. While some held this for our system, others did indeed use transclusion for information-sharing between the pages.

11.4.2.3 Disambiguation links (Shelf)

The simple user interface improvement easily facilitated by first-class links—the relations table—was a huge boon here. With Mediawiki, some participants struggled to find the articles, needing to activate a "What Links Here" feature in the side panel. In Open Weerkat, every single participant was able to rapidly find them.

In addition, some participants were able to improve their performance further, by realising that they could edit the first-class links directly, and avoid the need to find their embedded versions, which can be time-consuming. We believe that this could be a significant advantage to the 'many small edits' 'fiddler' wiki editors identified in Experiment I who perform this kind of maintenance task, as their less-casual role should make them more inclined to discover and use this capability.

Conversely, the increased verbosity of the link markup *did* slightly impede participants seeking through the markup to find the embedded link.

11.4.2.4 Summarise specialisations (Multiple Units)

Again, we were not expecting significant changes, as our previous study suggested that that the differing contexts would proscribe content sharing. Most participants, however, did seek to use a summary shared between the overview and specific pages. In addition, the relations table was again useful, as the act of linking from the overview to each specialisation also effectively navigationally related each specialisation: they each gained a backlink to the overview, allowing it to work as a hub. In Mediawiki, this kind of organisation requires the explicit use of categories.

11.4.2.5 Link prose (Cake)

Simple link creation *is* more complicated in Open Weerkat than Mediawiki, and this was encountered throughout the tasks, mostly when deriving links from examples and having to strip out superfluous attributes. Links created 'from scratch' were not so much complex as *verbose*. This task particularly highlighted the problem of requiring an explicit target and anchor text, requiring participants to duplicate words from the original plaintext.

Some participants were able to use generic links for this task, but within a single article this did not save labour: the process for creating a generic link is more involved, not least as there is no special case for creating a link for a given word to the node of the same name, with an autogenerated link name. It is a limitation of the experiment that we did not discount this against the link not needing to be explicitly created on other pages, but some participants were concerned about this (currently) boundless scope and how it may affect pages other than the one they sought to change.

11.4.2.6 Add attribute (Belgium)

Instance property editing proved here to be a substantial improvement over the Wikipedia infobox templating approach. Again, while some participants were unable to complete this under Mediawiki, all of them could under Open Weerkat. In addition, all participants added the date value in the correct format, despite having the fact presented to them in conversational speech. Some participants were even able to determine how to define new attributes which would appear in the suggested list.

11.4.3 Implications

A major limitation of the prototype system is that it has a web-based user interface, which is constrained to the capabilities of an HTML text area. This means that pages *must* serialise down to a string of plain text for editing, and that all context about each element to be preserved must be part of it, regardless of the importance of its visibility to the user. A WYSIWYG interface—either thick client, or some form of 'rich' web page—could help greatly by concealing details such as link identifiers by default as 'rich objects' in the page source. It may even be possible to match this up more closely with the reading view of the page, thus helping to avoid the disorientation of switching to edit mode: compared to desktop word processing, this is currently a strange interface relic of wikis, which this study and system has highlighted because of the increased markup verbosity.

We can make some improvements to markup usability even without such major changes. The Link type is just noise and, given that it conveys no semantics and it or a subtype are an implicitly-added requirement of all links, it can be stripped from serialisations without significant consequence. We persist link identifiers so that we can version links with semantic accuracy: so that we track 'the same link' over time. While this is theoretically good, we have found that the cost of tracking identifiers is high, and they could be dropped. Instead, the system would need to identify links which matched the base version of the page, and whose endpoints are now missing from the derived one.

Participants attempted or desired to use the historical wiki shortcut of a link containing just a page title to use it both as target and anchor text: we should better support this pattern. While it removes some semantic flexibility, links with empty anchors or no targets are slightly obscure edge cases, and implying one from the other during creation would allow for this shorthand. Care should be taken to not modify links with query-targets, but no literal targets, however.

Query-target, or more specifically in this prototype, *generic* links were, however, highly problematic. Participants required guidance to both consider and to succeed using them, and we must consider that a real-world situation, without the appeal of novelty, reduced consequence environment, and experimental incentive to be persevere more, they would be less appealing. Given that is is largely impossible to correctly create a generic link inline, it may be best to outright not support this approach. Combined with the implementation complexity costs, query-based links may not be suitable for general use, and best left as a special-case, expert-only feature.

The problem of instance property editing complicating first-class link copying could be resolved by providing an explicit node-copy feature. While it is tempting to also consider this derivative in a versioning context, it is worth remembering that the new node is only related to the old by the rather arbitrary connection that the user found it first as an example. Instance property editing in general would benefit from user interface improvements which allow the table to be dynamically expanded, as well as the aforementioned highlighting of blank, suggested fields.

The next version of the model should give greater consideration to the subdivision of pages into sections (for simplicity, the current model does not). An initial approach could be to consider each section to be natively transcluded into the page, as with anchors, which allows for each to have a full identity. This then leads to, as with ‘pushing out’ transcluded regions, a form of effective sub-component addressing, without needing to subdivide URIs. By giving sections a `heading` attribute, we can automatically generate headings to the appropriate nesting level depending on context, as alluded to by participant seven.

Transclusion itself was largely successful, but for the desire to share regions spanning multiple DOM branches. This is a complicated problem, as such regions must be separated out into standalone nodes, which then get transcluded back in at the point of an anchor in the DOM. An initial approach may be to mark these cut regions as ‘sticky’, such that they attempt to reattach to adjacent DOM branches, but complexity can rise very quickly, especially if the region also begins and ends at dissimilar tree depths.

Overall, we have demonstrated that the system successfully implements useful hypertext functionality which can be used by untrained, if adept with wiki systems in general, users. In the final chapter, we look to how to next develop the Weerkat model and system to help further its benefits and mitigate its costs.

Chapter 12

Conclusions and future work

12.1 Introduction

In this work, we have studied notable hypertext and semantic wiki systems in chapter 2, and sought usefully meaningful overlap between the two fields in chapter 3. This led to a set of proposed improvements in chapter 4 which could be applied to semantic wikis to enrich their utility and reduce the cost of hyperstructure maintenance. To show the scope for improvement, we performed a macro-scale experiment in chapter 5, and then prioritised the possible improvements with a micro-scale experiment in chapter 6.

We have, from this experimental work, devised a model which encapsulates the functionality of open hypermedia in a semantic wiki context in chapter 7. In chapters 8 and 9 we have addressed the implementation practicalities of, and reported upon our experience with, building on this work to create a prototype system by which the claimed benefits of hypertext can be tested. As a concrete example of its use, we applied this prototype to real-world use-cases brought up in the introduction in chapter 10.

Finally, we tested the system with untrained users in chapter 11, and concluded that the system had advantages over conventional, web-style hypertext, wiki systems.

12.2 Conclusions

12.2.1 Wiki editing patterns

We have performed, presented in chapters 5 and 6, a pair of studies on the real-world patterns of editing on popular wiki site Wikipedia.

The first experiment was a large-scale quantitative analysis of the evolution of a subset of pages over time in terms of the kind of edits made, classified by specially-developed software. In particular, we assessed which edits affected the text content of nodes, and which affected the navigational linking (links, categorisation, and such). We found that *twice as many* edits only changed links than changed content and that approximately one edit in ten was maintaining indexes of pages. This demonstrated that significant effort is being spent on maintaining the hyperstructure, and in ways which can potentially be improved by richer hypertext support.

The second experiment was a complimentary small-scale qualitative analysis of the observed editing patterns of a set of Wikipedia contributors. We captured both unguided, 'natural' editing that the participants made during a period on the live Wikipedia system, and a more detailed analysis of their interactions with a set of synthetic tasks on a high-fidelity simulation of the site. This was used to inform our design process with regards to which potential hypermedia features we could most expect to yield observable benefits, and also to ensure that our work was well-grounded in real practices. In particular, our attention was drawn to the important of learning-by-example; implicit collaboration by deliberately leaving articles in a strictly inconsistent state, knowing that another, more knowledgeable, editor would complete them; and the importance and distinguishing effect of context with regards to what can be semantically considered the same text. We also gathered information on how words are linked to their eponymous articles in Wikipedia practice: there was a consensus amongst participants that such links should generally exist, but only once per term, and that there was an optimal link density to achieve in readable body text.

From this, we were able to assess the requirements of a richer hypermedia wiki: that permitting 'broken' hyperstructure, such as dangling links, is important; that level-of-detail adaptation is likely to be negatively affected by the need to tailor text to its context; that transclusion is likewise affected in some cases, but still useful for sharing larger, more stand-alone, sections; that the existing mechanism for 'semantic' information in Wikipedia, the infobox, as scraped by DBpedia [80], is significantly complicated; and that the heuristics for determining if there should be a 'generic' link from an term to its article vary between editors but do have some common constraints.

12.2.2 Implementation of hypertext in wikis

We have shown that hypertext features can be combined with the semantic wiki paradigm by defining a unified model in chapter 7. This model defines what we term an 'open semantic hyperwiki': a wiki where every link has a first-class identity as a node, and where every node is a semantic web resource. From these basic building blocks one can build n-ary links and transclusions. We define an RDF mapping of the model, suitable for use in the world of Linked Data, including support for distinguishing between the subject topic of a node, and the identity of the node and its content itself. The model also supports multiple representations of a node—that is, a variety of different data formats for what is conceptually the same content—and defines semantics for referring to versions of the node through its history that are consistent with the separation of node identity from that of their subjects. Finally, it supports parametric nodes via an 'instance space' approach, which allows for parametric links, and is expandable up to fully generic and functional links.

We have also worked through an implementation strategy for this model, described in chapter 8, with an eye toward efficient and flexible implementation. Further, we have demonstrated that the model is practical to implement by beginning to do so with a prototype which covers many of the most challenging aspects of the system, including the edit-time presentation of transclusion, and bidirectional support for generic links.

Chapter 9 covers our experience in detail; of particular note is the cost of generic linking to a modularised and efficient design. Generic links necessarily involve several otherwise disjoint areas of the system, and this maintenance cost is also met by an increased computational burden, even with careful use of indexes and caching. We believe that it would be possible to implement fully-functional linking, as it is a relatively small developmental step for the prototype (requiring the definition of additional link matchers), but we have reservations about the computational complexity and scalability costs.

12.2.3 Utility of hypertext in wikis

In chapter 11, we have experimentally compared our system against the previous user experiences with Mediawiki. This allows us to address the hypothesis of the thesis: that these hypertext features assist users in managing knowledge collections in wiki systems. We believe we have found considerable support for this hypothesis from the improved user task performance observed, and have also collected useful data to guide future research in the area.

In particular, despite our earlier study suggesting that contextual adaptations would impair the ability to re-use content in multiple nodes, participants made widespread use of transclusion. We also discovered the limitations of our simple, logical DOM-tree-merging implementation as users attempted more complicated re-use; we know now to focus on the complex problems of supporting regions which span branches of adjacent trees without containing any common roots.

The 'relations table', made trivial to implement by the use of first-class links, and originally introduced to the prototype as a debugging aid for such, proved a popular way to navigate the typed graph of data of which the wiki comprised. First-class links themselves were also useful directly to some participants, as they were able to use them to perform certain kinds of wiki maintenance tasks without the costly process of locating the embedded form of the link within a large document. Conversely, along with its implementation issues, we found generic linking, when presented as such, to be hard to use. In particular, the flexible and orthogonal model led to users creating complicated, hybrid links with a mixture of literal and generic endpoints by accident.

Finally, informed by this experiment, chapter 10 provides real-world examples where this model and system can be applied. We cover a range of domains: beyond the general encyclopædic context drawn from Wikipedia, we address enterprise knowledge management, software development documentation, and community systems. Each is presented with a mapping of how the current document set can be represented in the system, and how this can assist users over the conventional wikis currently employed.

12.3 Future work

From the conclusions of Experiment III, we know that there are usability problems with the existing generic linking which must be resolved. On a non-research, engineering note, we also realise that importance of updating the preview feature to function correctly for real-world deployment; and that ideally the system should be equipped with a WYSIWYG interface.

12.3.1 RDF mapping

While our model defines an RDF mapping in section 7.3, we do not currently export this. Evidently, to be good Semantic Web citizens, the implementation needs to be improved to make our data available as designed. A simple RDF export is conventional for semantic wikis, with a SPARQL endpoint also being a desirable publication mechanism.

Our work on explicitly defining representations behind identifiers in section 7.4 means that the system can also work as part of the web of linked data, as our resources are retrievable with appropriate metadata available. In addition, our XHTML serialiser is currently stripping out type information when creating hyperlinks for display, but this behaviour could be adapted and improved to instead generate XHTML+RDFa (inline RDF annotation, covered in section 2.2.5.2), preserving our link typing information for other systems to consume directly from the output web pages.

12.3.2 Other media types

Weerkat is currently a text-only hypermedia system, in that each node is expected to contain a tree which the implemented renderer will transform into XHTML. This is not, however, a hard invariant of the system, and node content could potentially be of some other type, such as an image.

The main consideration for such types is the need for embedded anchors, which in the text case are supported intrinsically in the DOM, so as to alleviate the editing problem. For arbitrary types that cannot be reasonably expected to be edited the wiki itself or some a Weerkat-aware editor, the anchors cannot be part of the actual content stream and shown as explicit markers to the user. In this case, the anchor details must be stored alongside the content, in the same manner as which we store node attributes. This allows us to have anchors in the node while supporting arbitrary existing content formats, but comes at the cost of making it less obvious to the user where those anchors are while manipulating the content.

12.3.3 Advanced versioning

The current Weerkat versioning model is the simple, linear one common to wiki systems. We feel that adopting a more advanced versioning model, adapting lessons and designs from software development version control tools such as Subversion[1] and Git[2], may offer advantages to users. At a high level, these models work on the basis that every version of a document is a derivative of some other (or others), which can be expressed as a semantic relationship recording its evolution over time. This is capable of capturing more complicated progressions such as pages being split apart, which is of particular concern in a system which encourages such behaviour for the purposes of transclusion.

As a participant in Experiment II observed, support for branching and merging operations would be appealing to users making non-trivial changes to pages, where they may wish to keep a "local branch" of a node they are working on so that they may save incomplete or inconsistent states as checkpoints without disrupting the public view. More generally, support for explicitly-managed changesets would allow users to prepare more complex multi-page edits and ensure that they are all published as an atomic operation. A user may even wish to have their changeset applied to their view of the system: for it to contain a new link node, and see the effects of that link when browsing the wiki, effectively a simplistic adaptive hypermedia "lens". This functionality is quite comparable to the technical requirement to correctly implement preview, which requires that potentially multiple node changes (especially to links) are overlaid upon the node storage when embedding and rendering the page.

12.3.4 Distributed model

The fundamental model of the World Wide Web is distributed, in that there is no central index or repository that one must register with to be "on the web" above the Internet level. Web applications generally do not share this decentralisation, by nature of their

[1]`http://subversion.apache.org/`
[2]`http://git-scm.com/`

implementation as a single program install on a single domain: you cannot add pages to Wikipedia by hosting them on your own server. Some wiki systems have recognized this problem with support ‘interwiki’ links, which are effectively namespace macros to point to URIs on other wikis, but this is no tighter a coupling between the documents than a web-style link.

We feel that it may be worth reinvestigating research into distributed hypertext systems and considering their application to bidirectional cross-node relationships in wiki systems. The HyperG project [25] developed a flooding algorithm for a similar purpose [81]. The “Web 2.0” community have developed a potential solution to distributed web authentication in the form of the OpenID protocol [82], which has seen adoption by major websites such as LiveJournal. Weerkat already inherits from its previous design iteration a strong separation between the data interface—the API of wiki operations upon nodes—and the user interface, analogous to the “data bus” of the pre-web-applications World Wide Web [2]. Exposing this data interface may allow a preferred, or even thick-client application, user interface to transition smoothly to other wiki installs without needing to direct the user’s web browser to a different site.

Ideally, such a system should come close to realising the dreams of systems such as Xanadu [20], in that anybody who can publish on the web (or, more accurately, host a Weerkat install on the web), can become part of the global hyperdocument, freely transcluding from and creating semantic relationships between arbitrary pages on different servers, complete with bidirectional navigation. In this case, a user who creates a Weerkat install containing links between nodes in other Weerkat installs has effectively created an open hypermedia linkbase.

Appendix A

Experiment II resources

A.1 Script: Off-line review

This sub-session should be kept to under twenty minutes, to avoid participant fatigue, and thus discomfort and poor recall.

- Hello. Thanks for agreeing to participate in this study.
  - Would you like to look over the information letter again?
  - Feel free to keep that copy.
- What is the account name you have been using for this study over the past week? *(Go to contributions thereof.)*
- Are there any particular edits that you feel deserve particular attention? *(This is picking out what the 'expert' considers important.)*
- *For this, and any other edits until out of time (prefer major edits, and seek participant guidance to those they took longest on):*
  - What did you change? *(Trigger participant to bring edit back into memory: call up differences from system.)*
  - Why did you change it?
    - What was wrong with it before?
    - Why this page in particular?
    - Why this change? *(e.g. this new text)*
  - *If not simple text changes...*
  - How did you go about changing it?
    - Did you visit other pages?

        * Did you use any tools? *(e.g. backlinks)*
    - Could you have done it another way?
        * Why did you choose this way?
        * What would have made <*other way*> a better choice?

- Thanks. Would you like to take a short break before the next part?

A.2 Script: On-line self-report

This sub-session should be kept to under twenty minutes, to avoid participant fatigue, and thus discomfort and poor recall. The tasks should be randomly ordered for each participant.

- Now I am going to ask you to speak aloud about your thought processes while performing some observed editing tasks.
- We are using a small copy of some of Wikipedia with some errors and omissions introduced.
- Your changes are not going to be applied to real Wikipedia, so do not worry about having to write well and cite references.
- *Tasks:*
    1. Here are two pages about villains in two bond films: 'List of James Bond henchmen in The Spy Who Loved Me' and the same with 'Moonraker'. Please show me how you'd go about adding that Jaws' teeth are metallic. *(Content re-use.)*
    2. Here is the page about 'London Underground'. Nobody has written that the Underground is the oldest metro in the world, starting in 1863. Show me how you'd add this information. *(Level of Detail.)*
        - This information is probably notable enough to go on the 'Transport in London' page's summary about the Underground, too.
    3. Here is the disambiguation node for 'Shelf'. Show me how you'd fix any links to it to point to the specific type of shelf the article means. *(Any-anchor link editing.)*
        - Remember that the 'what links here' tool shows which pages link to the current page.
    4. Create a page about 'Multiple units', which are self-powered train cars, including the short descriptions of 'Multiple Units of Ireland' and 'Elektrichka'. *(Aggregation.)*

    - What if someone then needs to correct something in this summary?
    - What if someone adds another page about country-specific multiple units?
5. Here is the introductory paragraph of the 'Cake' article, with links removed. Add links to the text where you feel is appropriate. *(Generic linking.)*
6. This is the node for 'Belgium'. Please add its date of EU membership, the 25th of March 1957, to the infobox. *(Property editing.)*

- Thank you. The transcript of this experiment will be anonymised.
    - Do you have any final comments about your experience of performing the tasks?
    - Would you like to see results when available?
    - Would you prefer a cinema, or an Amazon UK, voucher?

A.3 Questionnaire

Approximate wiki edits per week (circle):	Less than 10 - 10–30 - 30–50 - More than 50
Of which are major (circle):	Less than 10 - 10–30 - 30–50 - More than 50
Experience with MediaWiki (tick):	○ Basic wikitext (e.g. titles) ○ Advanced wikitext (e.g. templates) ○ Page tools (e.g. 'what links here') ○ Administration (e.g. page moves; deletes)

Investigator use only

Participant ID: ____________________

Appendix B

Experiment III resources

B.1 Script: On-line self-report

This session should be kept to under an hour, to avoid participant fatigue, and thus discomfort and poor feedback. The tasks should be randomly ordered for each participant.

- Hello. Thanks for agreeing to participate in this study.
  - Would you like to look over the information letter again?
  - Feel free to keep that copy.
- I am now going to show you a wiki system with some unusual features.
  - *(Demonstrate and explain first-class nature of a link.)*
  - *(Demonstrate and explain transclusion.)*
  - *(Demonstrate and explain generic linking.)*
- Now I am going to ask you to speak aloud about your thought processes while performing some observed editing tasks.
- We are using a Wikipedia-like set of pages, with some errors and omissions introduced.
- Your changes are not going to be permanent, so do not worry about having to write well and cite references.
- *Tasks:*
  1. Here are two pages about villains in two bond films: ‘List of James Bond henchmen in The Spy Who Loved Me’ and the same with ‘Moonraker’. Please show me how you’d go about adding that Jaws’ teeth are metallic. *(Content re-use.)*

2. Here is the page about 'London Underground'. Nobody has written that the Underground is the oldest metro in the world, starting in 1863. Show me how you'd add this information. *(Level of Detail.)*
    - This information is probably notable enough to go on the 'Transport in London' page's summary about the Underground, too.
3. Here is the disambiguation node for 'Shelf'. Show me how you'd fix any links to it to point to the specific type of shelf the article means. *(Any-anchor link editing.)*
    - Remember that the 'Relations' table at the bottom shows which other pages link to the current page.
4. Create a page about 'Multiple units', which are self-powered train cars, including the short descriptions of 'Multiple Units of Ireland' and 'Elektrichka'. *(Aggregation.)*
    - What if someone then needs to correct something in this summary?
    - What if someone adds another page about country-specific multiple units?
5. Here is the introductory paragraph of the 'Cake' article, with links removed. Add links to the text where you feel is appropriate. *(Generic linking.)*
6. This is the node for 'Belgium'. Please add its date of EU membership, the 25th of March 1957, to the attributes box. *(Property editing.)*

- Thank you. The transcript of this experiment will be anonymised.
    - Do you have any final comments about your experience of performing the tasks?
    - Or of the system in general?
    - Would you like to see results when available?

B.2 Questionnaire

Did you participate in the previous MediaWiki experiment?	○ Yes
	○ No
Experience with wikis (tick):	○ Basic wikitext (e.g. titles)
	○ Advanced wikitext (e.g. templates)
	○ Page tools (e.g. 'what links here')
	○ Administration (e.g. page moves; deletes)

Participant ID: ____________________ **(Investigator use only)**

Appendix C

Model ontology in Turtle

This is the ontology from section 7.8, presented in Turtle. It assumes that wk is a namespace for the Weerkat model, and rdf and rdfs are RDF and RDFS respectively.

We do not constrain the values of wk:Detail, as they are somewhat specific to the implementation of the model, and the domain to which it will be applied. A likely possibility for a textual system is an enumerated class, with instances such as 'sentence', 'paragraph', 'section', and 'chapter'.

```
wk:Node a rdfs:Class .

wk:alternateNode a rdf:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  wk:Node .

wk:detailLevel a rdf:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  wk:Detail .

wk:Detail a rdfs:Class .

wk:versionOf a rdf:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  wk:Node .

wk:obsoletedBy a rdf:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  wk:Node .
```

```
wk:representation a rdf:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  xsd:anyURI .

wk:parametricInstanceOf a rdf:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  wk:Node .

wk:Link a rdfs:Class ;
  rdfs:subClassOf wk:Node .

wk:source a rdfs:Property ;
  rdfs:domain wk:Link ;
  rdfs:range  wk:Node .

wk:target a rdfs:Property ;
  rdfs:domain wk:Link ;
  rdfs:range  wk:Node .

wk:sourceQuery a rdfs:Property ;
  rdfs:domain wk:Link ;
  rdfs:range  rdfs:Literal .

wk:targetQuery a rdfs:Property ;
  rdfs:domain wk:Link ;
  rdfs:range  rdfs:Literal .

wk:linkType a rdfs:Property ;
  rdfs:domain wk:Node ;
  rdfs:range  wk:Node .

wk:Alternation a rdfs:Class ;
  rdfs:subClassOf wk:Link .

wk:Transclusion a rdfs:Class ;
  rdfs:subClassOf wk:Link .
```

Appendix D

SEML parser state diagram

Figure D.1 shows the state transitions for the core of the SEML parser, starting at IN_CONTENT, which matches to SEML::Reader::read_raw() in the source. In practice, this is wrapped by a read() method which adds support for auto-paragraphing, system elements (such as future !verbatim support), and full guard validation. The lexer also performs escaping during tokenisation, so the appearance of syntactic tokens such as brackets here indicates that they are unescaped: escaped ones are passed as character tokens (‘*char*’). Whitespace (‘*wsp*’) excludes newlines (any line-ending format), which are represented as ‘`\n`’.

Study of the source code is also advised to fully understand the parser, as some states rely upon minor side-effects for efficiency and conciseness, and this diagram does not show event emission. In particular, transitions marked with the symbol ‘**R**’, here meaning ‘reparse’, cause the token to be re-evaluated in the context of the new state via a tail-recursive call (we have implied some here where the implementation actually accumulates internal state instead). Termination is controlled by an end-of-stream token from the lexer.

Not shown is the ELEM_ERR state, which absorbs all undefined transitions, and attempts recovery by discarding tokens until it encounters an element end (‘]’), at which point it transitions back to IN_CONTENT. The similar state, ATTR_ERR, is shown due to its reduced scope. It discards everything up to the next attribute or content gap, and tries to continue from there. (Strictly speaking, tokens representing lexing errors, such as unknown escapes, are discarded, generate error events, then parsing resumes uninterrupted. This prevents them from truncating following content.)

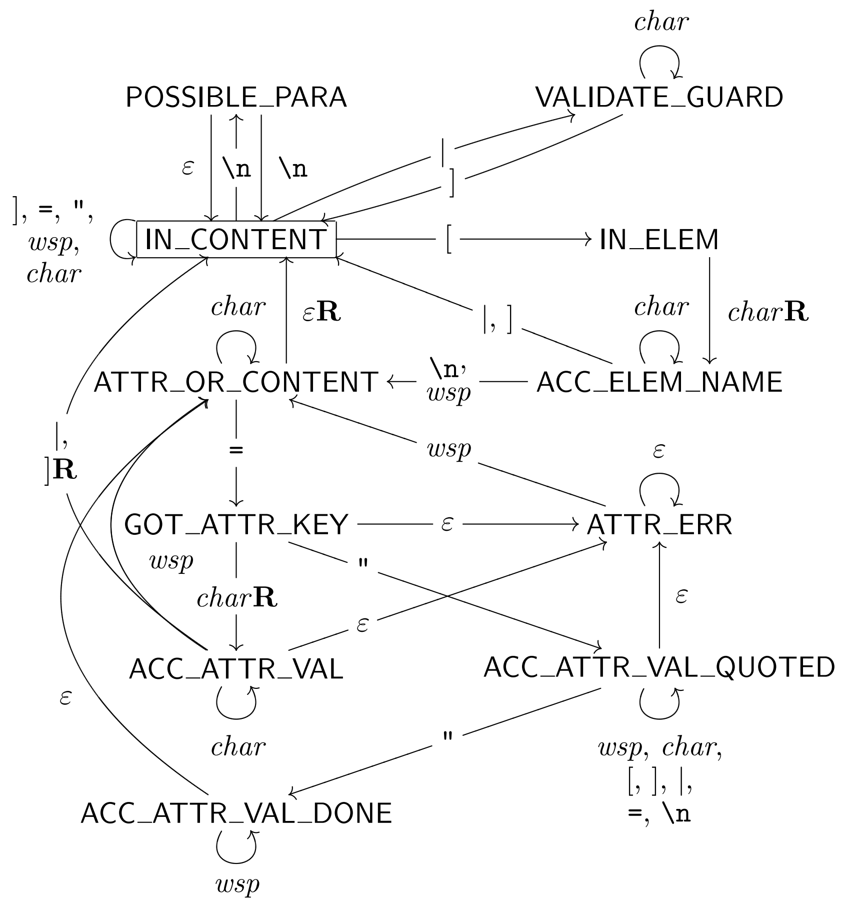


FIGURE D.1: SEML parser state diagram

Appendix E

Open Weerkat entity types

E.1 Current

E.1.1 node

The root entity for a node. Has no attributes.

E.1.2 link

A link which has been embedded into a document at this endpoint. The children of this element are the content of the anchor it matched at this endpoint or, for transclusive links, the content of the target node.

Note that these elements are never stored: they exist only for editing and display. The core embed/separate routines convert between these and anchor elements.

id Node for the first-class representation of this link.

aid Number of the anchor this link attaches to in the document in which it has been embedded.

to A literal destination endpoint of the link; maps to target node attribute.

from A literal source endpoint; maps to source.

toq A query destination endpoint; maps to targetQuery.

fromq A query source endpoint; maps to sourceQuery.

type Type(s) of the link node.

E.1.3 trans

As link, except that the XHTML render does not convert them into hyperlinks. Theoretically these also allow the separation algorithm to detect changes between transclusive and regular link types which can be a subtle but dangerous editing error.

E.1.4 anchor

The storage representation of a anchor point in a node, used for native transclusion. Does not have any content, as the embed/separate routines should replace these with link or trans elements for editing or display.

id Leaf-name of the node; interpreted under a namespace of *parent note*.anchor.

E.1.5 heading

A HTML-style standalone heading.

level Depth of the heading in the document structure; comparable to HTML's 'H1', 'H2' etc. numbering.

E.1.6 em, strong, para, quote, q

These elements function as defined in XHTML; 'para' is the name for HTML's 'p' element, and 'quote' for 'blockquote'.

E.1.7 err

An error in the the document markup. The content is the error message.

E.2 Future

E.2.1 section

A logical subdivision of the document, which should replace heading. This content of the element would be the entire section, and a heading of appropriate depth would be automatically generated for it from the section's title.

title Title of the (sub)section.

Appendix F

Open Weerkat base nodes

This appendix lists the default set of 'core' nodes used with the Open Weerkat prototype. They are shown as rendered, for readability; underlining is used to show where links have been embedded.

F.1 Node

This is the type of Nodes in the system. When using nodes as link targets, just give their title.

F.2 Link

This is the type of ordinary links.

Links can specify endpoints in two ways: literal endpoints with source and target attributes (which are 'from' and 'to' in the markup); and query endpoints with sourceQuery and targetQuery attributes ('fromq' and 'toq' in the markup).

Links of the subtype Transclusion will transclude their targets over the top of their source anchors. This even works during editing, and you can use it to seamlessly change a node *and* any other nodes it transcludes in one go. Query-based translcusive links are currently not supported as they introduce a lot of very nasty problems and capabilities.

Links that appear in nodes while you are editing will gain "aid" and "id" attributes. The "aid" is the identifier of the anchor, *namespaced by the node it belongs to.* For example, an aid of 1 here refers to Link.anchor.1. If the aid is "_", the anchor is a special virtual anchor for a query endpoint, and you can't actually change it—it's just there so you can see which link is affecting that part of the document. id is the *full* name of the Link which that anchor is acting as an endpoint for. By and large, you should just leave these attributes alone.

Currently, “fat” links—those with multiple targets—are not well supported due to the limitations of HTML. You will likely find that only one target works when clicked.

F.3 Transclusion

This is the magic type for transclusive links.

subClassOf Link

F.4 Query

This is the type of link queries.

Currently, the only query construct you can use is CONTAINS(). It will match against a single, quoted word: sourceQuery=CONTAINS(‘ontology’) will act as an anchor for occurrences of the word “ontology”.

F.5 source, target

This is the type for a literal link (“from”/“to”) endpoint.

domain Link

range Node

F.6 sourceQuery, targetQuery

This is the type for a query link (“from”/“to”) endpoint.

domain Link

range Query

Bibliography

[1] M. Krötzsch, D. Vrandečić, and M. Völkel, "Wikipedia and the semantic web - the missing links," in *Proceedings of the WikiMania2005*, 2005. Online at `http://www.aifb.uni-karlsruhe.de/WBS/mak/pub/wikimania.pdf`.

[2] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann, "World-Wide Web: The Information Universe," *Electronic Networking: Research, Applications and Policy*, vol. 1, no. 2, pp. 74–82, 1992.

[3] F. Manola and E. Miller, "RDF Primer," tech. rep., W3C, Feb 2004.

[4] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 285, pp. 28–37, May 2001.

[5] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, no. 2, pp. 199–220, 1993.

[6] D. Brickley and R. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema," tech. rep., W3C, Feb 2004.

[7] D. L. McGuinness and F. van Harmelen, "OWL Web Ontology Language Overview," tech. rep., W3C, Feb 2004.

[8] S. Harris and N. Gibbins, "3store: Efficient Bulk RDF Storage," in *1st International Workshop on Practical and Scalable Semantic Systems (PSSS'03)*, pp. 1–15, 2003.

[9] S. Harris, N. Lamb, and N. Shadbolt, "4store: The Design and Implementation of a Clustered RDF Store," in *5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pp. 94–109, Oct 2009.

[10] E. Prud'hommeaux and A. Seaborne, "SPARQL Query Language for RDF," tech. rep., W3C, Jan 2008.

[11] T. Berners-Lee, "Linked Data—Design Issues," Jul 2006. Online at `http://www.w3.org/DesignIssues/LinkedData.html`.

[12] L. Sauermann, R. Cyganiak, and M. Völkel, "Cool URIs for the Semantic Web," Tech. Rep. TM-07-01, DFKI, Feb 2007.

[13] D. Connolly and I. Hickson, “An Entity Header for Linked Resources.” Expired IETF document draft-connolly-link-header-01. Online at `http://www.w3.org/Protocols/9707-link-header.html`., Apr 1999.

[14] M. Nottingham, “HTTP Header Linking.” Expired IETF document draft-nottingham-http-link-header-00. Online at `http://www.mnot.net/drafts/draft-nottingham-http-link-header-00.txt`., Jun 2006.

[15] P. Stickler, “The URI Query Agent Model.” Online at `http://sw.nokia.com/uriqa/URIQA.html`.

[16] B. Adida and M. Birbeck, “RDFa Primer,” tech. rep., W3C, Oct 2008.

[17] W. B. Rayward, “Visions of Xanadu: Paul Otlet (1868–1944) and Hypertext,” *Journal of the American Society for Information Science*, vol. 45, pp. 235–250, May 1994.

[18] P. Otlet, “The science of bibliography and documentation,” 1903. In *The international organization and dissemination of knowledge: Selected essays of Paul Otlet*, (W. B. Raynard, trans. and ed.), Elsevier, 1990.

[19] V. Bush, “As We May Think,” *The Atlantic Monthly*, vol. 176, pp. 101–108, Jul 1945.

[20] T. Nelson, *Literary Machines.* Sausalito, California: Mindful Press, 93.1 ed., 1993.

[21] S. R. Newcomb, N. A. Kipp, and V. T. Newcomb, “The “HyTime”: hypermedia/time-based document structuring language,” *Commun. ACM*, vol. 34, no. 11, pp. 67–83, 1991.

[22] H. Davis, W. Hall, I. Heath, G. Hill, and R. Wilkins, “Towards an integrated information environment with open hypermedia systems,” in *ECHT ’92: Proceedings of the ACM conference on Hypertext*, (New York, NY, USA), pp. 181–190, ACM Press, 1992.

[23] A. M. Fountain, W. Hall, I. Heath, and H. Davis, “MICROCOSM: An open model for hypermedia with dynamic linking,” in *European Conference on Hypertext*, pp. 298–311, 1990.

[24] T. Berners-Lee, Y. Chen, L. Chilton, D. Connolly, R. Dhanaraj, J. Hollenbach, A. Lerer, and D. Sheets, “Tabulator: Exploring and Analyzing linked data on the Semantic Web,” in *Proceedings of the 3rd Int. Semantic Web User Interaction*, 2006.

[25] K. Andrews, F. Kappe, and H. Maurer, “Serving information to the Web with Hyper-G,” in *Proceedings of the Third International World-Wide Web conference on Technology, tools and applications*, (New York, NY, USA), pp. 919–926, Elsevier North-Holland, Inc., 1995.

[26] D. T. Michaelides, D. E. Millard, M. J. Weal, and D. C. De Roure, “Auld Leaky: A Contextual Open Hypermedia Link Server,” in *The 7th Workshop on Open Hypermedia Systems*, (Aarhus, Denmark), 2001.

[27] C. Bailey, W. Hall, D.Millard, and M. Weal, “Towards Open Adaptive Hypermedia,” in *2nd International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, 2002.

[28] D. Millard, H. Davis, M. Weal, K. Aben, and P. D. Bra, “AHA! meets Auld Linky: integrating designed and free-form hypertext systems,” in *HYPERTEXT ’03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, (New York, NY, USA), pp. 161–169, ACM Press, 2003.

[29] R. Furuta and P. D. Stotts, *HYPERTEXT: state of the art*, ch. 22, pp. 205–213. Intellect Limited, 1990.

[30] T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret, “The World-Wide Web,” *Commun. ACM*, vol. 37, no. 8, pp. 76–82, 1994.

[31] T. Berners-Lee, “Cool URIs don’t change.” Online at `http://www.w3.org/Provider/Style/URI`, 1998.

[32] M. Bieber, F. Vitali, H. Ashman, V. Balasubramanian, and H. Oinas-Kukkonen, “Fourth generation hypermedia: some missing links for the World Wide Web,” *Int. J. Hum.-Comput. Stud.*, vol. 47, no. 1, pp. 31–65, 1997.

[33] F. Halasz and M. Schwartz, “The Dexter hypertext reference model,” *Communications of the ACM*, vol. 37, no. 2, pp. 30–39, 1994.

[34] K. Grønbæk and R. H. Trigg, “Toward a Dexter-based model for open hypermedia: unifying embedded references and link objects,” in *HYPERTEXT ’96: Proceedings of the the seventh ACM conference on Hypertext*, (New York, NY, USA), pp. 149–160, ACM Press, 1996.

[35] M. d’Inverno, M. Priestley, and M. Luck, “A Formal Framework for Hypertext Systems,” *IEE Proceedings - Software Engineering*, vol. 144, no. 3, pp. 175–184, 1997.

[36] H. C. Davis, D. E. Millard, S. Reich, N. Bouvin, K. Grønbæk, P. J. Nürnberg, L. Sloth, U. K. Wiil, and K. Anderson, “Interoperability between hypermedia systems: the standardisation work of the OHSWG,” in *Proceedings of the tenth ACM Conference on Hypertext and hypermedia : returning to our diverse roots: returning to our diverse roots*, HYPERTEXT ’99, (New York, NY, USA), pp. 201–202, ACM, 1999.

[37] D. E. Millard, L. Moreau, H. C. Davis, and S. Reich, “FOHM: a fundamental open hypertext model for investigating interoperability between hypertext domains,” in

HYPERTEXT '00: Proceedings of the eleventh ACM on Hypertext and hypermedia, (New York, NY, USA), pp. 93–102, ACM Press, 2000.

[38] L. Moreau and W. Hall, "On the Expressiveness of Links in Hypertext Systems," *The Computer Journal*, vol. 41, no. 7, pp. 459–473, 1998.

[39] S. DeRose, E. Maler, and D. Orchard, "XML Linking Language (XLink) Version 1.0," tech. rep., W3C, Jun 2001.

[40] B. Halsey and K. M. Anderson, "XLink and Open Hypermedia Systems: A Preliminary Investigation," in *HYPERTEXT '00: Proceedings of the the Eleventh ACM conference on Hypertext and Hypermedia*, (New York, NY, USA), pp. 212–213, ACM Press, 2000.

[41] L. Carr, W. Hall, S. Bechhofer, and C. Goble, "Conceptual Linking: Ontology-based Open Hypermedia," in *WWW '01: Proceedings of the 10th international conference on World Wide Web*, (New York, NY, USA), pp. 334–342, ACM, 2001.

[42] M. Dzbor, J. Domingue, and E. Motta, "Magpie—Towards a Semantic Web Browser," in *The SemanticWeb - ISWC 2003*, vol. 2870 of *Lecture Notes in Computer Science*, pp. 690–705, Springer Berlin / Heidelberg, 2003.

[43] S. Buckingham Shum, E. Motta, and J. Domingue, "ScholOnto: An Ontology-Based Digital Library Server for Research Documents and Discourse," *International Journal on Digital Libraries*, vol. 3, pp. 237–248, 2000.

[44] P. Boulain, M. Parker, D. Millard, and G. Wills, "Weerkat: An extensible semantic wiki," in *Proceedings of 8th Annual Conference on WWW Applications*, (Bloemfontein, Free State Province, South Africa), 2006.

[45] A. Désilets, S. Paquet, and N. Vinson, "Are Wikis Usable?," in *The 2005 International Symposium on Wikis*, (San Diego, California, USA), Oct 2005.

[46] M. Gaved, T. Heath, and M. Eisenstadt, "Wikis of Locality: Insights from the Open Guides," in *WikiSym '06*, 2006.

[47] F. Ortega and J. M. Gonzalez-Barahona, "Quantitative Analysis of the Wikipedia Community of Users," in *WikiSym '07*, 2007.

[48] D. Wilkinson and B. Huberman, "Cooperation and Quality in Wikipedia," in *WikiSym '07*, 2007.

[49] B. T. Adler, L. de Alfaro, I. Pye, and V. Raman, "Measuring Author Contributions to the Wikipedia," in *WikiSym '08*, 2008.

[50] U. Brandes, P. Kenis, J. Lerner, and D. van Raaij, "Network Analysis of Collaboration Structure in Wikipedia," in *18th International World Wide Web Conference (WWW2009)*, April 2009.

[51] B. Suh, G. Convertino, E. H. Chi, and P. Pirolli, “The Singularity is Not Near: Slowing Growth of Wikipedia,” in *WikiSym ’09*, 2009.

[52] P. K.-F. Fong and R. P. Biuk-Aghai, “What Did They Do? Deriving High-Level Edit Histories in Wikis,” in *WikiSym ’10*, 2010.

[53] R. Tolksdorf and E. P. B. Simperl, “Towards Wikis as Semantic Hypermedia,” in *WikiSym ’06*, 2006.

[54] S. Schaffert, “IkeWiki: A Semantic Wiki for Collaborative Knowledge Management,” in *15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, (Manchester, UK), pp. 388–396, Jun 2006.

[55] M. Kiesel, “Kaukolu: Hub of the Semantic Corporate Intranet,” in *First Workshop on Semantic Wikis: From Wiki to Semantic*, (Budva, Montenrego), pp. 31–42, Jun 2006.

[56] J. Fischer, Z. Gantner, S. Rendle, M. Stritt, and L. S. Thieme, “Ideas and Improvements for Semantic Wikis,” *Lecture Notes in Computer Science*, vol. 4011/2006, pp. 650–663, 2006.

[57] M. Hepp, D. Bachlechner, and K. Siorpaes, “OntoWiki: Community-driven Ontology Engineering and Ontology Usage based on Wikis,” in *Proceedings of the 2005 International Symposium on Wikis*, Oct 2005.

[58] F. G. Halasz, “Reflections on NoteCards: seven issues for the next generation of hypermedia systems,” *Commun. ACM*, vol. 31, no. 7, pp. 836–852, 1988.

[59] F. Halasz, ““Seven Issues” Revisited.” Keynote Address, Hypertext ’91 Conference, Dec 1991. Online at `http://www2.parc.com/spl/projects/halasz-keynote/`.

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

[61] P. J. Brown and H. Brown, “Embedded or separate hypertext mark-up: is it an issue?,” *Electronic Publishing*, vol. 8, pp. 1–13, 1995.

[62] H. Ashman and J. Verbyla, “Dynamic link management via the functional model of the link,” in *In Proceedings of Basque International Workshop on Information Technology*, (Biarritz, France), 1994.

[63] P. Mika, “Ontologies are us: A unified model of social networks and semantics,” in *4th International Semantic Web Conference*, 2005.

[64] X. Wu, L. Zhang, and Y. Yu, “Exploring social annotations for the semantic web,” in *WWW ’06: Proceedings of the 15th international conference on World Wide Web*, (New York, NY, USA), pp. 417–426, ACM Press, 2006.

[65] D. Beckett, "RDF/XML Syntax Specification (Revised)," tech. rep., W3C, Feb 2004.

[66] J. Seidenberg and A. Rector, "Representing Transitive Propagation in OWL," in *25th International Conference on Conceptual Modeling*, 2006.

[67] P. Boulain, N. Shadbolt, and N. Gibbins, "Hyperstructure maintenance costs in large-scale wikis," in *Social Web and Knowledge Management* (P. Dolog, M. Krötzsch, S. Schaffert, and D. Vrandečić, eds.), vol. 356, (Beijing, China), CEUR Workshop Proceedings, April 2008. ISSN 1613-0073. Online at `http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-356/paper4.pdf`.

[68] A. Swartz, "Who Writes Wikipedia?." Online at `http://www.aaronsw.com/weblog/whowriteswikipedia`, Sep 2006.

[69] V. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics Doklady*, vol. 10, p. 707, Feb 1966.

[70] P. Boulain, N. Shadbolt, and N. Gibbins, *Weaving Services and People on the World Wide Web*, ch. 16: Studies on Editing Patterns in Large-scale Wikis, pp. 325–349. Springer, 2009.

[71] L. Bainbridge, "Verbal protocol analysis," in *Evaluation of human work* (J. R. Wilson and E. N. Corlett, eds.), ch. 7, Taylor & Francis Ltd, 1990.

[72] N. Shadbolt and M. Burton, "Knowledge elicitation," in *Evaluation of human work* (J. R. Wilson and E. N. Corlett, eds.), ch. 13, Taylor & Francis Ltd, 1990.

[73] N. S. Borenstein, *Programming as if People Mattered*, ch. 19. Princeton University Press, 1991.

[74] F. Vitali, "Versioning hypermedia," *ACM Comput. Surv.*, vol. 31, no. 4es, p. 24, 1999.

[75] K. Østerbye, "Structural and cognitive problems in providing version control for hypertext," in *ECHT '92: Proceedings of the ACM conference on Hypertext*, (New York, NY, USA), pp. 33–42, ACM, 1992.

[76] A. Dattolo and V. Loia, "Collaborative version control in an agent-based hypertext environment," *Information Systems*, vol. 21, no. 2, pp. 127–145, 1996.

[77] R. Daniel, Jr., "Harvesting RDF Statements from XLinks." W3C Note, Sep 2000. Online at `http://www.w3.org/TR/xlink2rdf/`.

[78] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," Jan 2005. IETF RFC 3986.

[79] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax," Aug 1998. IETF RFC 2396.

[80] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, and Z. Ives, “DBpedia: A Nucleus for a Web of Open Data,” in *In 6th Int’l Semantic Web Conference, Busan, Korea*, pp. 11–15, Springer, 2007.

[81] F. Kappe, “A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems,” *Journal of Universal Computer Science*, vol. 1, no. 2, pp. 84–104, 1995.

[82] D. Recordon and B. Fitzpatrick, “OpenID Authentication 1.1,” tech. rep., OpenID Foundation, May 2006. Online at `http://openid.net/specs/openid-authentication-1_1.html`.