Weerkat: An extensible semantic Wiki

PR Boulain
School of Electronics and Computer Science
University of Southampton
Southampton
United Kingdom
prb102@ecs.soton.ac.uk

MB Parker
Faculty of Informatics and Design
Cape Peninsula University of Technology
Cape Town
South Africa
parkerm@cput.ac.za

DE Millard
School of Electronics and Computer Science
University of Southampton
Southampton
United Kingdom
dem@ecs.soton.ac.uk

G Wills
School of Electronics and Computer Science
University of Southampton
Southampton
United Kingdom
gbw@ecs.soton.ac.uk

Abstract

Wikis are Web applications that blur the boundaries between readers and authors, allowing non-technical people to author hypertexts through a web interface. A Semantic Wiki is a Wiki that attempts the same thing with the Semantic Web, allowing non-technical users to create semantic resources and/or ontologies. In this paper we characterise the different ways in which a Wiki might support the Semantic Web and present Weerkat, a modular and extensible Wiki that has ontological hypertext support. Key to this has been the design and development of a highly flexible Wiki architecture which allows easy modification and augmentation of functionality.

Keywords: Semantic Web, Wiki, Weerkat.

1. Introduction

The World Wide Web is the most popular hypertext system, yet it suffers a number of problems when evaluated alongside other hypertext systems. In particular it has a very clear separation of author and reader; this has a number of consequences:

- Users of the Web cannot annotate or change the pages that they are viewing

- Creating a Web page requires specialist skills
- Collaborative authoring of a Web site is difficult

Because the Web is a framework for distribution it is possible to create Web applications that have these features. One general solution is a WikiWikiWeb (Wiki for short) a type of Web server (or application running on a traditional Web server) that allows any reader of its pages to log in and alter those pages, or create new ones (Leuf, & Cunningham 2001). The interface is part of the Web site and therefore allows non-specialist users to contribute to the hypertext.

Another problem with the Web is that it does not normally manage its links separately from its content. When this is done, such as with Open Hypermedia systems (Davis, Hall, Heath, Hill, & Wilkins, 1992) (Grønbæk, & Trigg, 1994) it enables broken links to be detected, links to be personalised to a user, and links to be dynamically generated.

The Semantic Web effort is an attempt to add machine readable semantics to the Web (Berners-Lee, Hender, & Lassila, 2001). These semantics take the form of meta-data tightly controlled by an ontology (schemas that define the classes and relationships in the domain between in some agreed manner) (Gruber, 1993:199-220). These annotations can be used for exchanging data between distributed processes, but it can also be used to enhance the hypertext functionality of the Web of today, by augmenting current Web content with semantics that allow it to be reasoned about.

This could improve the search power of the Web. Current searches are based purely of text matches - perhaps weighted according to their importance within the hypertext network (Page, Brin, Motwani & Winograd 1999). Contextual search systems attempt to improve this by looking at the context to decide the meaning (Carr, Hall, Bechhofer, & Goble, 2001) (El-Beltagy, Hall, De Roure, & Carr, 2001), for example, does the word 'pie' appear in the context of food, or in the context of a chart. Adding semantics to pages means that it is possible to make these distinctions explicit in the content itself and search systems can ignore homonyms but find synonyms.

It is also possible to create new hypertext structures based on the semantic information. This is sometimes called Ontological Hypertext and usually involves the process of converting a chain of semantic relations (some of which may be the product of reasoning systems) into a hypertext link or set of links (Weal, Hughes, Millard, & Moreau, 2001).

Currently the ideas of the Semantic Web idea are not known to regular users, and the size and complexity of the task of semantically enhancing content provides a barrier to authorship. The additional complexities of navigating and viewing semantically enriched hypertexts is a continuing research field (schraefel, Shadbolt, Gibbins, Harris, & Glaser, 2004), as are the problems of developing and agreeing new ontologies.

In this paper we examine the concept of a Semantic Wiki, an attempt to use the Wiki concept to make semantics accessible to ordinary users in the same way as ordinary Wiki's make hypertext itself accessible. We then present Weerkat, a modular and extensible Wiki system upon which we have built an initial implementation of a Semantic Wiki that combines the two technologies to provide an easily accessible application of ontological hypertext. Because of its modular and extensible nature it is our hope that in the future Weerkat can act as a platform for exploring other types of semantic web support.

The rest of the paper is organised as follows: Section 2 looks at the current Wiki systems and the ways in which semantics have already been combined with the Wiki concept. Section 3 looks at the requirements for a Semantic Wiki that could support semantics through an extensible architecture, and presents a design that fulfils the requirements. Section 4 presents Weerkat, our implementation of the extensible Semantic Wiki design that currently implements semantically-driven hypertext. Section 5 describes our future plans to extend Weerkat so that it offers further support of the Semantic Web and concludes the paper.

2. Related Work

In this section we look at some of the flavours of Wiki that are currently being used on the Web, and how Semantic Web technology has influenced them, in order to draw a comparison. We also look at how the semantic web has been used in hypertext systems.

2.1 MediaWiki

UseMod and MediaWik are pure WikiWikiWeb systems. UseMod is similar to the original WikiWikiWeb design: it has a simple, straightforward, and lightweight implementation in a single CGI script. MediaWiki is a LAMP (Linux, Apache, MySQL, PHP) application: thus significantly ‘heavier’, but with more features.

MediaWiki provides a degree of namespacing, templating and the ability to declare a node as a member of a number of categories, which generate automatic lists of members. Pages can be edited in individual sections, yet these sections are defined by subheadings in an overall page source, and you cannot, for example, transclude a section of one node into another: transclusion is restricted to use of templates, e.g. providing a “this information on this page is disputed” notice. MediaWiki also has an extension architecture, where new source files can be loaded in via the settings file and add to or modify Wiki behaviour through callbacks.

2.2 Platypus

Platypus Wiki is an open-source project which describes itself as “a project to develop an enhanced Wiki Wiki Web with ideas borrowed from the Semantic Web”. Platypus is still in its early stages and does not yet have the full intended functionality but it has similar goals to Weerkat. Documentation is limited, making detailed comparison difficult, although there are conference proceedings available (Campanini, Castagna, & Tazzoli,. 2004) and a demonstration system is occasionally running.

In Platypus each node becomes a URI which can be the subject or object of a RDF statement stored in the Wiki. This allows the Wiki to function as a user-editable semantic graph, with inter-node links themselves being specified as RDF statements relating the two nodes.

Through careful construction of the URL for accessing nodes, it is possible to retrieve the HTML content and RDF metadata about a node separately, and separate from the web interface—this facilitates transclusion into other systems, although not of content between nodes. However, Platypus exposes its users to the raw semantic information, presenting them with large quantities of RDF, RDFS, and technical terminology. For example, sidebars titled “rdfs:isDefinedBy” and “rdfs:seeAlso”, instead of “Defined by” and “See also". Metadata and page content are edited separately, and the editing mechanism is a text area of either RDF/XML or Notation-3.

Platypus also has elements of dynamic linking through a feature called "site links". This is a list of terms which are detected in node bodies and get linked to a given URL. However, these are not links to ontological URIs, but are instead simple HTML hyperlinks to web pages. Platypus also detects terms which it has metadata for, such as "projects:DOAP", and automatically links them to the appropriate node. 'Normal' unidirectional, anchored links are also supported.

2.3 Comparison of Systems

Table 1 summarises the differences in feature sets between related systems. Fully implemented is showed by a filled in diamond (♦), an outlined diamond (◊) indicates that a feature does not (yet) exist. Our intention is that Weerkat will support all of these features, providing much of the semantic functionality of existing semantic Wikis such as Platypus, but in such a way that the semantics becomes accessible to ordinary users in the spirit of the original WikiWikiWeb.

Table 1: Comparison of systems

Feature	UseMod	Mediawiki	Platypus
Basic wiki features	♦	♦	♦
Extensible design	◊	♦	◊ [1]
High-level documentation	◊	♦	◊
Intuitive interface	♦ [2]	♦	◊
Non-technical configuration	◊ [3]	◊[4]	
Annotates pages with semantics	◊	◊	♦
User editing of semantics	◊	◊	♦
Interface abstraction	◊	◊	◊

[1] A plugin-based system is planned.

[2] Usability testing showed several significant problems with areas of UseMod's interface, yet most users could complete most of the given tasks.

[3] Requires modification of the source of the wiki itself.

[4] Requires modification of a 'settings' source file.

2.4 Semantic Web and hypertext

The parallels between the semantic web as a network of semantically meaningful resources, and hypertext as a network of human meaningful resources have been described before (Millard, Gibbins, Michaelides, & Weal, 2005). Early hypertext systems saw their hyperstructures as meaningful semantic graphs and used that understanding to display links according to their types.

Generally hypertexts links are higher level than the statements on the Semantic Web. Other work has focused on deriving new links from semantically enriched resources, in these cases chains of semantic web statements are converted into links between dynamic pages which are themselves generated by collating information modeled in an ontology (Collier, 1987).

The Ontoportal project applies an exact mapping and uses an ontology to control the types of hypertext links that are possible between pages of content (Miles-Board, Kampa, Carr, & Hall, 2001) This is so that the information in the domain (for Ontoportal this is UK Research) is structured in a well-defined way.

COHSE (Conceptual Open Hypermedia ServicE) is a system that uses ontological reasoning to derive new Web links (Carr, Hall, Bechhofer, & Goble, 2001). The system analyses pages within the hypertext (as they are requested at runtime) and extracts the major keywords. These are then compared to a domain concept map (which is governed by an ontology that allows concepts to be related using terms such as broader or narrower). The concept which most closely matches the keyword is selected and any other pages associated with that concept are returned. In cases where the renderer believes it does not have enough link then the onological search is widened, and pages associated with concepts related to the main concept are also added to the page. It is these relationships between ontologies, semantic graphs and hypertexts that we wish to explore within Weerkat.

3. Design of a semantic Wiki

In this section we define the requirements for an extensible semantic Wiki and explore an appropriate design. For our first development phase we have concentrated on the first type of semantic support, ontological hypertext, and demonstrated the flexibility of the system by implementing this as an extension.

Existing Wiki systems tend to be designed in a monolithic fashion, with the Wiki functionality being provided in one process or module. This core behaviour can then sometimes be modified and augmented by additional ‘extensions’; otherwise, modification of the source itself is required to change behaviour.

Our efforts with Weerkat take a different approach; the core of the Wiki is simply a framework, breaking down the functionality into a set of individual components. Interactions between these components provide the basic Wiki functionality. This allows developers to make considerable changes to the function of the Wiki by writing new versions of these components.

3.1 Wiki requirements

There are a number of basic functions which are expected of a Wiki system. As with any content, it is desirable to know the author. On Wikis, this is of particular importance, as users may take roles as editors as well.

1. A Wiki should be able to uniquely identify users.

Wiki systems also require basic page editing facilities. Version control helps to limit the effects of vandalism and spamming; it must also deal with concurrent editing of the same node by multiple users. Some form of mark-up is required to add additional formatting to sections of text.

2. A Wiki should allow users to view, create and modify nodes.

3. A Wiki should perform simple version management, allowing comparison of different versions and the facility to revert to a previous version.

4. A Wiki should handle conflicts in page modification in a manner which helps prevent loss of changes.

5. A Wiki should have a simple mark-up language which does not require significant reading or training to use.

A semantic Wiki to support ontological hypertext has additional requirements.

6. The mark-up language should be conducive to implicitly deriving semantic information.

7. It should be possible to specify an ontology to be used across all nodes to add links anchored on terms within user-generated pages to contextual information on the given term.

8. It should support multiple ontologies per node.

9. It should support user-defined ontologies.

There are also general requirements for easy extensibility:

10. The architecture should be highly modular for maximum flexibility.

11. It should be possible to configure and administrate the Wiki without the need for programming skills.

12. The web interface must generate valid documents, as per W3C standards.

13. The web interface should support a variety of platforms.

14. The web interface should support users with disabilities.

3.2 High-level architecture

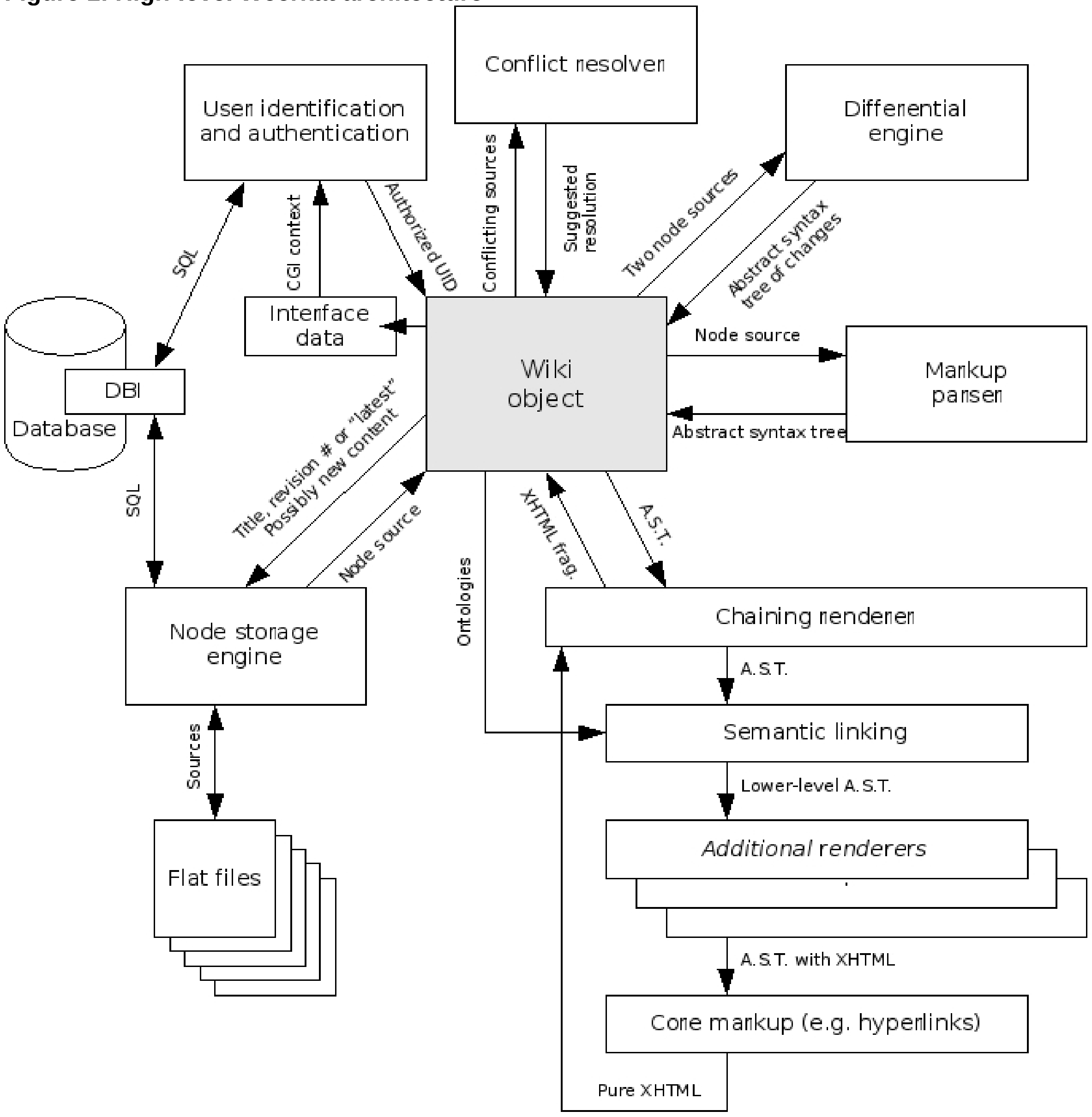
Based on the tasks the Wiki is to perform, and the requirements described above, we can identify several distinct Wiki functions: storage, user identification, conflict resolution, markup parsing, and rendering. Rendering itself can be broken down into the effects of semantics and markup which dictates style. By defining an interface for each of these components, it will be possible to choose between multiple different implementations. A block diagram of this design is shown in Figure 1.

Wiki: At the core is the Wiki Object, the central orchestrator which controls and invokes each of the other components. This provides a core of centralised behaviour, such as configuration reading and handling. Web interfaces, ‘thick-client’ GUI interfaces, or even some automated programmes, can use the Wiki Object like a library of common actions, with the ability to access the contained components for advanced use.

Node storage engine: The task of the node storage engine is to store the source—the raw, unprocessed content and markup—of nodes. The Wiki Object can identify a particular node and version by combination of a node title and revision number, and can ask the storage engine for a revision or provide it with a new one. It does not need to know how the storage is performed, so storage engines could be written which use a database backend via a technology like ODBC, JDBC, or Perl DBI, flat files, or even a triple store.

User identification and authentication: This is an important feature of Wikis, as knowing who made an edit to a node can be useful to identify and ban abusers. In some cases, it may be desirable to limit editing rights to certain individuals, so the node which they are attempting to edit is provided when applicable. This module simply provides the Wiki object with a username to use (or an unauthorised state), and the rest is left to the module's implementation. Usernames might be based on IP addresses, some form of login system, or something more bespoke, like tying into an institution's username and password database.

Figure 1: High-level Weerkat architecture



Conflict resolver: A conflict resolver defines the policy for merging conflicting versions into a suggested new version. The user can then review this alongside their submitted version and make any desired changes before resubmitting. The simplest conflict resolver simply provides the previous, clashing version, which results in a behaviour similar to the UseMod wiki engine, where the user must manually re-merge their changes with the conflicting version. A more advanced solution may attempt to automatically re-apply non-conflicting changes. This behaviour would be similar to that of collaborative version control software, such as CVS (Concurrent Versions System).

Differential engine: The differential engine is mostly simple, taking two node sources (which will have probably come from the node storage engine), and returning the results to render. Most variation here comes in how the differences may be presented: inline 'strikethrough' changes, UNIX 'diff' style, or even a two-column format.

Markup parser: The markup parser is responsible for converting a markup language into an abstract syntax tree (AST), which is used by the rendering engine. This provides separation between the markup language (which may need to differ between domains, depending on specialist needs), and the rendering of nodes. Some parsers may be intended for wikis where the users will be expected to be technologically confident enough to use XML-style markup, whereas others may employ the rather chaotic, but potentially simpler, special characters markup used by many existing wikis, such as UseMod and Mediawiki.

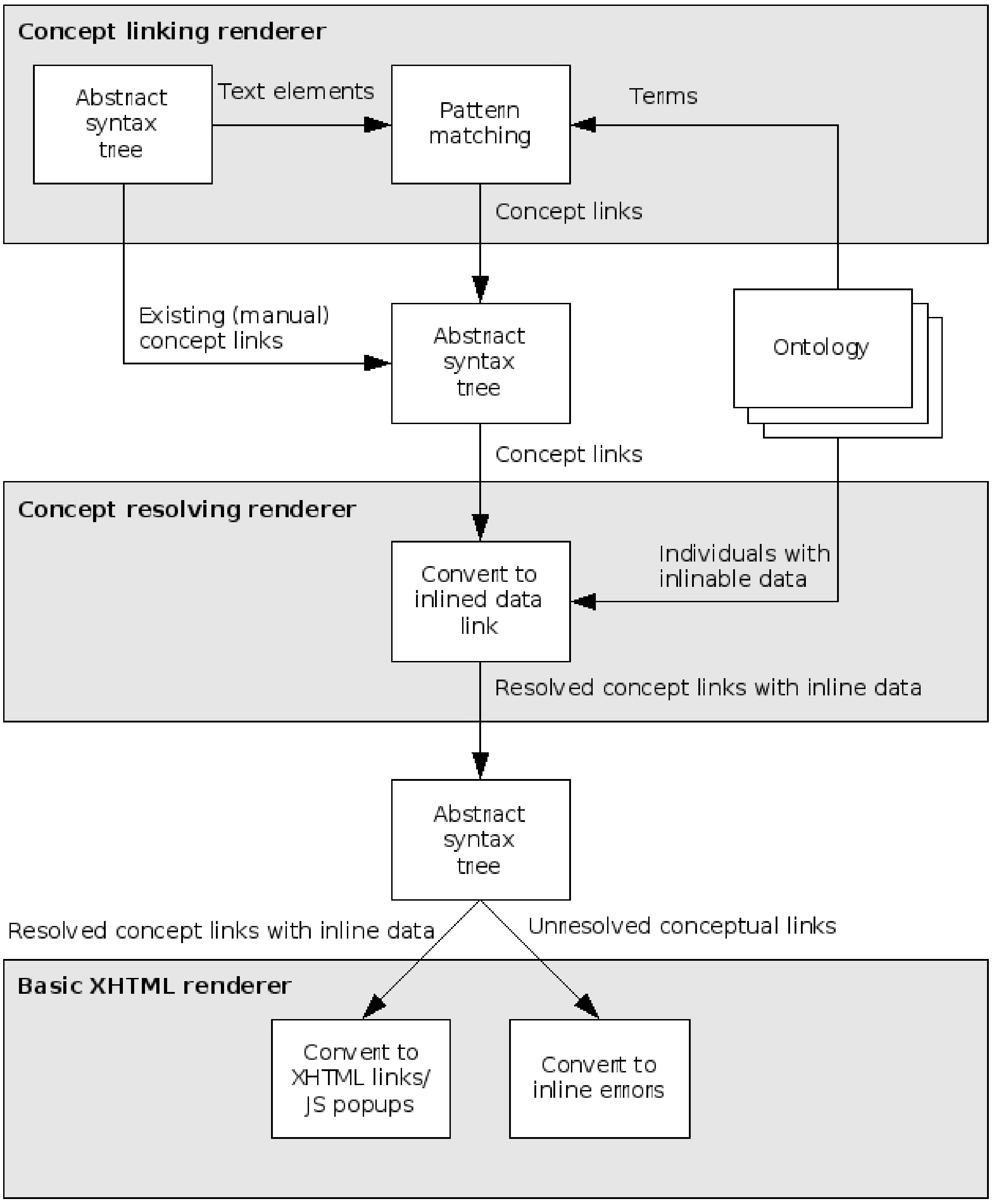
Rendering engine: The rendering engine's task is to convert the Abstract Syntax Tree (AST) of the markup parser into output: probably an XHTML fragment to be displayed in the web page, or potentially embedded elsewhere (for example, as a sidebar in a regular website). Since rendering is one of the most complex parts of the Wiki behaviour it in itself is broken down into a chain of renderers to allow even more flexibility. Each renderer is expected to convert some of the AST into output (or possibly simpler AST), with the results being passed onto the next renderer in the chain. The semantic linking itself can be implemented as a renderer, which looks at the information in the AST to find sections of prose, then adds hyperlinks in the form of more AST elements. A final core renderer converts basic AST elements, such as hyperlinks, into pure XHTML, such as the anchor element. Other renderers may perform tasks such as converting stylistic markup into inline CSS, or even passing through embedded SVG into the XHTML. It would even be possible to replace the core renderer with versions which generated other formats, such as plain text, or some future format for webpages. In the interests of complete modularity, even the chaining system itself is just a renderer.

Overall, by heavily modularising the design: the development load can be easily broken up into manageable segments; maintenance is made simpler by well defined boundaries, which reduce the need to know the entire code in detail; the Wiki is flexible enough to deal with a variety of different demands and future changes with a minimum of code modification.

3.3 Conflict resolution

It is necessary to detect edit conflicts so as to prevent data loss; however, it is not desirable to achieve mutual exclusion for the privilege of editing a node as the potentially web-based nature of the interface means that users may fail to complete and edit (and thus release their hold on the critical section) without notification reaching the system: one possible cause is a browser crash. Claims on the critical section cannot time out without either breaking the mutual exclusion or turning away edits that took too long too late: edits which users may have spent a lot of effort on in that time. Instead, conflict detection depends on knowing which version of a node an edit was based upon.

Figure 2: Semantic linking process overview


Concept linking renderer
Abstract syntax tree
Text elements
Pattern matching
Terms
Concept links
Existing (manual) concept links
Abstract syntax tree
Ontology
Concept links
Concept resolving renderer
Convert to inlined data link
Individuals with inlinable data
Resolved concept links with inline data
Abstract syntax tree
Resolved concept links with inline data
Unresolved conceptual links
Basic XHTML renderer
Convert to XHTML links/ JS popups
Convert to inline errors

3.3 Semantic Linking

The process for automatic linking is outlined in Figure 2. A semantic linking renderer is pre-loaded with ontologies to drive its linking. The renderer attempts to match terms from an ontology with the text within the node being rendered in order to create links: the use of the abstract syntax tree means that it is possible to determine which parts are plain text, as opposed to markup, and also which are already linked. Second, a resolving renderer looks up semantic links in the ontologies and attempts to inline appropriate data. Finally, the basic XHTML renderer converts these into XHTML for JavaScript popups to a page displaying the appropriate information

4. Implementation

Weerkat is implemented in Perl, and runs within a mod perl Apache environment. This combination provides a highly flexible scripting language with powerful object-oriented features, a large, central library of existing packages, and caching of compiled code to avoid latency.

4.1 Configuration

Storing the entire system configuration in a single XML file makes administration easier. The configuration file contains a section for each component, which is parsed into a tree using the standard Perl XML::Simple module: a common section is merged with this to allow configuration data to be shared, with the common section taking a lower priority such that it can be overridden. For example, both the node store and user identification and authorisation module may wish to share database configuration information. For the sake of compatibility, certain names exist to become reserved in the common section as more components are developed (for example, ‘Ontology’).

The sections in the configuration file also indicate the implementation to use, through a type attribute. For consistency, and to avoid special cases, only one implementation of each component can be specified. Renderers are chained using a ‘chain’ renderer which accepts as its configuration a list of other renderer configurations. The tree to be rendered is then passed through each one in turn. This way, the signature of every renderer—including the chain—is to accept a tree and return a tree.

4.2 S-ExpML

S-ExpML is the markup language developed for Weerkat, and is a combination of the terms ‘S-Expressions’ and ‘XML’. The syntax is based on pairs of brackets, and is more logical than the conventional Wiki markup approach of using an assortment of punctuation, which needs to be escaped when needed otherwise.

S-ExpML forms a tree structure with attributes, like XML. S-ExpML does not, however, use tags: elements are contained in brackets, e.g.: This is [emphasis S-ExpML].
The first whitespace-delimited token in the element is the element’s type—in this case emphasis. Everything after this is either content, which is tokenised, or attributes. Attributes are of the form NAME=VALUE. At present, the value isn’t quoted, but underscores are converted to spaces. Two underscores in a row can be used to represent a literal underscore. An ‘esc’ element is reserved for escaping. For the purposes of Weerkat, each line is treated as its own S-ExpML tree, and each becomes a paragraph.

Lexing and Parsing

The approximate grammar of S-ExpML is shown in Figure 3; the actual parser is slightly more lenient and will, for example, accept attributes after body content.
The S-ExpML markup parser in Weerkat uses a custom lexer and parser for performance reasons. The lexer tokenises strings upon the boundaries of whitespace and bracket characters. This converts a string such as ‘The [project Weerkat] tokeniser’ into a series of tokens ‘The’, ‘[’, ‘project’, ‘Weerkat’, ‘]’, ‘ ’, ‘tokeniser.’.

Figure 3: S-ExpML grammar

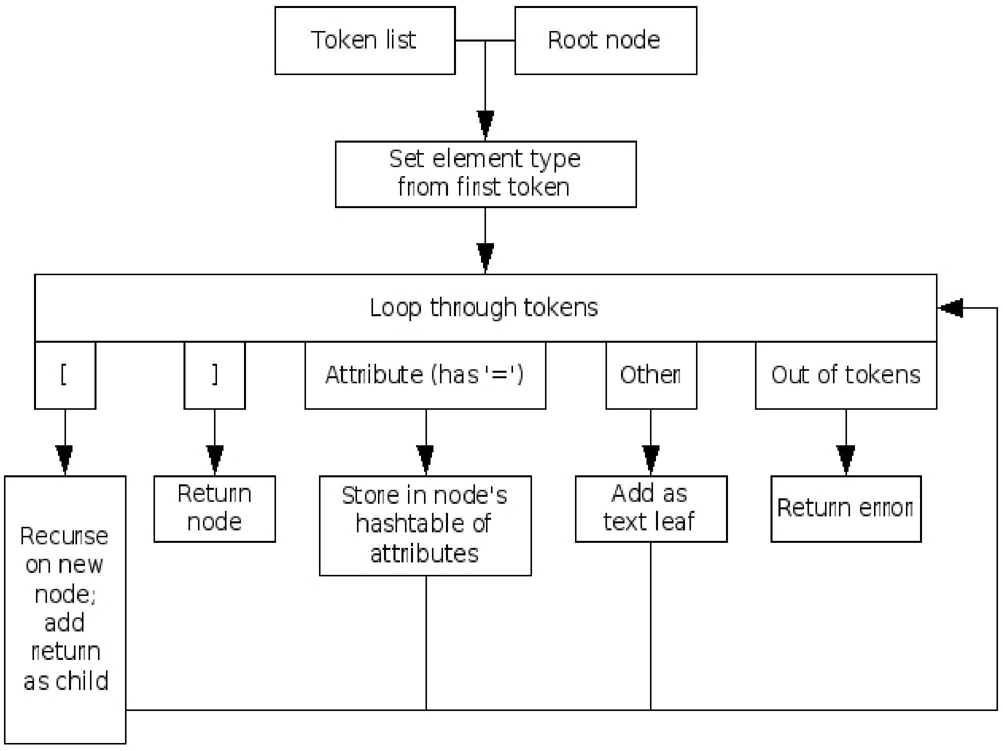
$$S \rightarrow [\ \mathit{text}\ \textsc{Attr}\ \textsc{Body}\ ]$$

$$\textsc{Attr} \rightarrow\ \mathit{text} = \mathit{text}\ \textsc{Attr}\ |\ \epsilon$$

$$\textsc{Body} \rightarrow\ \mathit{text}\ |\ S\ |\ \epsilon$$

The general algorithm of the parser is shown in Figure 4. This approach is more lightweight than a full recursive descent parser, and easier to maintain. The outer loop which feeds the source a line at a time into the parser, with a root node of a new paragraph in the output each time, it also checks that all tokens have been consumed after parsing; if not, then there was an additional closing bracket on the line which caused it to stop early, so an error is reported.

Figure 4: Recursive grammar parser structure



4.3 Abstract syntax tree

The abstract syntax tree is expressed as a data structure of AnnotatedTree subclass instances. There is a direct mapping from a tree node to an XML representation, with 'element' becoming the element name, the 'attributes' hashtable containing the attributes, and the 'children' array containing the content of the node, which may be further tree nodes or simple strings. As renderers act on the tree, parts are converted from 'semantic' to 'output'. It may even be desirable for the final output to be a single string, rather than a tree to convert to XML. If any 'semantic' nodes remain at the end, the known renderers have not been able to fully process the document, indicating an error condition.

Perl's XML::Simple package, used for the configuration, provides a very similar structure in terms of primitive Perl data structures, and an easy mechanism to convert this to and from XML. Unfortunately, it is not possible to use it for general document purposes, such as XHTML, as the Perl data structure it reads to and writes from does not preserve ordering over collections of heterogeneous elements. The semantic parts of the abstract syntax tree need to be at least informally defined for interoperability between markup parser and renderer implementations to be possible. Weerkat uses the following elements:

node: Root element, with a title attribute indicating the node's title.
para: Groups a set of elements and simple strings into one paragraph of text.
link: Generic link, with attributes type (one of node (inter-node link), uri (external link), concept (unresolved conceptual link), or uri concept (resolved conceptual link)), and to (the link target). A link to a node may have a revision or version attribute to link to a past version.
inlineprop: In-lined information about a concept; attributes match property labels to string values.
em: Emphasis; emphasise the contents of this node.
err: An inline error message, used to flag invalid markup.
nop: Semantically-empty container not to be rendered.
string: Strings indicate body text, which may contain arbitrary characters.

A node element should directly contain only para elements, which then contain all other kinds. Paragraphs may be nested, although renderers may ignore anything beyond the outer level; the same holds true for links, it assumes that the link for the longest string is the most specific (e.g. 'pork pie' over 'pie').

4.4 Automatic semantic linking

The automatic semantic linking information is held entirely in an OWL ontology: all classes which Weerkat deals with are subclasses of 'WeerkatConcept', and there is a DatatypeProperty from Weerkat- Concepts to strings called 'term', with an unconstrained cardinality. This provides the terms used by the linker in Figure 2. The information on which properties can be inlined is also held in the ontology. An AnnotationProperty 'inline' is defined, and applied to certain, suitable DatatypeProperties. The resolver tests if this annotation is present for any of the datatype properties of the individual it is inlining and, if so, adds appropriate inlineprop elements.

4.5 Rendering to XHTML

The abstract syntax tree maps neatly into XHTML, so much so that the XHTMLFinal renderer, rather than building an XHTML tree from scratch, simply recurses through the AST and converts it in-place. This is both faster and more memory efficient.

First, convert 'node' into 'div', and add a suitable heading for the node title if desired. 'para' nodes must be converted into 'p', with nested paragraphs being removed. Link behaviours vary by type. URI links are simply converted from being 'link' elements with 'to' attributes into 'a' elements with 'href' attributes; node links act similarly but need to point to the URL for viewing the page on the wiki, and need to set the CSS class of the link depending on whether or not it exists. Conceptual links need to be converted into links to the concept viewing page with an 'onclick' event for the JavaScript popup code.

To prevent attributes which have been read directly into the AST from the S-ExpML parser making they way into the XHTML output, attributes which are not explicitly set by the renderer are culled: this prevents malicious users from being able to inject JavaScript or other scripts into the page.

4.6 User interface

The user interface has been designed to be as accessible as possible: the font size is the browser default, color is not used as the sole method for conveying information and the page is constructed from the correct semantic elements (for example, the list of links at the top is actually an XHTML list).

4.7 Meeting the requirements

We identified a number of requirements that a Semantic Wiki that supports ontological hypertext should fulfil. Of these, all the basic Wiki requirements (1–5) have clearly been satisfied by Weerkat. The Semantic requirements have been mostly fulfilled. A markup language has been developed that is sufficiently minimalist so to aid semantic extraction (requirement 6); it is, however, semantically, rather than stylistically, oriented. The ontology is applied across all nodes by the SimpleConceptLinker (requirement 7). Our tests with Weerkat have used a sample ‘programming languages’ ontology. Multiple ontologies are supported (requirement 8), although this has only been tested informally. User defined ontologies are not yet supported (requirement 9) – although users might submit ontologies for the Wiki moderators to add. Weerkat does satisfy the general requirements for extensibility, its architecture is highly modular (requirement 10), with a chain of renderers capable of building up functionality and simple configuration via a single XML file (requirement 11). Weerkat also passes the final usability requirements. The web interface passes the W3C validation tests (requirement 12), has been tested on many browsers (requirement 13), and has been designed with accessibility in mind (requirement 14).

5. Conclusion

In this paper we have described our work to bring semantics to the Wiki concept and presented Weerkat, a Modular Wiki that demonstrates its extensibility by implementing ontological hypertext as a module.

We now hope to explore the other semantic functionality alluded to by the Platypus work but within our own extensible architecture. It should be simple to add functionality to export Wiki pages as Semantic Web resources, a greater challenge is to analyse the resulting semantic graph and derive the user’s implicit ontology. However there are some practical uses for Weerkat as it stands today. For example it could form the bases of a collaborative authoring tool and allow users to relate concepts as they are writing (Wills et al, 2005:69-89). By being used in this way it may also improve data quality between different expert domains, as the concepts and terminology of each domain can be noted and referenced clearly by the others (Parker, 2004).

We believe that just as the original WikiWikiWeb made hypertext authoring a possibility for non-technical users, so Semantic Wikis could make the Semantic Web accessible to the same group. By using evolving ontologies, and meaningful markup, users could construct semantic graphs and develop new ontologies without ever seeing an RDF statement or

OWL declaration. Weerkat is an extensible modular semantic wiki, and it is our hope that it could be used as a basis for future Semantic Wiki development.

6. List of references

Berners-Lee, T., Hender, J. and Lassila, O. 2001. The Semantic Web. *Scientific American*, May 2001.

Campanini, S.E., Castagna, P and Tazzoli, R. 2004. Platypus Wiki: a Semantic Wiki Wiki Web. *SWAP, Semantic Web Applications and Perspectives*, Ancona, December 2004.

Carr, L., Hall, W., Bechhofer, S. and Goble, C. 2001. Conceptual Linking: Ontology-based Open Hypermedia. *In Proceedings of Tenth International World Wide Web Conference*, Hong Kong, May 1-5: 334-342.

Collier, G. H. 1987. Thoth-II: hypertext with explicit semantics*. In Proceeding of the ACM Conference on Hypertext (Chapel Hill, North Carolina, United States)*. HYPERTEXT '87. ACM Press, New York, NY, 269-289.

Davis, H., Hall, W., Heath, I., Hill, G., and Wilkins, R. 1992. Towards and Integrated Information Environment with Open Hypermedia Systems. *ECHT'92 European Conference on Hypertext 1992*. Milano, Italy, ACM: 181-190.

El-Beltagy, S. R., Hall, W., De Roure, D., and Carr, L. (2001). Linking in context. In Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia (Århus, Denmark, August 14 - 18, 2001). HYPERTEXT '01. ACM Press, New York, NY, 151-160.

Grønbæk, K. and Trigg, R.H. 1994. Design issues for a Dexter-based hypermedia system. *Communications of the ACM,* 3(2):40-49.

Gruber T. R. 1993. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199-220

Leuf, B. and Cunningham, W. 2001. *The Wiki way: quick collaboration on the Web*. Boston: Addison-Wesley

Miles-Board, T., Kampa, S., Carr, L. and Hall, W. 2001. Hypertext in the Semantic Web. *Proceedings of Twelfth ACM Conference on Hypertext and Hypermedia (HT'01)*. Aarhus, Denmark: 237-238.

Millard, D. E., Gibbins, N. M., Michaelides, D. T., and Weal, M. J. 2005. Mind the semantic gap. *Proceedings of the Sixteenth ACM Conference on Hypertext and Hypermedia*. Salzburg, Austria, September 06 - 09, 2005.

Page, L., Brin, S. Motwani, R and Winograd, T. 1999. The PageRank Citation Ranking: Bringing Order to the Web, *Stanford Digital Library working paper* SIDL-WP-1999-0120

Parker, M. 2004. A generic business intelligence data model to analyse data within a small to medium medical practice (SMMP). Conference Paper. *South African Institute of Computer Scientists and Information Technologists (SAICSIT)*, October 2004. Stellenbosch, South Africa: 111-114.

Schraefel, M., Shadbolt, N. R., Gibbins, N., Harris, S., and Glaser, H. 2004. CS AKTive space: representing computer science in the semantic web. *Proceedings of the 13th international Conference on World Wide Web*. New York, NY, USA, May 17 - 20, 2004.

Weal, M. J., Hughes, G. V., Millard, D. E., and Moreau, L. 2001. Open hypermedia as a navigational interface to ontological information spaces. *Proceedings of the Twelfth ACM Conference on Hypertext and Hypermedia* . Århus, none, Denmark, August 14 - 18, 2001.

Wills, G., Miles-Board, T., Bailey, C., Carr, L., Gee, Q., Hall, W. and Grange, S. 2005. The Dynamic Review Journal: a scholarly archive. *New Review of Hypermedia and Multimedia*, 11(1):pp. 69-89 [Online]. Available WWW: http://eprints.ecs.soton.ac.uk/11154/ (Accessed 16 May 2006).