SemWiki*

A RESTful Distributed Wiki Architecture

Max Völkel
Institute of Applied Computer Science and Formal Description Methods (AIFB)
Englerstrasse 11, D-76131 Karlsruhe, Germany
mvo@aifb.uni-karlsruhe.de

ABSTRACT

Current Wiki engines are mostly monolithic applications which intermingle parser, user interface and data management backend. In this paper we show how these three components can be realised as lightweight, REST-style web services. We explain why this separation is useful and how the wiki community benefits from such an approach. Additionally, the presented wiki allows semantic statements and queries over the model.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.12 [**Software Engineering**]: Interoperability

General Terms

Design

Keywords

Wiki, REST, System Architecture, Web Service, Semantic Web, Semantic Wiki

1. INTRODUCTION

Wikis have been proven by massive usage to be a useful tool for collaborative note-taking. Several efforts try to bring more database-like structures and search capabilities to wikis. The approach presented in this paper is characterised by two main contributions: First, the system architecture is realised as a distributed set of REST (REpresentational State Transfer, c. f. [1])-style web services. Second, the presented wiki-engine *SemWiki* allows to create and query RDF-style statements within the wiki paradigm. This paper concentrates on the distributed architecture of SemWiki.

*This research was partially supported by the European Commission under contract FP6-507482. The expressed content is the view of the authors but not necessarily the view of the *Knowledge Web Network of Excellence* as a whole. The author would like to thank Werner Thiemann for his help.

WikiSym'06, August 21-23, 2006, Odense, Denmark.
ACM 1-59593-413-8/06/0008.

```
 <!ELEMENT Page (PageName, PageContent)>
<!ELEMENT PageName (Lexical)>
<!ELEMENT PageContent (Item*)>
<!ELEMENT Lexical (#PCDATA)>
<!ELEMENT Item (Text | Resource | Query )>
<!- text with linked WikiWords ->
<!ELEMENT Text (TextContent | WikiWord)*>
<!ELEMENT TextContent (Lexical)>
<!ELEMENT WikiWord (Lexical)>
<!- semantic statements and queries ->
<!ELEMENT Resource (Lexical, Property*)>
<!ELEMENT Property (Lexical, Object*)>
<!ELEMENT Object (Lexical)>
<!ELEMENT Query (Lexical)>
```

Figure 1: Parser Output Format as DTD

2. DISTRIBUTED WIKI ARCHITECTURE

We believe, wikis are popular due to their simplistic approach. The time needed to learn how to use a wiki is less than for most content management systems or HTML editors. By restricting the features to a manageable minimum, this became possible. Today, we face a situation where hundreds of different wiki engines exists. All have a different syntax and different user interfaces. In order to re-use existing wiki engine code, one could either use the component approach from software engineering and write, e. g. a new parser for an existing wiki engine. The new parser could be deployed on the server and be used. Unfortunately, even with good software design, the *deployment* of a new parser is far more complicated than editing a wiki page. Additionally, the new parser would work only in the wiki it was written for.

We propose the modelling of a wiki parser as a REST web wervice[1]. In order to make integration easy, the parser should accept HTTP GET method calls with a parameter „wikitext". The method should return an XML representation according to an agreed structure. In Fig. 1 we present a simplified XML Document Type Definition (DTD). It supports no wiki formatting beyond WikiWords, but offers the option to assert statements and pose queries.

The page rendering side of the user interface should also be an external application. This leads to a system architecture, as depicted in Fig. 2. Now parsers can be re-used across user interfaces and different user interfaces can operate on the same data. Even rich clients on the desktop can be part of the game.

In order to make this possible, the interfaces have to be defined precisely. REST advocates the use of HTTP methods and leaves

[1]See http://www.xfront.com/REST-Web-Services.html

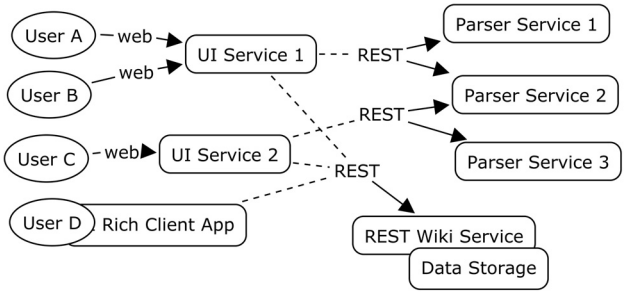

Figure 2: Wiki Web Service Architecture

open the data format for message exchange. As XML is the defacto standard for message exchange, we define the messages between the wiki services as XML messages. We considered using RDF serialised as RDF/XML for message exchange, but this would not be easy to read by humans and the technology is not as mature as XML is.

REST preaches a resource-centric thinking, not a method-call approach like XML-RPC[2]. A "get" should not change the internal state of a resource, a "put" should set it, "delete" resets the state and "post" produces a new state, taking into account the old state. HTTP POST thus is the most powerful method with the least well-defined semantics.

2.1 REST Wiki

Unlike most wiki engines, *SemWiki* addresses pages not by page titles but by URIs. By explicitly using globally unique identifiers, data merge and exchange operations become much easier.

HTTP GET /wiki?uri=*page uri*
Fetches the XML document for the given page valid for `wikipage-out.dtd`

HTTP GET /wiki?uri=*page uri*&format=rdf
Returns an RDF representation (serialised as RDF/XML) of the page's content

HTTP PUT /wiki?uri=*page uri* ; body = {*newxml*}
Expects the HTTP body to be an XML document according to `wikipage-in.dtd`. Stores content in internal model.

HTTP POST /wiki?uri=*page uri* ; body = {*oldxml*,*newxml*}
Calculates the diff from old version to new version and applies it to the wiki data model. By explicitly submitting the old content again, the *intended change operation* can always be calculated correctly. Whether the diff causes a conflict or not is another question.

HTTP DELETE /wiki?uri=*page uri*
Simply deletes this page from the wiki data model.

2.2 REST Parser

The REST Parser has no internal state and supports only one HTTP method:

HTTP GET /wiki?uri=*page uri*&wikitext=*text*
Parses the text given as "wikitext" and returns an XML representation according to `wikipage-in.xml`.

[2]http://xml-rpc.org

2.3 User Interface

The user interface is the integration point. When the user wants to see a wiki page, the UI calls the REST Wiki, renders the XML as HTML and adds additional navigational elements. E. g. the typical breadcrumb trails would be implemented in the user interface layer. When a user wants to edit a page, the user interface has to transform the XML document to the correct Wiki syntax – a simple XSLT stylesheet can do this. The user edits the text and the UI sends it to the parser service. The resulting XML is then PUT or POST to the REST wiki. A new GET is issued and the result is rendered for the user.

3. CURRENT IMPLEMENTATION

The current implementation supports the three components REST Wiki, parser and user interface. All are running on different servers, implemented as Java servlets running on instances of the Jetty web server. At startup time, the user interface needs to ne called with a `GET ?config=`*config URL* call in order to set the URLs for the REST wiki, the parser and the used XSLTs. XSLTs are used to render the content as HTML or text suitable for editing. The displayed pages are rendered using Velocity templates. The REST Wiki uses internally RDFReactor[3] in order to manipulate and query the internal RDF model as Java objects. RDFReactor itself manipulates a Jena[4] model.

4. DEMONSTRATION

In the demonstration we will show the individual web services, as they can all be tested with any browser. In order to overcome the limitations of browsers to support only GET and POST, we added a parameter `method=`*(GET | PUT | POST | DELETE)*. We will show how each module can be debugged easily and independently. Then we will show how the different web services operate together and form a semantic wiki. In particular, we will show how the same content can be accessed by machines using the primitive REST wiki services and the human-oriented web-user interface. We will also show how semantic statements can be made and how this „background knowledge" can be queried. Query results are rendered as tables.

5. CONCLUSION AND FUTURE WORK

We have shown how wikis can be opened up for machine access by separating the wiki functionality from the user interface using lightweight REST-style web services. This allows wiki engines to adapt faster to user needs, as crucial parts of the wiki ecosystem can now be developed *and deployed* independently.

6. REFERENCES

[1] R. T. Fielding. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, U. Mass., 2000.

[3]rdfreactor.ontoware.org

[4]jena.sf.net