SemWiki

A RESTful Distributed Wiki Architecture

Max Völkel
Institute of Applied Computer Science and Formal Description Methods (AIFB)
Kollegiengebäude am Ehrenhof, 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

Keywords

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

1. INTRODUCTION

Wikis have been proven by their continuing 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)[1]-style web services. Second, the resulting wiki allows to create and query RDF-style statements within the wiki paradigm[2]. This paper will concentrate on the distributed architecture of SemWiki.

2. DISTRIBUTED WIKI ARCHITECTURE

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 form software engineering and write, e. g. a custom parser for e. g. the wiki engine MoinMoin. Later the new parser can be deployed on the server and be used. In effect, even with good software design, the set up of a new parser is far more complicated than editing a wiki page. Additionally, the new parser would work only in MoinMoin.

We propose the modelling of a wiki parser as a REST web wervice[3]. In order to make integration easy, the parser should accept HTTP GET method calls with a parameter ,,wikitext". The method should return an XML response according to an agreed structure. For SemWiki we defined such a structure (see Figure 1) as an XML Document Type Definition (DTD). It supports no wiki formatting beyond WikiWords yet, but offers already the option to assert statements and pose queries.

```
<!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

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

In order to make this possible, the interface have to be

[1]A term coined by Roy Fielding
[2]RDF Schema for the REST Wiki available at http://purl.org/net/xamde/ns/semwiki

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

WikiSym '05 San Diego, California USA

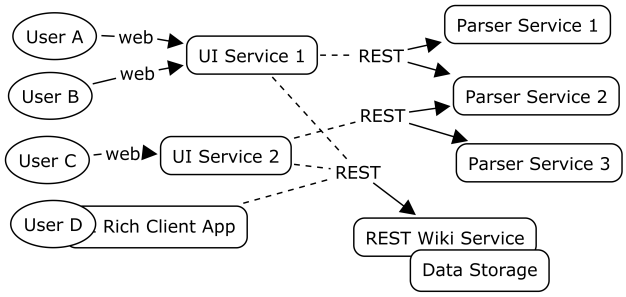

Figure 2: Wiki Web Service Architecture

defined precisely. REST advocates the use of HTTP methods and leaves open the data format for message exchange. As XML is the de-facto standard for message exchange *and* human readable, 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. 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, database merge 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***`&newxml=`***xml***
: Expects newxml to contain an XML document according to `wikipage-in.dtd`. Stores content in internal model.

`HTTP POST /wiki?uri=`***page uri***`&oldxml=`***xml***`&newxml=`***xml***

: 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 calculate 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`.

2.3 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. 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 total pages are rendered using Velocity templates. The REST Wiki uses internally RDFReactor[4] in order to manipulate and query the internal RDF model as Java objects. RDFReactor itself manipulates a Jena[5] 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 primitiv 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 simple REST-style web services. This will allow wiki engines to adapt even faster to users needs, as crucial parts of the wiki ecosystem can now be developed *and deployed* independently.

6. ACKNOWLEDGMENTS

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.

[4] rdfreactor.ontoware.org
[5] jena.sf.net