# The Radeox Wiki Render Engine

Matthias L. Jugel
Fraunhofer Institute for Computer Architecture and Software Technology
Kekuléstr. 7
12489 Berlin, Germany
matthias.jugel@first.fraunhofer.de

Stephan J. Schmidt
Fraunhofer Institute for Computer Architecture and Software Technology
Kekuléstr. 7
12489 Berlin, Germany
stephan.schmidt@first.fraunhofer.de

## ABSTRACT

The Radeox Wiki markup render engine is a basic component for the construction of a Wiki or any system that wishes to integrate basic Wiki functionality. With the availablility of such a component the compatibility of different Wiki systems can be improved and the simplicity of the Wiki way is now ready for deployment in business applications. This paper explains how this component emerged from its host Wiki SnipSnap and how it enables software developers to integrate the Wiki way into their own implementations and other software.

### Categories and Subject Descriptors

D.3.4 [**Programming Languages**]: Processors; I.7.2 [**Document and Text Processing**]: Document Preparation

### General Terms

Wiki, Text Rendering, Software Components

### Keywords

wiki, markup, conversion

## 1. INTRODUCTION

At first, Radeox [1] was not a standalone software component, but rather the markup translator for SnipSnap [2]. SnipSnap is a Java based Wiki implementation developed by the Fraunhofer Institute for Computer Architecture and Software Technology. Its background stems from knowledge management and is intended to be part of projects researching the benefits of knowledge based technologies in the software development process. This integrative approach required a componentized and easily reconfigurable architecture.

SnipSnap gained a fair amount of interest from researchers as well as companies looking for a Java based implementation with special support for the software development process. It is available as an Open Source project under the terms and conditions of the GNU General Public License.

This enabled developers to learn from the code and to contribute. However, the viral effect of the GPL kept commercial companies from using SnipSnap software components. Upon a request by Atlassian, the makers of Confluence [3], it was decided to extract the Wiki markup rendering from SnipSnap and to distribute the sources as a standalone software component, called Radeox. It is available under a relaxed, BSD style, license. This enabled many projects like XWiki [4], Blojsom [5] and others [6][7][8][9][10] to use Radeox and to focus on important features of their Wiki implementations.

## 2. MOTIVATION

All Wikis have a similar architecture consisting of a web based user interace, a middle layer of wiki aware content handling logic and a storage backend. An example of a Wiki architecture is shown in the Wiki Architecture figure. Although it is very common to use standard frameworks for the user interface as well as the storage backend the wiki content handling is usually custom made software.

The drawbacks of custom made Wiki handling software are obvious. Not only do all the different implementations have different semantics in how they handle Wiki content, they also bind users technologically to proprietary software. Also, the unavailability of a Wiki content renderer as a software component is a disadvantage for wiki technology in general.

Most Wiki implementations only understand one special flavor of Wiki markup and it is usually difficult to switch to a different flavor without losing content. However, the harmonization of the markup is an important issue within the Wiki community.

Providing software components for handling Wiki markup would be a benefit for Wiki and application developers. Since many different flavors exist such a component must be able to handle the rendering as well as the translation between them.

Before describing the architecture and application of Radeox we will take a look at Wiki markup as well as different ways to handle the translation into HTML or other output formats. At the end of this section the benefits and drawbacks of the different methods are discussed.

### 2.1 Wiki Markup

Basic Wiki markup consists of tags found in plain text documents that denote a change in the presentation of the

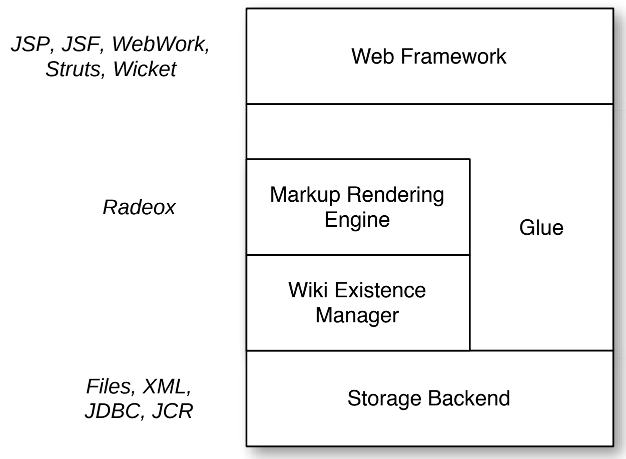

Figure 1: Wiki Architecture

enclosed or following text. Simple tags are commonly available for bold, italic and strike-through text:

```
__bold text__, ~~italic text~~, --deleted text--
```

For general text structurization heading styles as well as other markup similar to HTML tags are available. These markup tags are usually valid for the rest of a line or on their own, like in the case of a horizontal rule.

Somewhat more complicated markup exists for lists, tables and macro-style tags. While simple markup, headings, lists and tables are common text components and can usually be implemented by simple replacement techniques, macros allow the implementor of a Wiki more freedom in how content enclosed by a macro is rendered. For example, in SnipSnap macros allow the inclusion of special content into the output, such as the rendering of tree based data using a special notation (see figure 2):

```
{graph} (root (child1) (child2 (child3))) {graph}
```

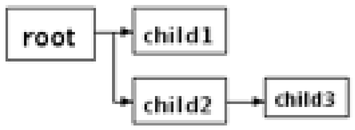

Figure 2: graph macro result

Many implementations support different flavors of the basic markup including headings and lists. Others additionally have markup similar to SnipSnaps macros. However, the latter is usually incompatible to most of the other Wiki implementations.

There has been quite some effort to standardize Wiki markup and to make at least the basic markup interoperable, the latest started after WikiSym 2005 [11].

## 2.2 Rendering Methods

### 2.2.1 Custom Markup Parsing

Implementations of manual parsers are most common. Even Wikipedia [13] with a demand for high performance uses a custom written PHP based parser. Custom implementations can be very flexible as they allow the use of side effects, like breaking out of the parsing to do collection for a table of contents. The structure of such a parser is usually similar to what parser generators produce by recursively looking for begin and end marks and exchanging the text in-between using rules.

### 2.2.2 Regular Expression Parser

Regular expressions can be used whenever text portions have to be replaced using matching algorithms. These expressions are very powerful and have good performance. Since they are not aware of a hierarchy, regular expressions simply match everything available and the developer decides the order of replacing and changing matched text.

### 2.2.3 Parsers Using Context Free Grammars

From a software developer's point of view using a parser generator is preferable over any other custom made parser. Using a parser generator allows the definition of a grammar for the markup language with strict syntax checking . These parsers generally have excellent performance and work by presenting the developer the text contents found by defined syntactical constructs.

### 2.2.4 Discussion of the Methods

For quick implementations a custom markup parser may be preferable over regular expression or generated parsers. Results are available in a short time and sufficient for many Wiki implementations.

However, the flexibility such a custom parser offers also poses a threat to the whole implementation. Custom parsers are prone to errors. Maintenance and understanding are time-consuming. In addition, these parsers can be very inefficient when working with large amounts of text. That is reason enough to consider generated parsers or regular expressions for handling Wiki markup.

For Radeox, the decision to use regular expressions has two reasons. First of all, regular expressions can be exchanged relatively easy during runtime. This is hardly possible with generated parsers and even more difficult using a custom parser. Second, regular expressions are explicitely made for efficiently matching text.

These benefits outweigh even the drawback that there is no control over the context where a match for some Wiki markup occurs in the text. Simple markup is declared by a match and replacement rule and more complicated markup, like macros, allows the execution of arbitrary code. Using Radeox it is possible to sort the rules to define preference of the matching within the rendering pipeline.

## 3. ARCHITECTURE OF RADEOX

The following chapter describes the architecture of a Wiki markup renderer using Radeox as an example. First, the pipeline architecture is explained and how its behaviour can be adapted to the needs of a specific Wiki implementation.

## 3.1 Rendering Pipeline

A Wiki Render Engine basically transforms patterns into other patterns as described above. Each transformation should be independent, so the transformations can be changed and extended independently. New transformations can be added without influencing other code. For extending the

Wiki markup with special features like the graph macro, a simple extension mechanism is needed. This mechanism should be implemented as a extension of the transformation mechanism.

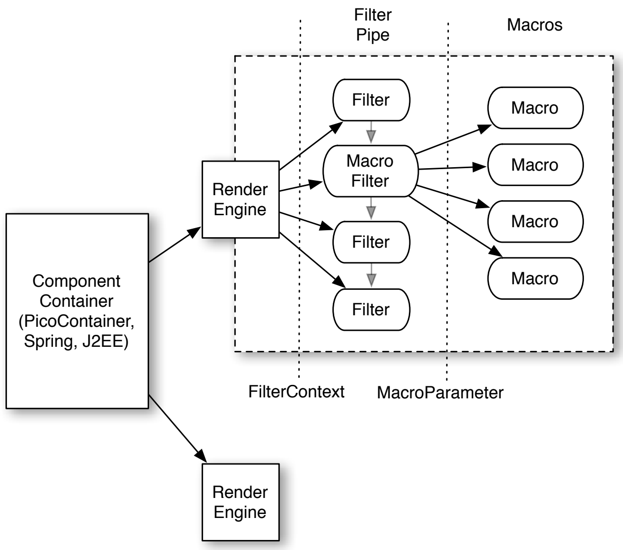

Figure 3: Radeox Architecture

#### *3.1.1 Filters*

The described transformations can be implemented with filters. A filter takes an input pattern and transforms the pattern into some output. In Radeox filters are the basic operating unit. Those filters take text input and create text output. This mechanism is too low level for developers to create useful filters. Therefore higher abstractions should be added. In Radeox these higher abstractions are regular expression filters. They use patterns expressed as regular expressions to transform the input into some output. They cover most of the wiki markup.

#### *3.1.2 Macros*

The extensions for special applications can be realized with macros. Macros are simple extensions for dynamic behavior and do not necessarily transform input into output, like filters. They take some information form a source and inject the information into the wiki page. Usage scenarios include displaying currently active users, browsing a source code repository or showing Google adsense advertisements. As with the graph macro they do not need to render text output but can use graphics or non-visible semantic web markup. Macros are referred by their name and beside the current render context they can be customized with parameters. In Radeox macros are easily implemented by inheriting from a base class.

### 3.2 Selecting Radeox Behavior

The described architecture and extensions are aimed at developers. But some users of a wiki render engine might want to customize the engine for their needs without writing code. A render engine therefore needs to be customizable with a simpler mechanism.

Locales are a successful application developmen strategy to customize software with different languages. Most language frameworks support this in one way or another. Seeing render engine customizations and different wiki markups the same way as different languages like German and English in the application development domain, locales are an ideal solution to customzing wiki engines without the need to write code. In Radeox most markup patterns and output results are stored in languages bundles. So adapting Radeox to one's needs is as simple as replacing the locale files with custom markup and transformation rules. Markup, like Wikipedia or Textile [12], can be easily supported with different locales.

How different the Wiki markup styles are shows the following example of the markup dialects from Texttile, Wikipedia [13] and Radeox:

```
h1. Textile Header 1
h2. Textile Header 2
h3. Textile Header 3

==Wikipedia section==
===Wikipedia Subsection===
====Wikipedia Sub-subsection====

1 Radeox First Order Heading
1.1 Radeox Second Order Heading
1.1.1 Radeox Third Order Heading
```

### 3.3 How Radeox fits in a wiki architecture

An expressive Wiki render engine is the heart of each wiki implementation. The render engine encapuslates the wiki domain. Around this engine the other parts of a wiki a very generic and can be found in weblog, content management or document management systems.

A wiki needs four applications parts: The web frontend or GUI part, the storage backend, the Wiki render engine and some glue code to connect the other parts. The web frontend needs views and forms to display, edit and create new wiki pages and can easily be implemented with web frameworks like JSF [14], Struts [15] or WebWork [16]. The storage backend is most of the time a thin database layer or object relational mapper. Some implementations use XML storage in files or databases. New wiki implementations should use higher abstractions which suit their content storage needs better.

One such layer ist the common content managment API JSR [22] which abstracts common backends like databases for content storage. With such a backend content repositories of wikis, content and document management systems can be consolidated into one backend. With Radeox the wiki render engine part as the heart of the wiki implementations needs only a very thin adaption layer for integration. Some glue code to integrate the three other parts of a wiki application finishes the implementation.

For example, it was very simple to write a wiki application with Grails [17]. This is a web framework combined with an easy to use object storage mechanism. With the help of Radeox and the Groovy [18] scripting language as the glue code, it took the authors around thirty minutes to implement a simple wiki.

### 3.4 Example

A wiki render engine needs to be simple to use with a small and clean but powerfull API. This enables developers to integrate the render engine in their applications with thin adaption layers and will result in higher acceptance of the render engine. In Radeox only two classes are needed for rendering wiki markup:

```
RenderContext context = new BaseRenderContext();
RenderEngine engine = new BaseRenderEngine();
String result = engine.render("__Radeox__", context);
```

The rendering takes part in the render method of the RenderEngine object. Filters and macros in Radeox need a context which is supplied to the render call. Radeox is implemented with Java interfaces to make it customizable and parts exchangable, but delivers default implementations like BaseRenderEngine and BaseRenderContext. The engine executes all known filters and all known macros and transforms __Radeox__ into HTML: <b class="bold">Radeox</b>.

## 4. FUTURE WORK

### 4.1 Markup Bundles

It has become apparent that while most users of Radeox use the bundled markup it would make sense to provide a set of bundles that contain the most commonly used markups. Especially since the popularity of Wikipedia a Wikipedia bundle should be standard. Additionally, bundles for conversion between different markup syntax might prove useful and would be a powerful tool to aid compliance testing should there be a standard Wiki markup language.

### 4.2 Better Context Sensitivity

As discussed in the rendering methods, regular expression matching is not context sensitive in that it is never known what other markup might surround the current match. Providing such information to implementors of complex rules and macros would improve the text conversion results. It would also allow developers to selectively execute macros depending on the current context.

### 4.3 Combined Parser

Since very early implementations of Radeox an improvement was discussed, that would combine generated parsers with regular expression matching: a runtime customizable generated parser. Such a software component would allow to create a high performance parser with strict syntax and context sensitive execution of matching rules while maintaining the flexibility of changing the runtime behavior of the rules.

An important goal here is to keep the simple way to create matching rules. This is, unfortunately, currently not possible with existing parser generators. Also, all discussions with parser generator experts lead us to the conclusion that such a piece of software is possible. However, parser research has a different focus.

## 5. CONCLUSION

### 5.1 Simple to Integrate

Our own work with Radeox and comments from the user community show that a software component for rendering Wiki markup into HTML or other formats was necessary. The componentized architecture makes it possible to integrate Radeox quickly into all kinds of applications. This combined with the great configurability makes Radeox a powerful tool with chameleon-like features with respect to Wiki markup.

### 5.2 Ready for Business Applications

This simplicity combined with the power of a being a business level component makes Radeox ready to be integrated and deployed in business applications. Even though there is still much to be desired, as described in future work, the flexibility of the Radeox framework together with the liberal BSD like license will help the Wiki way into business for better content or knowledge management.

## 6. RELATED WORK

The Radeox project aims at providing a software component for generic translation of Wiki markup into other languages for Java. Other projects exist, such as anyMeta Wiki [19], a markup to HTML translator. It aims at providing a two-way translation of markup to HTML and back. Like PmWikis [21] parser and the Text_Wiki [20] component it is a PHP based implementation. A common feature of all these Wiki parser implementations is that they use regular expression rules to translate the markup into a designated target language.

All efforts try to replace the custom made Wiki parsing layer with standardized components for selected programming languages. This will help in the standardization and harmonization of Wikis which is an important precondition for industry acceptance.

## 7. REFERENCES

[1] Radeox; A wiki markup render component; http://radeox.org
[2] SnipSnap; Fraunhofer FIRST, Matthias L. Jugel and Stephan J. Schmidt;http://snipsnap.org
[3] Confluence, Atlassian Software Systems Pty Ltd; http//confluence.atlassian.com
[4] XWiki; http://xwiki.org
[5] Blojsom; http://www.blojsom.com
[6] Orionsupport; http://orionsupport.com
[7] Galena; http://galena.sphene.net
[8] Biscuit; http://biscuit.javanicus.com
[9] Blogunity; http://blogunity.sourceforge.net
[10] Pebble; Blogging Tools written in Java; http://pebble.sourceforge.net/
[11] WikiSym 2005; Conference Wiki; http://wikisym.org/ws2005
[12] Textile; A Humane Web Text Generator; http://www.textism.com/tools/textile/
[13] Wikipedia Wiki Markup;http://en.wikipedia.org/wiki/How_to_edit_a_page
[14] J2EE JavaServer Faces Technology; http://java.sun.com/javaee/javaserverfaces/
[15] Apache Struts; http://struts.apache.org/
[16] WebWork; http://www.opensymphony.com/webwork/
[17] Grails Application Framework; http://www.grails.org
[18] Groovy Scripting Language; http://groovy.codehaus.org
[19] anyMeta Wiki; http://source.mediamatic.nl; Mediamatic wiki translator
[20] Text_Wiki; A php wiki markup parser; http://wiki.ciaweb.net/yawiki/index.php?area=Text_Wiki
[21] pmWiki; http://www.pmwiki.org, rule based wiki translator
[22] JSR-170; Content Repository for Java technology API; http://www.jcp.org
[23] Wicket; http://wicket.sourceforge.net/