Wiquila – a Wiki rich client that mixes well with other sources of software project information

Tim Romberg
FZI Research Center for Information Technology
Haid- und Neustr. 14
76131 Karlsruhe, Germany
romberg@fzi.de

ABSTRACT
Wikis are well-established as tools for freely-structured knowledge exchange, and especially popular in the software engineering community. However, there remain many challenges regarding usability, productivity, and integration with existing information sources. This paper presents Wiquila, a rich client able to provide comfortable Wiki authoring either for “legacy” Wiki engines, or for the knowledge base we develop in the same project. Wiquila allows easy insertion of dynamically updated references to internal and external resources.

Keywords
Wikis, Knowledge Management, Authoring, SOA, Java Swing, Rich Client, Hypertext, Usability, Open Source

1. INTRODUCTION
Wikis are an established tool for knowledge sharing and documentation in software projects. They can especially play out the strengths of their hypertext content model in knowledge domains that are themselves rather ad-hoc-structured or semi-structured, and where the content lifetime is rather long, such as the pattern library maintained in the original WikiWiki by Ward Cunningham, background knowledge about a problem domain (e.g. legislation that the target software has to conform to), or initial vision and requirements analysis. On the other hand, they are less well-suited for much of the day-to-day project information management in professional software development, with more structured data on work items (bugs and feature requests), releases, automatic build and test management, and test coverage, where specialized applications and integrated Application Lifecycle Management suites such as Polarion or CodeBeamer\(^1\) provide specific workflows, analysis and monitoring tools, etc.

Several strategies of bridging the gap between “unstructured” / unspecialized Wikis and structured and specialized tools are currently being proposed. One strategy consists of adding structured data management features and software-engineering-specific tools to Wikis. For example, SnipSnap, developed at Fraunhofer FIRST\(^2\) allows collaborative editing of UML diagrams within a Wiki page using special syntax. Other Wikis allow users to collaboratively model concepts and relationships between them in a formal ontology, along with free-form Wiki content. The RISE project and the subsequent SOP project at Fraunhofer IESE have followed this approach ([1], [2]). The OntoWiki ([3]) and SoftWiki ([4]) projects at the University of Leipzig also support structured information types and browsing and editing of RDF data. Most of these functionalities could, like Wikis, be applied in a wide range of domains. Within Software Engineering, they are well-suited to formal requirements modelling.

Another strategy consists of improving the integration between a traditional Wiki and a specialized Application Lifecycle Management suite or individual applications. Examples in the commercial mainstream include Atlassian Confluence and Edgewall Trac. However, beyond common user administration, this integration remains quite shallow. Referring to work items from a Wiki page, for example, involves either memorizing item numbers or going through a quite tedious search dialogue. In actual use, the Wiki therefore often remains a separate application.

The WAVES\(^3\) project, a joint project by FZI and 7 other partners, strives to improve the support for distributed knowledge management in Software Engineering, and specifically the integration of Wikis with other information sources in- and outside the enterprise, where the Wiki plays the role of a kind of “glue code” between the more structured applications.

This paper focuses on Wiquila, the Rich Wiki Client developed as part of this project, and its integration facilities.

2. CHALLENGES OF USING TODAY’S WIKIS IN AN ENTERPRISE CONTEXT
A survey with our partners, some of whom have had extensive, enterprise-wide Wiki installations for years, as well as our own experience indicate that enterprises struggle with the following issues when using today’s Wikis:

Usability. The appearance of text completely changes between viewing and editing. Learning Wikisyntax is a slow process. No direct feedback is given, making errors more likely.

Productivity. Even those users who know Wikisyntax perfectly will lose time because common operations are more complicated than they need be (e.g. find&replace), and their attention is drawn away from the content itself to technicalities. This may be acceptable for a volunteer project like Wikipedia, where these difficulties may provide an element of identification of its members with “Wiki culture”, but it must be a real concern for enterprises whose goal is not the Wiki itself.

\(^1\) [http://www.polarion.com](http://www.polarion.com); [http://www.intland.com](http://www.intland.com)

\(^2\) [http://snipsnap.org](http://snipsnap.org)

\(^3\) „Wissensaustausch bei der vernetzten Entwicklung von Software" = Knowledge Exchange in Distributed Software Development. Funded with support from the German Federal Ministry for Science and Education (BMBF). Project home page: [http://waves.fzi.de](http://waves.fzi.de)
Rampant growth and other quality issues. Large Wikis can quickly become poorly structured, if there are not volunteer “Wiki gardeners” who engage in cleaning up and direct communication with users. It is quite difficult to see the real structure of a Wiki from a distance, and to get an overview of all the available content, and the part that is still applicable, as a new user.

Integration. A Wiki today is, in a way, a silo of its own, whereas many large enterprises currently seek to reduce the number of applications and increase their integration. The only integration Wikis provide is being able to insert hyperlinks to other resources in the Enterprise which can be accessed via a browser, but even this process is fraught with difficulties, since URLs may contain session IDs. It can be often seen that information that is already available in other sources is manually replicated in the Wiki (in our specific case: phone numbers of employees).

Crossing company borders. It is very hard to find satisfactory solutions when companies want to collaborate for some time. If only one partner hosts the solution, the other partners may be reluctant to contribute because they may not be able to access their contributions later. Single sign-on for several partners is hard to realize.

Offline access. Sales and management spend a lot of time on the road, and therefore need offline access and offline authoring capabilities, otherwise they will always stick with e-mail.

3. GOALS FOR WIQUILA’S DESIGN
Early in the project, we found it legitimate to look beyond pure browser-based interfaces, towards a more comfortable client. First, most other groupware (e-mail, IM) and document creation applications are installed on Desktops. Second, several Rich Internet Client technologies now exist which combine the Web’s ease of deployment with the Desktops richness of interaction.

The design goals for the Wiquila client were:
- Easy editing of formatted text
- Easy (assisted) insertion of links to Wiki pages and existing information objects
- High productivity for power users – no need to switch from the keyboard to the mouse for common actions
- Desktop integration (Drag&Drop, Copy&Paste)
- Built-in tools for quality analysis and improvement
- Extensibility to account for today’s heterogeneous environment
- Ability to be integrated as a component into Eclipse, and perhaps the Web browser
- Support for cross-company scenarios
- Possibility to access “legacy” wikis
- Scalability and performance. Lower latency than in today’s mainstream Wikis for common actions.
- Security
- Protocols can be accessed from similar clients which are based on different technology (.NET, AJAX)
- Robustness – the client works even if some of the integrated backend systems are not running

One of several driving use cases was the Meeting Minutes use case: A project team meets, at an early stage in the project, and meeting minutes are entered in the Wiki. These minutes contain some free-form text as well as “items”: Todos, Dates, Requirements, etc. We want to make it as easy as possible to create these structured items while entering the minutes, and also allow post-processing of minutes that where entered as a draft, perhaps offline. As the project matures, one may want to add additional structure and links to the minutes, to improve traceability of decisions.

4. ARCHITECTURE AND CURRENT IMPLEMENTATION
The remainder of this paper, as well as our current implementation, focus on the information integration and “legacy Wiki” goals. Work on cross-company scenarios and other features is currently in progress.

4.1 User Experience
At first sight, the Wiquila client resembles the familiar “multi-tabbed” Web browser interface. At initial start-up, the user configures at least one Wiki server and a start page. After a Wiki page gets loaded, it is in VIEW (read-only) mode, and hyperlinks can be activated like in a normal Web browser. Pages can also be opened via the “Goto” field in the toolbar. It is enough to type a few letters of the beginning of the page title for a list of suggestions to pop up. The toolbar also has a “Search” field for full-text search of the pages.

4.1.1 Basic Editing
The user can switch to EDIT (read/write) mode by double-clicking into the text. The text can then be edited in WYSIWYG word processor style, using the toolbar, keyboard shortcuts, etc. A few well-established Wikisyntax patterns are recognized as well, e.g. typing an asterisk (“*”) at the beginning of a line will switch to a bulleted list style. Formatted text can be copied to and from other Office applications. However, only a relatively restricted set of “logical” text styles is available in Wiquila. “Physical” attributes (like font family and size) are ignored.

4.1.2 Inserting References to Existing Resources
Wiki pages typically consist of static, formatted and structured text (including static tables), interspersed with internal and external hyperlinks, and dynamic parts (often called variables or templates) that are expanded by the server before being sent to the client. In Wiquila, we have the general concept of a reference to a resource (page or other), which may be a simple static link, but can also dynamically represent some aspects of the resource, provide a mouse-over preview, etc.

References to existing pages or other resources can be inserted in a variety of ways: First, references will occasionally be suggested automatically while the user types, based on the current context of the text cursor. In the example shown below, based on the current “near text” of the cursor (the current implementation uses the last three words), two intra-wiki links, one reference to a Java class, and to one work item are suggested.
Automatic suggestion will only occur if there is a close match between the text entered and the items found, and the number of items is not too large (i.e. the context is specific enough). Also, automatic suggestion will typically not search all registered information sources, but only the most relevant ones.

A list of suggestions can be explicitly summoned by activating the link command (Ctrl-L), followed by a single lowercase keystroke indicating the type of object that the user wants to refer to. E.g. “w” is pressed in order to link to an existing Wiki page, “i” for an image, etc.

As in any Wiki, one can also insert a reference to a new page or resource. To insert a reference to a new Wiki page, the user invokes the link command, then presses capital W, then enters the name of the new page. Depending on the backend, this may trigger the immediate creation of a page stub. References to new resources (other than pages) are created in a similar way through a call to the appropriate methods to create, load, and save Wiki pages and other resources.

4.2 Client platform
A number of alternative client platforms, among them Eclipse RCP, Adobe Flex, AJAX and Mozilla XUL, were evaluated according to criteria such as the availability of an extensible rich text editor, platform independence, a zero-install option, and the option of persisting data locally. Swing’s text framework delivers an HTML editor which supports HTML 3.2 and – partially – style sheets, and, although significantly less robust and finished than other parts of the JDK, is very extensible and well-modularized.

The basic client can be used as a complete standalone Java application (in the Wiquila Sunrise shell which contains the menus and the toolbar), or parts of it integrated in other applications, the key component being the WikiEditorPane which is responsible for representing a single Wiki page.

4.3 Persistence
The Wiquila Persistence extension point provides the client with methods to create, load, and save Wiki pages and other resources that can be uploaded or edited. The content of Wiki pages is represented as feature-reduced HTML. Internal references and backlinks are represented in a collection of metadata objects. Dynamic parts are represented as resources of their own and inlined by the client. These features are key for enabling high-performance servers: In classic Wikis, pages are persisted as Wikisyntax and servers usually convert this syntax to HTML each time a page is viewed. Some servers cache the HTML, but then the cached copy needs to be invalidated each time relevant metadata of linked pages, or dynamic content changes. With our architecture, it is possible to implement servers that never need to parse the static content.

Our current default persistence module uses Subversion and the Subversion Web protocol to store pages. When using this module, the user can access any previously accessed pages in offline mode, thanks to the local “working copy” of Subversion. Implementers who want to access “legacy” Wiki servers through the extension point need to map the standard operations and page lifecycle states of Wiquila to those of their backend, and to convert between the HTML used by Wiquila and the backend’s syntax. This approach has already been proven viable at least for core Wiki features by the WikiPipes project.4

How do we deal with specific features of the backend? Clearly it is impossible for the client to support out-of-the-box the feature superset of all existing Wiki engines. There are currently two approaches, and the persistence extension has some responsibility for both of them. The easy way is to translate engine-specific parts into their static, read-only representation, and mark them as a well-identified, erasable but not editable span of text which can be displayed in the client. In order to edit these parts, one has to switch back to Wikisyntax. The other approach is to represent the...
specific part as a referenced resource (see below), although this may not be viable in all cases.

How do we deal with features that the client supports, but the backend does not? Certain features are optional, e.g. the ability to deliver a collection of backlinks with each page. The extension provides a capability manifest which describes the supported features.

4.3.2 Full-Text Search
Implementers of a persistence extension should also provide an implementation of the full-text search extension point. The extension accepts a search string and delivers a collection of hits. A hit contains page metadata (title, author, etc.), a score, and relevant text fragments. Additional extension modules may be registered for searching non-wiki-page resources (such as tracker items in the example above). The interface closely follows Apache Lucene’s concepts and naming.

4.3.3 Reference Suggestion (during typing)
Reference suggestion seems to be very similar to full-text search at first, but is in fact much more complex. First of all, it is clear that a brute force approach, where a new server-side search is issued for every character entered, cannot work. Issuing one or two search queries per word entered can be considered acceptable; according to [6], a typical typing speed when composing text is about 20 words per minute. The query rate is kept at an acceptable level using several techniques:

- Keyboard events which immediately follow each other (within about half a second) are consolidated into a single “suggester context” event. (The user therefore has to pause briefly before suggestions are calculated.) This is built into the base client.
- A suggester context event is only created when the currently written word has a minimum number of letters (usually 4, less when a left bracket is used). This is also built in.
- Using a client-side database of common word prefixes in normal text, certain suggester context events can be filtered away as to avoid a lot of irrelevant suggestions. This also avoids Wiki page titles containing common words to be constantly suggested out of context. A default filter is available, but it can be replaced or augmented by a specific filter based on data in the backend. These filters are bypassed when suggestions are explicitly requested by the user.
- Suggesters are expected to cache and reuse recent results. For example, when a list of suggestions has been retrieved, the list can usually be filtered locally when the user enters another character.

Figure 3: Conceptual architecture of reference suggestion
In order to allow for several suggesters (one for each information source), each at its own speed, the overall reference suggestion framework follows a modified pipes&filters architecture. At the source, suggester context events are created. An event contains the word at, and the words just before and after the current text cursor, as well as existing references in the current section. The context event then has to pass through one or several filters, as described. It then gets distributed to all registered suggesters. Each suggester, usually an independent thread, will return a list of suggestions or an empty result after some time, during which it might access a remote server. It may also produce some suggestions right away and more suggestions later. Suggestions may be of different classes. The typical suggestion contains a hyperlink, a link title, and the words in the current that are to be replaced. The suggestions from all suggesters are then collected, an overall ranking determined, and the popup displayed to the user (with progressive updates if some suggesters are much slower than others). When the user selects one of the suggestions, it then gets dispatched to a reference completion action (see below), based on its class.

4.3.4 Reference suggestion (pasted text)
A related feature currently under preparation suggests references in pasted text from other applications. Here, the suggester extension will receive the entire pasted text (in plaintext form) at once, and can supply a list of reference suggestions, along with their respective position in the text. The corresponding words are then highlighted in the interface, and the user can right-click each highlighted position to select one of the suggestions there.

4.3.5 Reference Completion and Maintenance
Suggesters, as described, can produce suggestions of several classes. The built-in class just produces a static hyperlink (static URL and link title). This may be sufficient for many types of resources, but it is already not enough for internal Wiki links, where the link should indicate the current state of existence of the destination (normal, stub, or non-existent), and where, as an option, the displayed link title should be updated when the name of the destination page changes. Even more maintenance is needed for general references to resources: References to tracker items should display the item status (open, closed, ...); meeting dates need to be updated when the meeting is rescheduled.

Therefore, for each suggester class, an extension must be registered which is responsible both for reference completion and maintenance. The maintenance operation receives the previously stored HTML of the reference, queries the original resource, and delivers an updated version which replaces the old one. This update is then saved inside the static HTML whenever the user saves some changes of her own. This copy acts as a sort of backup: When the maintenance operation cannot access the resource, a user will always be able to see the reference as it has previously been updated.

This extension may also provide a configuration dialog that allows choosing the appearance of the reference during the initial completion, or later.

4.3.6 Reference Activation
The built-in behavior when a user clicks on a reference is opening the URL in the default Web browser of the OS. Internal Wiki links are opened in Wiquila, of course. This behavior can be overridden for certain types of references, in order to open the resource in another local application, for example.

4.3.7 Future extensions
It is our goal to provide even better integration with the Desktop in the future. For example, dragging&dropping, or
copying & pasting images, files, or desktop application objects onto the Wiki page should insert a reference to the resource, and upload it as a new resource to the server if necessary.

Special resource viewers and editors could be registered to provide more flexible display and manipulation of references inside the Wiki text (“in-place-editing”), without being limited to the built-in capabilities of the Swing HTML editor.

4.4 Security aspects

4.4.1 Access control
Some Enterprise Wikis (e.g. TWiki, Confluence) implement powerful access control schemes where access to certain spaces or pages can be granted or denied via access control lists (ACLs). Wiquila supports creating authenticated connections as well as a linear collection of namespaces under a connection, but does not currently support displaying or modifying ACLs. In our experience with Enterprise Wikis, the context information used to retrieve suggestions exposes the Wiki text as the user writes it, much like a flashlight. The resources suggested by the internal server may themselves lay outside the company network.

4.4.2 Reference suggestions and confidentiality
It is recommended to only access servers in the company network for purposes of reference suggestion. First, because the network traffic is considerable and latency is critical. Second, specifically for internal Enterprise Wikis, the context information used to retrieve suggestions exposes the Wiki text as the user writes it, much like a flashlight. The resources suggested by the internal server may themselves lay outside the company network.

4.4.3 Malicious extensions
Extension mechanisms for Web browsers such as ActiveX, or even Dynamic HTML used to have a bad reputation as enabling malicious code to be executed on the client. To avoid this to happen, the described extension interfaces are designed so that in order to function they most likely do not need access to the objects representing the application as such. Instead, they are passed well-defined input, and produce some output the application then processes. They can keep their own state, persist data in a well-defined local directory, and access the network. The application itself will in the future be shielded from access by the Java security framework. In the Enterprise, the extensions for all Wiquila clients and their privileges are to be centrally managed.

4.5 The WAVES server
In larger organizations, and when there are many different information sources, the approach shown so far may rely too much on the client. Too many individual search requests are generated, individual authentication to each source must be managed, reaction times are not reliable enough. Another outcome of the WAVES project is therefore the WAVES server, which is an integrated, centralized knowledge base, combining the functionalities of a content repository (in our case Subversion), a metadata store (Sesame), and fulltext search (Lucene). All relevant information sources in an enterprise are indexed and their metadata extracted (except for volatile data). This enables search requests that summarize results from several sources in one go, and even combine information from several sources in an individual hit.

5. RELATED WORK

Beyond the other strategies, mentioned in the introduction, for integrating Wikis and Software project management tools, Wiquila borrows on concepts that had been developed even before the Internet era: For example, the idea of an integrated Hypertext browser and authoring tool is as old as Hypertext itself. Many pre-WWW Hypertext clients (e.g. Storyboard) included editing capabilities, although often based on markup. The first Web browser by Berners-Lee and others supported WYSIWYG editing and spellchecking.[5] The W3C’s Amaya continued this legacy somewhat. However, these tools did not suggest hyperlinks automatically.

In comparison, Wikis can be seen as a clumsy, but so far quite effective workaround to reintroduce read/write abilities for the mainstream read-only browsers.

The idea of combining static text with dynamic objects whose state is stored elsewhere, and which can be activated and edited in place was pioneered on the Desktop by Microsoft at the beginning of the 1990s with Object Linking and Embedding (OLE). In the late 1990s, Web Browsers introduced extension technologies such as ActiveX and Java Applets, which could display special objects, and access the network. Around 2001, Microsoft presented the Smart Tags technology for recognizing typed information (e.g. dates, person names) from plain text and suggesting a variety of actions, e.g. hyperlinks to outside resources. However, this would also happen whenever users read a document, not so much as an authoring technique. It was therefore criticized as being potentially a sort of advertising technology.

6. CONCLUSION

The Wiquila Rich Wiki client supports informal knowledge articulation in Enterprise contexts where a lot of structured information is already available in various sources, and offers usability and productivity on a new level.

Wiquila is currently being evaluated by the industry partners of the WAVES project, all software companies with extensive Wiki experience.

7. REFERENCES

supported by semantic wikis. In: Proceedings of the Workshop on Semantic Web Enabled Software Engineering (SWESE), ISWC 2005


