Ajax を用いた Web コミュニケーションツール

稲津 和磨
電気通信大学 大学院 情報工学専攻

Design and Implementation of Web Comunication Tool with Ajax

Kazuma Inazu
Department of Computer Science, The University of Electro-Communications
ina@ipl.cs.uec.ac.jp

概要

インターネットが急速に普及し多くの人が Web を閲覧できるようになり、特に最近は閲覧者が主体となるようなコンテンツに注目が集まっている。そういった中で本研究では Wiki に注目した。Wiki は従来の Web ページとは異なり、閲覧者がページの内容を書き換えることができる。編集する際には HTML のような難しい文字装飾用の言語を用いる必要は無く、簡易的な記法を用いることで文字の装飾を行うことができる。Wiki は画期的で便利なものではあるが、従来の単純な Web ページに比べ、待ち時間が長くなる、サーバへの負荷が増大する、複数人で編集する際にコミュニケーションをとる手段が提供されていないなどの問題点がある。そこで、本研究では Wiki の機能の大半をクライアント側で処理するシステム Karuki を提案する。

1 はじめに

インターネットが急速に普及し、多くの人が Web を閲覧できるようになってきている。得に最近は閲覧者が主体となるようなコンテンツに注目が集まっている。登録をするだけで、すぐに誰でも日記などの文章を公開できるブログの普及に始まり mixi[1] などの SNS の流行には目を見張るものがある。

そういった中で注目したのは、Wiki[2] という、誰でも編集できる Web ページを運用するためのシステムである。

Wiki は画期的で便利なものではあるが、現状において広く用いられている実装には、従来の単純な Web ページに比べ、待ち時間が長くなる、サーバへの負荷が増大する、複数人で編集する際にコミュニケーションをとる手段が提供されていないなどの問題点がある。

本研究では、上で述べたような既存の Wiki システムの問題点を改善した新しい Web コミュニケーションツール Karuki を設計し実装を行った。

本稿では、まず第 2 章で今回の実装で利用する技術を紹介し、第 3 章で従来の実装の問題点を述べる。そして第 4 章で提案機構の設計を説明し、第 5 章で実装方法や各機能の詳しい説明を述べる。第 6 章でサーバへの負荷を測定することにより提案機構の評価を行う。最後に本研究をまとめ、今後の展望について述べる。

2 関連技術

2.1 Wiki

Wiki とは Web ブラウザから閲覧や編集ができる Web ページを運営するシステムである。Web ブラウザのみで編集ができるため、ネットワーク上のどこからでも、いつでも、手軽にコンテンツを書き換えて保存することができる。編集時には HTML よりも簡潔な文法を用いて編集できるため、編集する人に HTML の知識は必要ない。これにより、編集の敷居が低くなり、より多くの人がコンテンツを編集することができる。

こういった特徴を生かして、世界中の人々が百科事典を作ろうというプロジェクトである Wikipedia[3] や、大学の研究室のページ、さらにオープンソースプロジェクトのドキュメントやヘルプ等に広く利用されている。

広く使われている実装としては MediaWiki[4]、PukiWiki[5]、FSWiki[6] 等が挙げられる。

2.2 Ajax

Ajax(Asynchronous JavaScript and XML) [7] とは、Web ページに記述した JavaSctipt プログラムによって、ページの遷移を伴わずにサーバとの通信を行う技術である。Google Maps [8] の基盤技術として、広く知られるようになり、近年、様々な Web アプリケーションに利用されている。

従来の Web アプリケーションは、ページ遷移時のみサーバと通信を行っていたため、ユーザによる入力をサーバに送信しその結果を得るためには、ページ全体を読み直す必要があった。また、同期型の通信であるために、ページを読み直している間、ブラウザは真っ白なページを表示しユーザの処理は受け付けない。これによりユーザは事ある毎にサーバからの返答を待つ必要があった。また、サーバからの返答に時間がかかると、ブラウザがフリーズしたように見えてしまうという問題点もあった。しかし Ajax を用いることで、ページ遷移を伴わずにユーザの入力したデータをサーバに送信したり、サーバからのデータを受信できるようになり、高い応答性の Web アプリケーションが作成可能となった。Ajax は非同期型の通信を行うため、通信中もブラウザ処理を受け付けることができる。

また、Ajax を用いた通信で得られた結果をページに反映するには、Web ページに記述した JavaScript プログラムによって、ページの一部を動的に書き換える。これは、ページ全てを読み込みなおすという従来の方式に比べ高速であり、転送量が減るという利点もある。

3 従来 Wiki システムの問題点

3.1 同期型の通信

上で述べたように、クライアントがサーバに処理を要求する際に同期型の通信を行う従来の方式では、クライアントはサーバの処理が終了するまで待つため、その間は他の処理は何もできない。そのためサーバ側の処理が滞った場合、クライアントは何も操作できず、ただ待つだけになってしまい、ユーザがフラストレーションを引き起こす原因となっている。

3.2 サーバへの負荷集中

従来の Wiki システムの簡単な動作は以下のようになる。

読み込み (図 1)

1. クライアントがページを要求する
2. 要求を受けてサーバは Wiki プログラムを起動する
3. Wiki プログラムは、ページをファイルから読み出し、Wiki の文法を解釈し HTML に変換した後、レイアウトなどを整える
4. サーバは、クライアントにページを返信する

書き込み (図 2)

1. クライアントがページの書き込みを要求する
2. 要求を受けてサーバは Wiki プログラムを起動する
3. Wiki プログラムは、受け取ったデータをファイルに書き込む
4. Wiki プログラムは、受け取った Wiki の文法データを解釈し HTML に変換した後、レイアウトなどを整える
5. サーバはクライアントにページを返信する

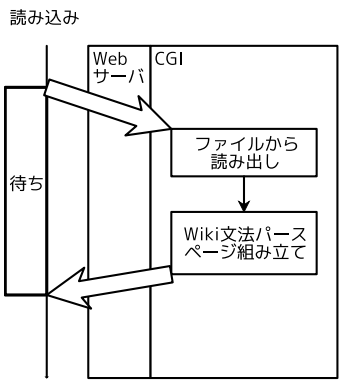


図 1 読み込み処理

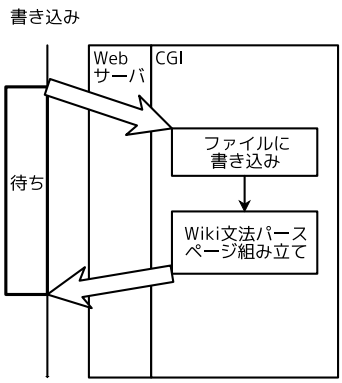


図 2 書き込み処理

この方法ではサーバの行う処理が多く、多数の閲覧要求が来た場合、サーバに負荷がかかりサーバの返答が遅れたり、最悪の場合は返答できなくなることもある。返答が遅れると前述したように、クライアントの待ち時間が増えるためユーザが不愉快な思いをすることになる。

また、従来の実装のほとんどは、閲覧画面から編集画面への移行などの際に、サーバにデータを要求する設計になっている。このような処理によるサーバへの負荷集中や通信時間も、クライアントの待ち時間が増える原因となっている。

3.3 インタラクティブ性の欠如

Wiki は閲覧する人が誰でも編集できるが、結局のところページ編集作業は個人単位であり、真の意味での共同編集には成り得ていない。共同編集をより意味のあるものとするためには、複数の人で相談しながら編集を行ったり、意見を募るような機能が必要である。従来の Wiki では、そのようなインターフェースが十分に用意されていない。

4 設計

第 3 章で述べた問題点を改善するために、以下のような特徴を持つ Wiki システムを設計・実装した。

一般的な Wiki としての機能の提供

従来の Wiki が標準的に提供する機能を実装する。具体的には、閲覧者によるページの自由な書き換え、簡単な文字装飾、ページ検索、ページ間リンク、ファイルアップロードを可能にする。

非同期通信によるサーバ負荷の隠蔽

サーバの返答を待つ間クライアントが停止させないために非同期に通信を行う。これと次の項目と連携する事で、従来の Wiki の問題点を改善する。

サーバ負荷のクライアントへの分散

従来の実装ではサーバが行っていた Wiki 文法の解釈やページレイアウトの構成などを、クライアント側の JavaScript で処理させることで、サーバにかかる負担を減らす。さらに、前の項目の非同期通信と組み合わせることで、読み込み、書き込み時の待ち時間を短縮・隠蔽する。(図 3、図 4)

サーバ側の処理は、ファイルの読み出し、保存、ロックの管理などである。サーバにおける処理を極力少なくするために、クライアントに処理を移行し、サーバ側でしか処理する事のできない入出力関係の処理だけを実装した。このように処理を分けることで JavaScript を用いた Web アプリケーションの複雑さを軽減した。

インタラクティブコンテンツの提供

ページの編集者や閲覧者のコミュニケーションを円滑にし、真の意味での共同編集を可能とするために、チャット機能を実装する。また意見を募る場としてダイアグラムページを実装する。これは付箋のようなものを自由に貼り付けられる掲示板のようなシステムである。

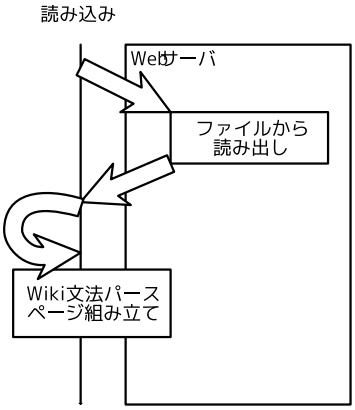


図 3 Karuki の読み込み処理

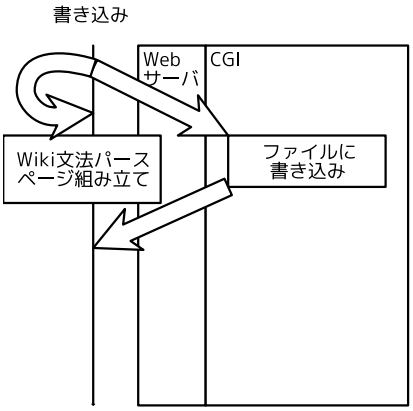


図 4 Karuki の書き込み処理

5　実装

提案機構について、実装の概観とそれらを構成する Wiki、ダイアグラム、チャットの実装の詳細について述べる。

5.1　概観

提案機構の概観を図 5 に示す。

クライアント側で実行されるプログラムは、画面の遷移や、Wiki 文法からブラウザで表示できる HTML への変換、サーバへのデータ要求などを行う。クライアント側のプログラムは JavaScript で記述した。またページのデザインは CSS を用いて行った。これにより、CSS に関する知識があれば Wiki ページのデザインを簡単に変更することができる。

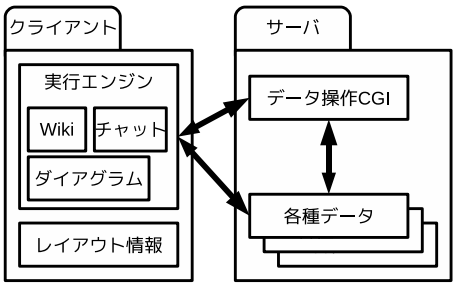


図 5　システム概観

サーバ側で実行されるプログラムはファイル入出力に関する基本的な命令を提供している。サーバ側のプログラムは Python で記述した。

Wiki 文章の読み込みは、CGI を介さず直接 Wiki 文法で記述されたテキストファイルをダウンロードすることにより行う。これにより、サーバの負荷を下げ、ページの閲覧の高速化を図っている。

5.2　システム概観

提案機構の動作画面を図 6 に示す。左側のペインはメニュー、チャット、ページ一覧などを表示する領域で、上部のメニューを選択する事により切り替えを行う。右側のペインは選択したページを表示する領域で、上部のメニューからデータの保存や、履歴の閲覧などを行う。

5.3　Wiki

Wiki 機能は従来の一般的な Wiki の標準的な機能に倣って実装した。ページの閲覧、簡単な文字装飾ルールを用いた編集、ファイルのアップロードなどを備えている。

5.3.1　ロックファイル

Wiki データは同時に 1 人だけが編集できる。そのため排他性を確保するために各ページがロックファイルを保持している。

最終的なロックの判断はサーバ側で行っているので、通信によるタイムラグなどにより、複数人のユー

図 6　動作画面

ザが同一のドキュメントを編集しようとした場合には、一方にロックの取得エラーを返すことで衝突を防ぐ。ロックがかかっている場合はロックファイルを調べることで誰が編集中かを知ることができる。

5.3.2 閲覧時の動作

閲覧時の動作を図 7 に示す。新しいページが読み込まれるとき、クライアントはサーバに新しいページのデータを要求する。ページデータがテキストデータとして返されるのでクライアント側でそれを解釈し、ページを組み立てる。これにより、転送量を抑え、従来の Wiki ではサーバが行っていた Wiki の文法解釈をクライアントが行うことでサーバに集中していた負荷を軽減している。

閲覧中は一定の間隔でサーバにデータを要求し、ページの更新があれば画面を再構築する。こうすることで多少はタイムラグが生じるものの、ユーザは常にほぼ最新のデータを閲覧することができる。

非同期通信を用いているため、各ページの遷移は動的なページの書き換えとなっている。そのため、ブラウザの「戻る」ボタンが使えない、アドレスが変化しないのでブックマークができないという問題があったが、本機構では「#」以下にページを表す名前を付加することでブックマークや履歴に残るようにした。

5.3.3 編集時の動作

編集時の動作を図 8 に示す。編集開始時はクライアントがサーバにファイルのロックを要求する。ロックは固有の ID で管理され、クライアントは同じ ID を用いて再ロックや、ロックの解除を行えるようになっている。ロックが正しく獲得できると編集モードに移行する。

編集中は一定の間隔でユーザの入力した文を解析しプレビューを表示する。これはクライアント側の JavaScript プログラムのみによって行われるものであり、サーバへ負荷をかけることはない。またロックにはタイムアウトが設けてあるため定期的にロックの更新を行う。

編集終了時はサーバにロックの解除を伝える。保存するときは書き込んだ内容をサーバに送信し、サーバ側では CGI プログラムがそれをファイルに書き込む。その際対応するロック ID でページがロックされている場合のみ書き込みが成功する。未知のデータの場合は新たにファイルを作成する。

5.3.4 オートプレビュー

オートプレビュー時の提案機構の画面を図 9 に示す。

クライアントが Wiki 文法を解釈し、HTML に変換するという仕組みにしたことで、サーバに負荷をかけないオートプレビューを実現することができるようになった。この機能により Wiki データがどのように表示されるかを編集中に知ることができ、Wiki の編集をより強力に支援することができた。

5.3.5 解釈できる Wiki 文法

現在解釈できる Wiki 文法は表 1 に示したような独自のものである。しかし、データの保存と Wiki の文法は独立しているため、JavaScript で記述した Wiki 文法変換部分を変更することで、様々な文法に対応することが可能となっている。

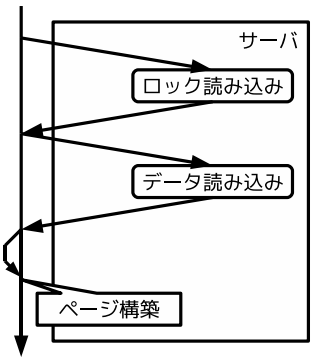


図 7　閲覧時の動作

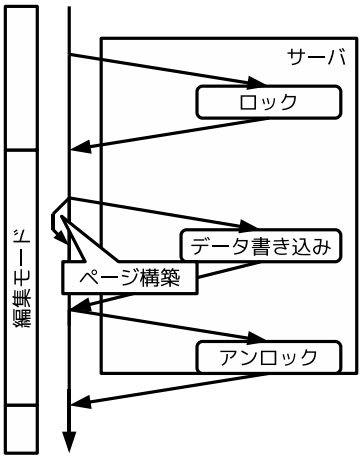


図 8　編集時の動作

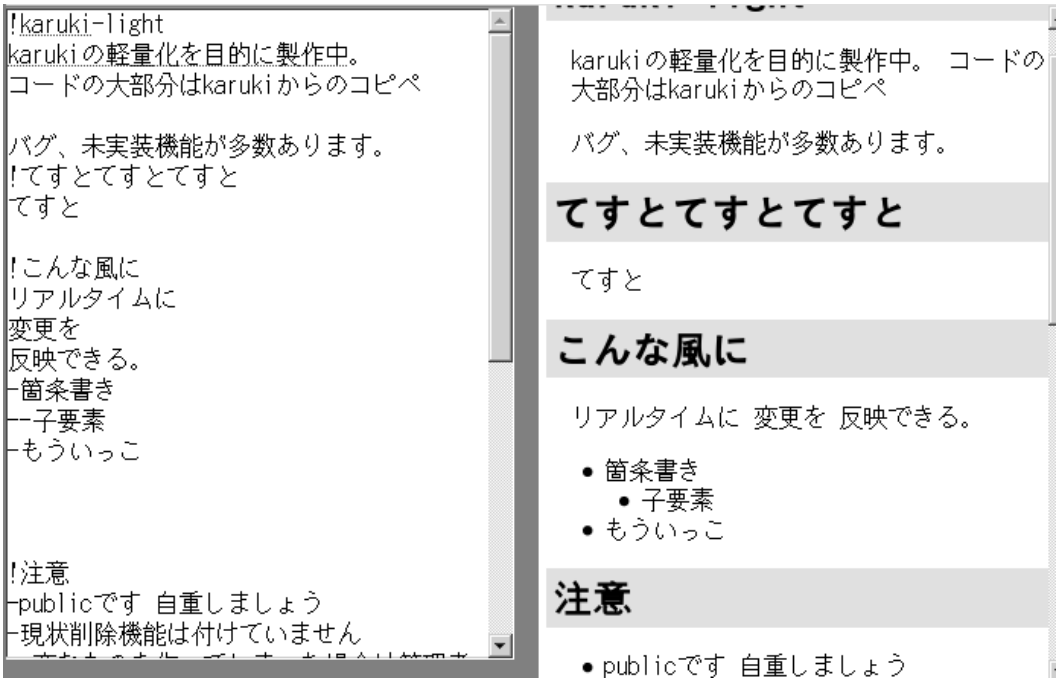


図 9　オートプレビュー

5.3.6　履歴管理

サーバでは変更があったデータの履歴を管理している。これを読み込むことで過去の状態に復元することができる。履歴を表示する際にはクライアント側で差分がわかりやすく見えるように加工して表示する。

5.4　ダイアグラム

本機構の特徴的な機能の一つとしてダイアグラム編集機能がある。ダイアグラムとは、図 10 に示したような付箋型の矩形の中に文字列が書かれたものを指す。

本機構ではこのダイアグラムを好きな場所に貼り付けることによりコンテンツを作成する。この機能を利用することで、意見の出し合いやリアルタイムでの議論などを円滑に進めることができる。

ダイアグラムデータは一つの付箋単位での編集になるため一つのページを複数人で編集することができる。編集に衝突があった場合は各付箋の最終更新日時を調べることでそれを検知し、付箋が複製されるように実装した。

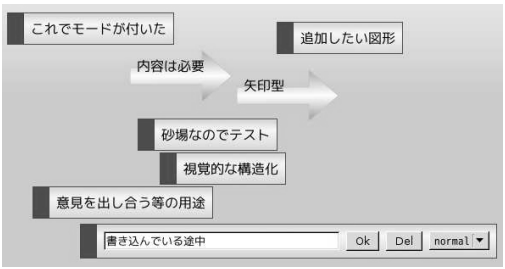


図 10　ダイアグラム

5.4.1　動作の流れ

各付箋は座標と表示する文字列 (ラベル) に加え、固有の ID と最終更新日時を保持している。変更をサーバに伝える際には、座標とラベルに加え ID 情報も一緒に送信する。複数人が編集しても不整合が起きないように同じ付箋への編集が起きた場合、それぞれの編集を反映した付箋を作成する。

5.5　チャット

本機構では閲覧者同士や、閲覧者と編集者との間でコミュニケーションをとる手段としてチャットを提供する。チャットにはどのページからも参照できる Global チャットと、各 Wiki ページ、ダイアグ

表 1　対応している Wiki 文法

行単位の文法

!または*	見出し 1
!!または**	見出し 2
!!!または***	見出し 3
-	箇条書き 1
--	箇条書き 2
---	箇条書き 3
----	水平線
, または —	表 (以降要素を—で区切る)
スペース	整形済みテキスト
:	定義リスト (「:語|定義」という書き方をする)

文字単位の編集

~~	改行
//	コメント
'' ''	強調
''' '''	引用
%% %%	削除
%%% %%%	追加
__ __	下付き
[[ ]]	Wiki 内リンク
http://...	外部リンク
(( ))	脚注

ラムページを閲覧・編集中にのみ参照できる Local チャットがある。

Global チャットには Wiki の文法と同じ表記を使ってリンクを作ることができる。この機能を利用することで、チャットから簡単に特定のページへのリンクを作成することができる。

Wiki ページに併設されている Local チャットには「ここ!」機能を実装した。これは Wiki ページのある部分を指定し、ハイライトする機能である。この機能を利用することで、ページ内の特定の場所を簡単に示すことができる。

5.5.1　動作の流れ

チャットのデータは追記していくものであり、衝突は発生しない。サーバは受け取ったデータを順にファイルに書き込んでいくことでチャットのデータを管理している。

データを見やすい形にレイアウトしたり、リンクをたどったりする動作はすべてクライアント側の JavaScript が処理をするため、サーバはデータを保管しているだけである。

6　既存システムとの比較

6.1　非同期通信を用いた Wiki ライクシステム

Karuki は既存の Wiki[5][6] と比べサーバに負荷をかけない設計になっている。また非同期通信を用いているため仮にサーバの負荷や、ネットワーク遅延でデータが届かなかったとしてもユーザの操作が中断される事は無い。

また JavaScript を用いたオートプレビューは、サーバの負荷軽減と同時にレスポンスの向上という効果ももたらす。更にチャットやダイアグラムといったインターフェースを備えているため、ユーザ同士で円滑なコミュニケーションをとることができる。

現在普及している Wiki システムのほとんどはこのような機能を備えていない。類似のシステムとしては TiddlyWiki[9]、ScrapMemo[10] などが挙げられるが、これらは個人用のメモといった色合いが強く Karuki のように Wiki 上でコミュニケーションをするような用途を前提としていない。

6.2　負荷テスト

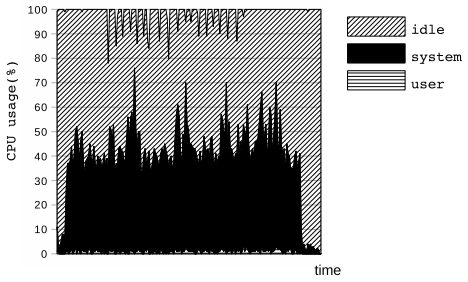
サーバの負荷が軽減ししている様子を測定するために PukiWiki と Karuki の比較を行った。

WebLOAD[11] を用いて初回のページ読み込みを多数のクライアントが行っている状態を 3 分間シミュレートしサーバ CPU 使用率を測定した。結果を図 11(a)、図 11(b) に示す

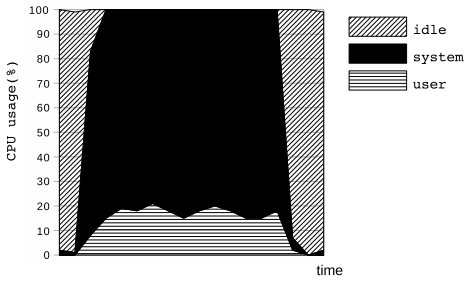
ページの読み込みにおいて、提案機構ではデータを読み出すだけなので、CPU 負荷が少ないことが確認できる。一方、PukiWiki では、データの読み出しから HTML への変換までを行うため CPU 負荷が高くなっている。

7　まとめ

本稿では Ajax を用いて従来の Wiki の問題点を解消したコミュニケーションシステム Karuki を実装し、その設計と実装の詳細を説明した。提案機構を用いることで、Web 上で多人数でコミュニケーションをとりながらデータを編集することができるようになる。今後の課題としては、システムとしての完成度を上げると同時に簡単に新しいデータ構造を追加できる仕組みを導入していこうと考えている。



(a) Karuki



(b) PukiWiki

図 11　データ読み込み時のサーバの負荷

謝辞

本機構は未踏ソフトウェア創造事業部未踏ユースのプロジェクト「Ajax による Web コミュニケーションツール」として開発しました。ご支援いただいております IPA の諸氏、開発にあたり様々な面でご指導をしてくださった、慶應義塾大学の安村通晃教授、株式会社オープンテクノロジーズの皆様に厚くお礼申し上げます。また、本機構を開発するにあたり終始手厚いご指導をしてくださった電気通信大学の岩崎英哉教授に感謝致します。

参考文献

[1] ソーシャル・ネットワーキング サービス mixi . http://mixi.jp/ .

[2] Leuf, Bo. The Wiki Way:Quick Collaboration on the web.

[3] Wikipedia . http://en.wikipedia.org/ .

[4] MediaWiki . http://www.mediawiki.org/ .

[5] PukiWiki . http://pukiwiki.sourceforge.jp/ .

[6] FSWiki . http://fswiki.org/ .

[7] Ajax: A New Approach to Web Applications . http://www.adaptivepath.com/ideas/essays/archives/000385.php .

[8] Google Maps. http://maps.google.com/ .

[9] TiddlyWiki: a reusable non-liner personal web notebook . http://www.tiddlywiki.com/ .

[10] ScrapMemo . http://espion.just-size.jp/files/js/smemo/ .

[11] WebLOAD: Open Soucrce Load Testing. http://www.webload.org/ .