手軽で柔軟な ML アーカイバ msgcab の設計と実装

上野 乃毅　　**田中 哲**
Daiki Ueno　　Akira Tanaka

独立行政法人 産業技術総合研究所 情報技術研究部門
Information Technology Research Institute,
National Institute of Advanced Industrial Science and Technology (AIST)

概要

フリーソフトウェアの開発の場での利用を視野に入れ、導入が簡単でプラグインによる拡張が可能な ML アーカイバ msgcab を開発した。議論のスレッド表示や全文検索、複数の ML を跨いだメールの参照などの機能を標準で提供する。本論文では msgcab の内部構造を解説する。

1 はじめに

1.1 フリーソフトウェアの開発と ML

分散バージョン管理システム (distributed version control system)・問題追跡システム (issue tracking system, ITS)・自動ビルドシステム (continuous integration system) などのツールの充実に伴い、フリーソフトウェアの開発手法は次第に洗練されつつある。その一方で、開発に必要なコミュニケーションの手段は昔も今も電子メールが中心であり、開発上の意思決定は開発者の参加する **ML (mailing list)** で行なわれる。

開発者にとって ML は身近なものであるが、時系列を遡って情報を発見する際に不便なことがある。よくあるのは、大昔の変更に起因する**息の長いバグ**の修正である。バグを正しい形で解決するには変更の背景事情を知らねばならない。当時から ML を購読していれば手元のメールボックスから変更にまつわる議論を発掘できるかもしれないが、そうでなければ別途にメールを入手する必要がある。

1.2 ML アーカイバ

幸いにも、最近では ML に投稿されたメールがウェブに公開されていることが多い。ML のメールを収集・公開するサービスとして、主にフリーソフトウェアに関連する ML を揃えた Gmane[1] や MARC[2]、オブジェクト指向スクリプト言語 Ruby[3] に関連する ML を揃えた blade[4] などがある。これらのサービスを実現するソフトウェアは **ML アーカイバ** と呼ばれる。残念なことに、現在運用中のサービスで使用されている ML アーカイバの多くはソースコードが公開されていないか、導入が困難である。一方で、自由な ML アーカイバは 1990 年代後半から 2000 年代初頭に開発されたものが多く、近年は目立った進歩が見られない。そこで筆者らは新たな ML アーカイバ msgcab (message cabinet) を開発した。msgcab のソースコードは `http://msgcab.nongnu.org` で配布している[*1]。

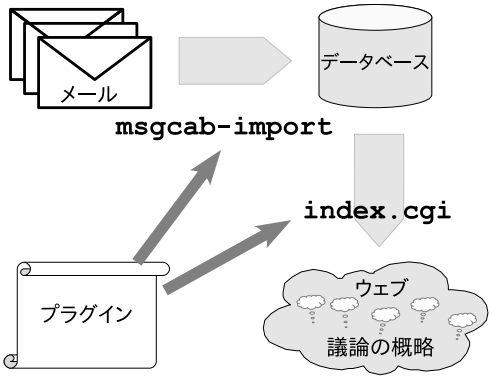
1.3 msgcab の概要

msgcab の動作の概要と導入手順を図 1 に示す。

`msgcab-import` は外部のメールボックスからメールを取り込むスクリプトである。Maildir, MH, mbox などの大抵のメールボックス形式に対応している。また、通常の ML アーカイバは ML 毎に振り分け済みのメールを扱うが、msgcab は振り分けの処理を内部で行なう。

`index.cgi` はウェブのユーザーインターフェイスを提供する CGI スクリプトである。コマンドラ

[*1] 産総研知財管理番号 H16PRO 241




```
$ tar zxf msgcab-X.Y.Z.tar.gz
$ cd msgcab-X.Y.Z
$ ./msgcab-import ~/Maildir
$ ruby index.cgi server 12345
```

図 1　msgcab の動作の概要と導入手順

インから起動すると単独の HTTP サーバとして動作する。この状態で `http://localhost:12345` を開いたのが図 2 である。

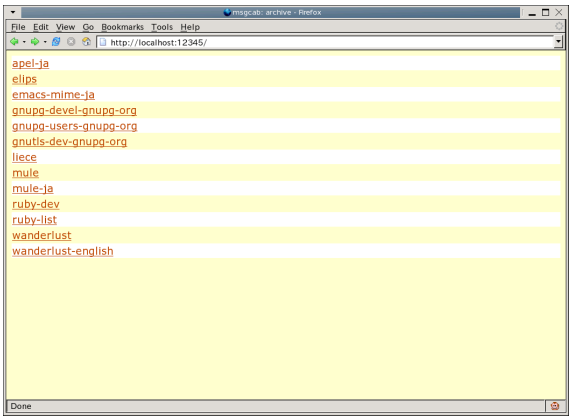


図 2　msgcab の初期ページ

メールボックスの内容が 13 個の ML に振り分けられたと分かる。ML 名のリンクを辿ると図 3 に遷移する。このページでは上半分に最近の議論での話題がまとめられ、下半分に番号順に区分された過去のメールへのリンクがある。

図 4 はメールの内容を表示しているページである。引用部分が色分けされ読み易く表示される。また、このページの下方には現在のメールを含むスレッド (3.1) が表示される。

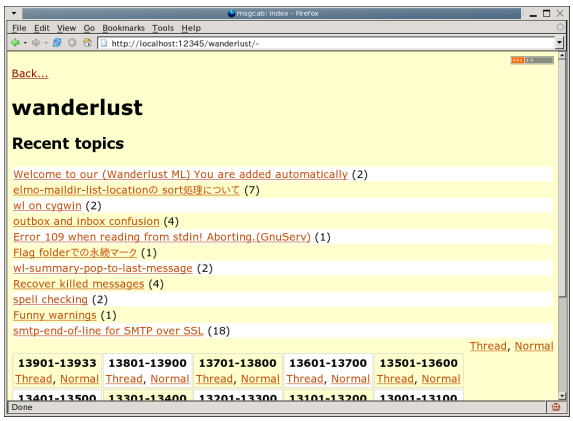


図 3　msgcab の ML のページ

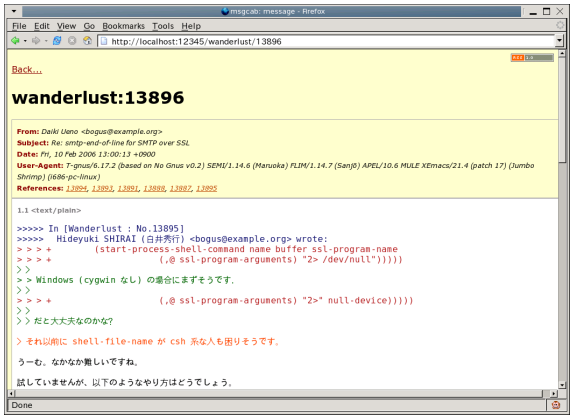


図 4　msgcab でのメールの表示

2　関連研究

自由な ML アーカイバには Hypermail[5], MHonArc[6], Pipermail[7], w3ml[8] などがある。

いずれもスレッドの表示に対応しているが、Hypermail, MHonArc, Pipermail はメールを一ヶ月毎や 100 通毎といった単位で分割して整理するためにスレッドが途切れてしまい、議論を追うのが困難なことがある。w3ml ではメールの内容を表示するページの下方にスレッド全体が表示されるので、この問題はない。

Hypermail や MHonArc は開発の歴史が長く、メールの振り分けや全文検索などの進んだ機能を提供しない。このため、現実的には外部のソフトウェアと連携して運用する場面が多く、導入

にシステム管理の知識が必要となる。MHonArc に procmail[9] や Namazu[10] などのソフトウェアを組み合わせて導入の敷居を下げるパッケージ mharc[11] が開発されている。Pipermail は ML 管理ツール Mailman の一部として配布されているため、Mailman と併用する限りにおいては導入の手間はかからないが、全文検索などの機能に乏しい。

3 設計

設計上の要件として特に次の三点を考慮した。

1. 議論の追跡可能性の向上
2. プラグイン機構による拡張可能性
3. スケーラビリティの確保

第一に議論の追跡可能性の向上である。議論の発見にメールの全文検索は有用であるが、得られた検索結果から議論の全貌を把握するのが困難な場合がある。例えば、同一のメールが複数の ML に投稿されると別々の ML で並行に議論が進むことがある。このような場合にも ML 間の垣根を意識することなく議論を把握できると良い。

第二にプラグイン機構による拡張可能性である。プラグイン機構とは、機能の追加や取り外しを msgcab 自体を改変することなく可能にする仕組みである。先進的な機能をプラグインとして実装し、十分なテストが済んでから本体に還元する効果も狙える。1990 年代後半からの迷惑メールの増加に伴い、ML アーカイバには spam フィルタやメールアドレスの隠蔽機能が求められてきたが、従来の ML アーカイバでは機能の追加にあたって本体を改変する必要があった。プラグイン機構があれば、このような未知の問題にも柔軟に対応できる。

第三にスケーラビリティの確保がある。ソフトウェアの設定を変更するだけで少人数での利用から世界規模のサービスまで幅広い運用形態に対応できるのが望ましい。

本節の後半では、これらの要件を満たすための工夫をひとつずつ紹介する。

3.1 スレッドの復元アルゴリズム

スレッドとはメール間の参照の親子関係を木構造で表現したものである。msgcab でスレッドを表示したようすを図 5 に示す。

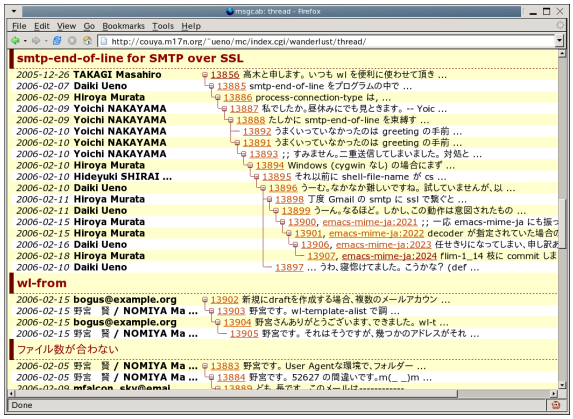


図 5 msgcab のスレッドの表示

メール間の参照に関する情報はメールのヘッダ部にある Message-ID, In-Reply-To, References の各欄から知ることができる。本節では説明を簡単にするためにメールを Message-ID 欄の値で識別する。また、Message-ID 欄の値を記号 A, B, C, ··· で表わす。図 6 では、C が A と B を参照していることが分かる。

```
From: bob@example.com
To: foo-devel@example.com
References: A B
Message-ID: C

ボブです。どうみてもバグです。
本当にありがとうございました。
```

図 6 メールの参照関係の明示

本来は References 欄内の要素の順番にも意味があり、B が A を参照していることが表明されている [12]。しかし、広く普及した MUA (Mail User Agent) の送信部のバグなどから厳密にはこの規則は守られていないことが多い。また、メールの欠落や到着の遅延、メールのヘッダ部の書き換えも生じ得るため、スレッドを意図された形で復元するには

頑健なアルゴリズムが必要となる。

Zawinski の考案したアルゴリズム [13, 14] は頑健で高速ではあるが、元来 MUA のために設計された経緯から ML アーカイバに採用するにはスケーラビリティに問題がある。具体的には、復元されたスレッドのインクリメンタルな更新に対応していないため、10,000 通のメールからスレッドを復元した後に新たなメールが到着した場合、10,001 通の処理を最初からやり直さなければならない。

そこで ML アーカイバに適した新たなアルゴリズムを考案した。このアルゴリズムの特色は References 欄内の要素の順番に依らない点である。直感的には「全てメールの親が高々一つに定まればスレッドが復元できる」という事実と「親の親は自分の親ではない」という事実に基づいている。アルゴリズムの概観は次に述べる通りである。全メールについて Message-ID 欄の値から References 欄内の要素への対応関係をあらかじめ調べておく。

Message-ID 欄の値が m のメールについて References 欄内の全要素の集合を $R(m)$ で表わす。図 6 の例では $R(\underline{\mathtt{C}}) = \{\underline{\mathtt{A}}, \underline{\mathtt{B}}\}$ である。m に対応するメールがまだ届いていない場合には $R(m) = \phi$ とする。M を Message-ID 欄の値の集合とし、R を集合に対しても以下のように拡張して定義する。

$$R(M) = \bigcup_{m \in M} R(m)$$

また、m の全ての祖先の集合 A は R の閉包として次のように定義する。

$$A(m) = \{m\} \cup R(m) \cup R(R(m)) \cdots$$

A についても R と同様に集合に対して拡張する。m の親候補の集合 $P(m)$ は次で求められる。

$$P(m) = R(m) - A(R(R(m)))$$

$P(m)$ の要素数が高々一つに絞り込めれば m の直接の親が発見できたことになる。

新たなメールが届く度に $A(R(R(m)))$ の要素数は単調に増加するので $P(m)$ の要素数は単調に減少する。一回の計算で m の直接の親が発見できなくても、m に対応する $P(m)$ と、$P(m)$ の計算途中に現われた未到着のメールの集合 $N(m)$ を記録しておけば、アルゴリズムをインクリメンタルに実行できる。

3.2 プラグイン機構

3.2.1 プラグインのファイル構成

msgcab はウェブアプリケーションであるが、同時にコマンド行からメールの格納やシステムの管理を行なうインターフェイスを提供する。このような実行方式の差異をフレーバ (flavor) と呼んで区別する。プラグインは一つ以上のフレーバに機能を追加する。

全文検索エンジン HyperEstraier[15] を用いてメールの検索を行なうプラグイン (estraier) を例にとって説明する。このプラグインのファイル構成は次の通りである。

- estraier.webapp.rb
  webapp フレーバでの処理
- estraier.cli.rb
  cli フレーバでの処理
- msgcab-estraier
  索引を手動で更新するためのスクリプト

estraier プラグインをウェブから呼出した場合には webapp フレーバで動作する。また、コマンド行から呼出した場合には cli フレーバで動作することになる。つまり、ウェブから検索を実行する処理は estraier.webapp.rb に、コマンド行からメールを格納する際に全文検索の索引を更新する処理は estraier.cli.rb に記述される。

3.2.2 テンプレートエンジンの利用方法の拡張

msgcab では HTML を生成するテンプレートライブラリに htree[16] を選択した。htree ではテンプレートの記述は HTML そのものである。htree はテンプレートを HTML として解釈したうえで、その構文木を対象に展開処理を行なう。これによりタグのエスケープ漏れによる XSS (Cross Site Scripting) 脆弱性を予防できる。

htree は単一のテンプレートの記述には十分であるが、プラグインが新たなテンプレートを提供

する場合がある。例えば、先に挙げた estraier プラグインはページの上方に検索窓を追加する (図 7)。estraier プラグインが提供するテンプレートを msgcab 本体のテンプレートに埋め込むためにアンカー (anchor) と呼ぶ機構を導入した。

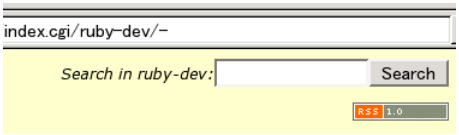


図 7　estraier プラグインでのアンカーの利用

アンカーを使うとテンプレート内で予め印付けられた位置に HTML の要素を埋め込める。図 8 はアンカーを使って拡張可能なテンプレートの例である。`body` 要素の直下に `:header` という名前のアンカーを呼出す式が書かれている。htree ではテンプレートに名前を付け、その場に埋め込む形で展開する機能がある。これをテンプレート関数と呼ぶ。`_call` は htree によって独自に解釈される属性で、`"replace"` という名前のテンプレート関数を呼出している。

```
<html>
  <head>
    <title>msgcab: summary</title>
  </head>
  <body>
    <div class="header">
      <span _call="replace(:header, self)"/>
    </div>
    ...
  </body>
</html>
```

図 8　テンプレート内のアンカーの呼出し

一方の estraier プラグインでは、`:header` に対応する `"replace"` テンプレート関数を定義する (図 9)。アンカーはリストに繋がれており、最後の行の `_call="next_anchor.replace"` では次のアンカーに処理を移している。

3.3　メールの格納方式

msgcab では障害からの復旧を容易にするため、一つのファイルに一つメールを保存する形式を採用した。しかし、単一のディレクトリ内に大量のメールがあるとファイルシステムによっては open(2) の呼出しに時間がかかることがある。従来の ML アーカイバでは ML 毎に別々のディレクトリに分割することができたが、msgcab では議論の追跡可能性を向上する目的で ML 間の垣根を排してしまった。

```
<div _template="replace">
  <form _attr_action='view.uri("/folder")'>
    <span _text='"Search in #folder.name:"/>
    <input type="text" size="16" name="q"/>
    <input type="submit" value="Search"/>
  </form>
  <span _call="next_anchor.replace"></span>
</div>
```

図 9　アンカーの定義

そこで、メールが格納される順に連番の番号を付け、この番号の桁数を元にディレクトリを分割する方式 MailTree を考案した。一階層目のディレクトリはディレクトリの深さを表わしている。具体的には次のようなディレクトリ配置になる。

- 0 .. 99 番目のメール
  `1/{00,...,99}`
- 100 .. 9999 番目のメール
  `2/{00,...,99}/{00,...,99}`
- 10000 .. 999999 番目のメール
  `3/{00,...,99}/{00,...,99}/{00,...,99}`

例ではファイル 100 個単位でディレクトリを分割している。メールの総数を n とすると、ディレクトリの深さは $\lfloor \log_{100}(n) \rfloor + 1$ であり、十分なスケーラビリティを期待できる。

4　実装

msgcab は Ruby で実装した。Ruby を選択した理由には、メールの解析や RDBMS へのアクセスを提供するライブラリが充実していることと、プラグインの書き易さが期待できる点がある。標準で提供するプラグインを表 1 に示す。

導入を容易にするために、特別なインストール手続きや設定ファイルの記述なしに動作するようにした。1.3 で述べた振り分け処理の規則も典型的なも

名称	機能
cite	メールの引用部分の強調表示
estraier	メールの全文検索
face	送り主のアイコンを表示
link	メールの本文中の URL の置換
mask_mail_address	メールアドレスの隠蔽
rss	RSS によるサマリの生成

表 1　標準で提供するプラグイン

のを標準で提供している。

また、スケーラビリティを確保するために Ruby/DBI[17] を用いて外部の RDBMS を交換可能にした。何も設定しなければ、スタンドアロンで RDBMS の機能を提供する SQLite[18] が用いられるが、大規模な運用の際には負荷分散の機能を持つ RDBMS に容易に移行できる。

また、JavaScript を随所に用いることでユーザインタフェイスの使い勝手の向上を図っている。例えば、マウスのクリックによりスレッドの畳み込みができる。

5　評価

評価に使用した環境を表 2 に示す。msgcab は執筆時の最新版である 0.0.2 を用いた。

CPU	Mobile Athlon(tm) 64 3400+
メモリ	2GB
OS	Linux 2.6.8
RDBMS	SQLite 2.8.16
Ruby	1.8.4

表 2　評価に使用した環境

対象のメールは Emacs 上で動くメール/ニュース管理システム Wanderlust[19] の 2 つの ML (wl, wl-en) と Ruby の 3 つの ML (ruby-dev, ruby-list, ruby-talk) の計 5 つの ML から採取した。

最初に、10,000 通のメールを一括で取り込み、表示にかかる時間を調べた。取り込み処理に要した時間は 16 分 12 秒であり、その後に wl ML の最新 200 通のメールからスレッドを表示するのに要した時間は 2.2 秒であった。メールを一括で取り込む場面は主に設置時であるから、現実的には問題にならない速度であると考えられる。表示に関しても問題のない速度であると言えるが、高速化の余地として HTML 出力の段階でのキャッシュが考えられる。

次に、スレッドの復元がインクリメンタルに行なわれたことを示すため、メールを 100 通単位で取り込む実験を行なった。スレッドの復元処理はメールの取り込みの際に行なわれるため、表示に要する時間の計測は割愛した。

所要時間の推移を図 10 に示す。

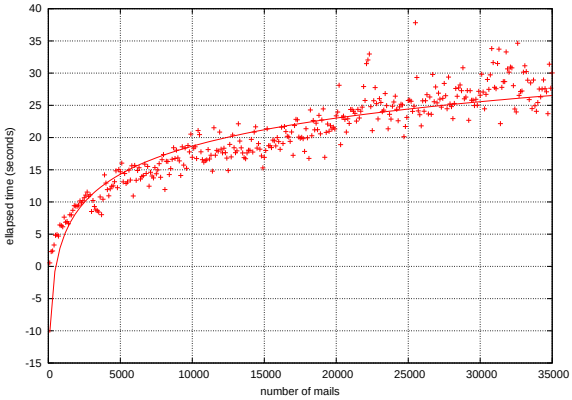


図 10　メールの取り込み処理の所要時間

実際の ML のデータを用いているために全体に揺れが大きいが、メールの増加に伴ない所要時間が緩やかに増加している。これには SQLite の性能特性が影響していると考えられる。SQLite のデータベースはディスク上では B-Tree で管理されているので、セルの挿入にかかる時間は格納済みのセルの総数 n に応じて $O(\log n)$ で増加することが推測できる。 実験結果を $a \log n + C$ で近似すると $a = 6.285, C = -39.22$ となった。

執筆時点で Gmane は 40,000,000 通のメールを蓄積している。仮に同規模に達した場合のメールの取り込みの所要時間を上式から外挿すると、1 通あたり 0.086 秒程度になる。この数値は許容範囲内と考えられる。ただし、現実に Gmane と同規模の運用をするには SQLite のファイルサイズ等の他の制約が問題になる可能性がある。

6 応用

他のウェブアプリケーションとの重ね合わせ (mushup) により有用なサービスを構築できる。例えば、ML での議論が錯綜すると、Wiki に議論の概略をまとめることがあるが、手作業でまとめるには労力が要る。JavaScript の CSI (Client Side Include)[20] の手法を応用し、msgcab から議論のスレッドの一部分を抜き出して Wiki に貼り付けることができればこの作業は簡易になるだろう。

また、筆者らは msgcab と自動テストシステムを組み合わせたバグ追跡システム Couya [21] のプロトタイプを作成した。

7 今後の課題

現在の msgcab は利用者を認証・識別する機能を持たないため、利用者に応じた未読・既読の管理のような、ウェブメールに典型的なユーザインターフェイスが提供できていない。今後は OpenID[22] や Passel[23] に代表されるシングルサインオン認証機構との連携を考えている。

また、デスクトップ向けクライアントの提供も考えている。例えば、Emacs から msgcab にアクセスするクライアントが考えられる。

スレッドの復元を Message-ID 欄と References 欄の値を頼りに行なっているが、メールのヘッダ部の書き換えが起きると意図した形でスレッドを復元できないことが考えられる。メール本文の引用を利用することでより頑健にスレッドを復元できるようになるだろう。

8 まとめ

導入が簡単でプラグインによる拡張が可能な ML アーカイバ msgcab の設計と実装について述べた。議論の追跡可能性の向上・プラグイン機構による拡張可能性・スケーラビリティの確保の 3 つの要件に的を絞り設計されている。msgcab は Ruby で実装され、実用可能な速度で動作する。他のウェブアプリケーションとの重ね合わせにより、さらなる応用が期待できる。

参考文献

[1] L. M. Ingebrigtsen. Gmane – Mail To News And Back Again. http://gmane.org.

[2] MARC: Mailing list ARChives. http://marc.theaimsgroup.com.

[3] Y. Matsumoto. Ruby: Programmers' Best Friend. http://www.ruby-lang.org.

[4] blade. http://blade.nagaokaut.ac.jp.

[5] P. McCluskey. Hypermail project home page. http://www.hypermail-project.org.

[6] E. Hood. MHonArc Home Page. http://www.mhonarc.org.

[7] A. M. Kuchling. Pipermail. http://www.amk.ca/python/unmaintained/pipermail.html.

[8] M. Tomita. w3ml. http://www.tmtm.org/ruby/w3ml/.

[9] S. R. van den Berg and P. A. Guenther. Procmail Homepage. http://www.procmail.org.

[10] Namazu Project. Namazu: a Full-Text Search Engine. http://www.namazu.org.

[11] E. Hood. mharc. http://www.mhonarc.org/mharc/.

[12] P. Resnick. RFC2822: Internet Message Format, April 2001.

[13] J. Zawinski. message threading, 1997. http://www.jwz.org/doc/threading.html.

[14] M. Crispin and K. Murchison. Internet message access protocol - sort and thread extensions. INTERNET-DRAFT, May 2004. http://www.ietf.org/internet-drafts/draft-ietf-imapext-sort-17.txt.

[15] M. Hirabayashi. Hyper Estraier: a full-text search system for communities , 2004. http://hyperestraier.sourceforge.net.

[16] A. Tanaka. htree - HTML/XML tree library. http://cvs.m17n.org/~akr/htree/.

[17] M. Neumann. Ruby/DBI. http://rubyforge.org/projects/ruby-dbi/.

[18] R. Hipp. SQLite. http://www.sqlite.org.

[19] Y. Teranishi. Wanderlust. http://www.gohome.org/wl/.

[20] Z. Xiao. Client-Side Includes: Faster Access to Dynamic Web Content. AT&T Innovation Forum, October 2002.

[21] 上野乃毅, 田中哲. 再現テストを自動的に行なうバグトラッキングシステム. 第 8 回プログラミングおよび応用のシステムに関するワークショップ (SPA2005) 予稿集. 日本ソフトウェア科学会, March 2005.

[22] B. Fitzpatrick. OpenID: an actually distributed identity system. http://www.openid.net.

[23] D. Smith and P. Saint-Andre. Passel Whitepaper: Passel: Identity for the Rest of Us, July 2005. http://www.passel.org/whitepaper.html.