卒業論文

Hikiプラグイン開発環境の構築

関西学院大学 理工学部

情報科学科 西谷研究室

2561 森下 慎也

2016年3月

概 要

本研究では Wiki クローンの 1 つである Hiki の環境構築の自動化及び Hiki プラグイン開発環境の構築を目的とした.

Hiki プラグイン開発を始めるにあたって，Hiki プラグインの動作の不透明さ, 開発をサポートするツールがないという問題にぶつかっていた. 本研究ではひな形を提示する機能や, GUI における編集を可能にするツールを開発する事で前述の問題の解決を目指した.

また, Hiki 環境構築に時間がかかってしまう事が多く, 環境構築だけで時間を取られてしまっていた. そこで本研究では巨大システムの構成管理にも使用される Chef というツールを使用する事で Hiki 環境構築の自動化をおこなった. Chef を用いる事で設定に必要となる多数のコマンドを 1 つのコマンドで実行できる. Hiki 環境構築の自動化を応用し, 本研究では開発の引き継ぎによる継続的なプラグイン開発を可能とする為に, 他者のプラグインをテストする環境を構築する Chef の設定ファイルも合わせて開発した.

目 次

第1章　序論

1.1　Hikiの現状と問題点

1.1.1　Hikiとは

Hikiとは, プログラミング言語Rubyによってコーディングされている Wikiクローンの１つである. HikiとWikiとの違いはコードがRubyで記述されている点, ページの編集がHiki記法によりおこなわれる点が挙げられる. Web上にページを作成することで, 多人数で情報を共有できるため, 多人数での作業における情報共有に適している.

HikiはWikiの Content Management Systemとしての特徴を引き継いでいる. Content Management System(CMS) とは, 画像やテキストを管理し, Web配信に必要な処理をおこなうシステムである. CMSを用いる事で, HTML等の知識の習得無しにWebページの作成をおこなう事ができる.

Hikiによって以下のような機能を提供する事ができる.

- オリジナルWikiに似たシンプルな書式
- プラグインによる機能拡張
- アクセス制限が可能
- 出力するHTMLを柔軟に変更可能

Wikiの特徴にも挙げられるプラグインによる機能拡張ができる特性を用いて, システムのプロトタイプを迅速に作成できる. 本研究ではver.1.0の環境構築について考える [1].

1.1.2　従来のHiki環境構築

西谷研究室では主にRuby 2.0系を使用している為, Ruby 2.0系に合わせたHiki 1.0系を使用する. しかし, デフォルトのHiki 1.0系にはいくつかの不具合があるため, 西谷が

改良をおこなったHiki 1.0系のダウンロードが必要となる．そのため, Hikiをダウンロードする際には, 公開サーバにあるGitリポジトリから, git cloneというコマンドにより自分のサーバへとダウンロードする必要がある．しかし, 研究室に配属されたばかりの学生はCUIの作業に慣れていない事が多く, Git操作のコマンド実行やHiki環境構築する為のディレクトリ構造の作成に時間が取られてしまう．

また, ダウンロードしたHiki内のデータ権限の変更等が必要になる．これはCUIの操作に慣れていない学生にとっては困難な作業となる．それに加えて, ブラウザ上に表示できるようにApacheの設定ファイルを変更する必要がある．エディタの利用に慣れていないとこの作業もかなり時間が取られ, 実際に研究やレポートの作成へ取りかかるまでにかなりの期間を要する事もある．

1.1.3 従来のHikiプラグイン開発

西谷研究室にはHikiプラグイン開発を研究対象とする学生もいるが, Hikiプラグインがどのような動作をおこなって出力が得られるのかを理解するまでに時間がかかってしまう．また, 外部にもHikiプラグインの作成マニュアルのようなものはほとんど無く, コードのひな形も用意されていないため, 開発に入る際にひな形作りから始まる．このため, Hikiプラグイン開発を始めにくいという状況に陥ってしまっている．

プラグイン開発が始まってからは, 図1.1に示すようにターミナル上での編集とブラウザでの動作確認を繰り返す事となり, 画面操作が多くなってしまうため作業効率が悪いと考える．また, Apacheのエラーログを読まなければならない際にHikiのライブラリの文法についての注意文が多数表示されてしまい, 肝心のプラグインが原因となっているエラーを読み取れなくなってしまっている．

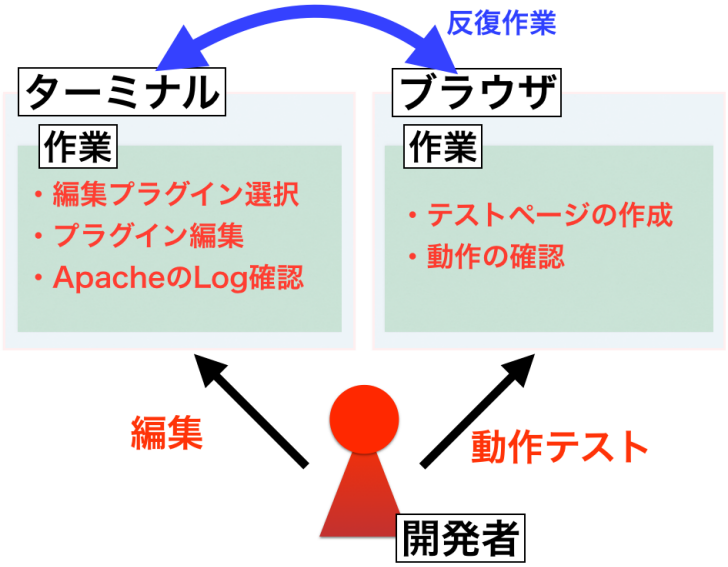


図 1.1: 従来のプラグイン開発の流れ.

1.2 本研究の目的

本研究では, Chef を用いた Hiki 環境構築を自動化する手法の考案及び Hiki プラグイン開発過程の手助けとなるツールの開発を目的とする.

Hiki 環境構築にかかる手間を解消する為に Chef というシステム構成管理ツールを用いる. 本研究では Hiki 環境構築をおこなう Chef の設定ファイルを開発する.Chef はシェルスクリプトに近い動作をおこなうが, 特徴として冪等性という何度実行しても同じ結果が得られるという特性を持っている点, Ruby の内部 DSL であるという事から Ruby の経験者にとって可読性が高いという点が挙げられる. これにより, 作成者と編集者の間の考えの食い違いを避ける事ができ, 運用していくにあたって必要となるコードの改善がおこないやすくなっている.

また, Hiki プラグインの動作は初心者には馴染みの無いものであり, 開発についてのサポートツールの配布もおこなわれていないため, Hiki プラグインによってシステム開発にとりかかるのが困難となっている. また開発が始まってからも画面移動が多い等の問題にぶつかる. これらの問題について, 本研究ではひな形を提示する機能, GUI による編集システムの開発による作業効率の向上を目指す.

第2章　開発要件と実装法

2.1　プラグイン開発のサポート機能

図 2.1 に示すような, プラグイン編集時には GUI のみを使用し, CUI では編集内容のリセットやプラグイン共有等のコマンドを実行する, という開発環境の構築を目指す.

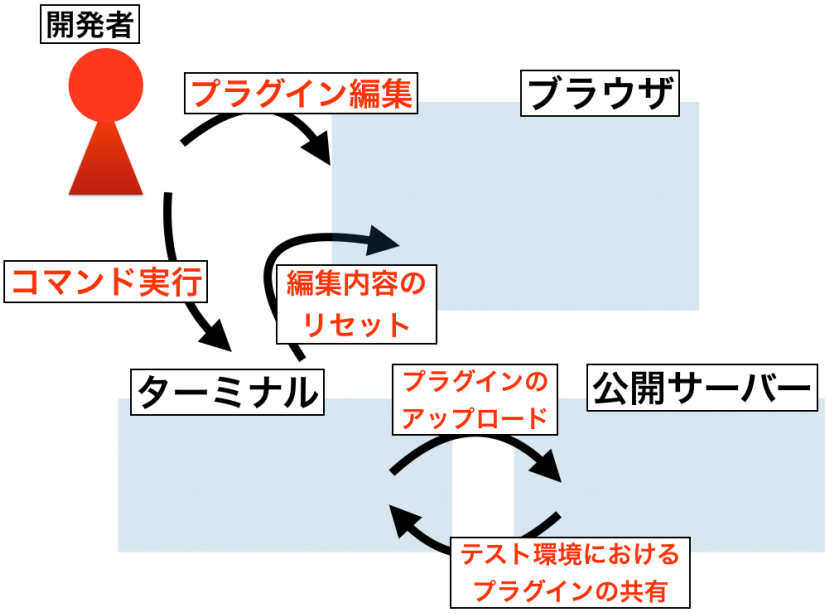


図 2.1: 本研究が目指す開発環境.

従来, プラグイン開発はターミナル上で Emacs, Vim 等の各種エディタによる編集をおこなっていた. しかし, プラグインのテストをおこなう為にはブラウザ上での動作を確認する必要があり, ターミナルとブラウザ間での反復的な画面移動が必要となってしまっている. 本研究では GUI によるプラグイン開発を可能とする事で Hiki プラグイン開発効率の改善を試みる. 加えて, GUI での編集システム運用におけるコマンドや CUI での編集コマンド実行の簡略化をおこなう.

2.1.1 GUI によるサポート機能

今回 Hiki プラグイン編集に使用する Hiki プラグインとして, たけうちひとし氏の comment.rb を参考に edit_plugin.rb を作成した [2]. edit_plugin の実装により, 図 1.1 で示した編集と動作テストに分かれていた操作が, 図 2.2 に示すように全てブラウザ上でおこなえるようになる. プラグインとしては Hiki ページ上へ`{{edit_plugin}}`の記述をおこなう事で呼び出せる.

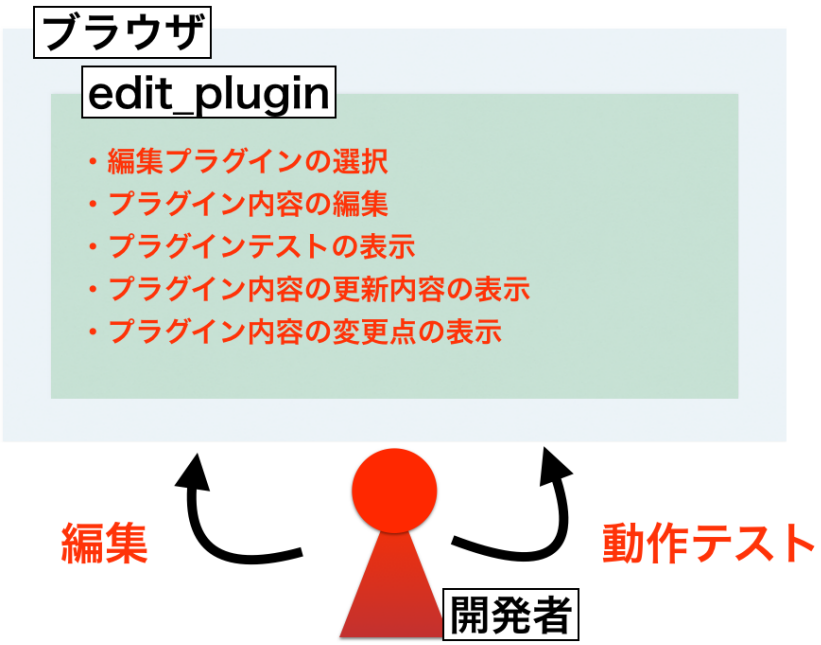


図 2.2: GUI によるユーザーの開発手順.

編集内容の表示と変更点の表示をおこなう事で編集作業の効率を向上させる. 変更点の表示は, diff-lcs(LCS は Longest Common Subsequence の略称) という 2 つの文字列に共通する最大の文字列を解析する Gem のパッケージを用いて表示する [3].

新規作成と既存のプラグインの編集の 2 種類のモード選択ができるようにし, 新規作成の場合には編集の手間が削減されるように選択したひな形が自動的に入力されるようにする. ひな形が提示される事により, Hiki プラグイン開発に不慣れな学生でもコードを書き始めやすくする.

また, Hiki ページ上で編集する際に編集内容にミスがあった場合に Hiki 自体のエラーに

より, Hiki ページ上での操作ができなくなるという事が考えられる. そうなった場合には, ターミナル上でコマンドを実行する事で 1 つ前の編集を取り消す事ができるようにする.

2.1.2 CUI によるサポート機能

CUI によるサポートは Hiki の中に Rakefile.rb という Rake の設定を記述したファイルを置く事で, Rake によってコマンドを呼び出せるようにする.

Rake

Rake とは Ruby の組み込みビルドユーティリティであり, 何度も実行するコマンド, あるいはコマンド群に対し, あだ名を付ける事で簡単に呼び出せるようになっている. 開発を進めるにあたって, 意味を理解しなくてもいいが何度も実行する為覚えておかなければならないというコマンドについては Rake を利用する事であだ名とその動作を覚えておくだけで良くなる為, 強力なツールであるといえる. Rakefile という Ruby の内部 DSL で記述されているファイルに設定が記述されており, その中で入力データやコマンドの実行結果等に対する処理も可能である [4].

Rake を用いて実装した機能

CUI においては, Rake を用いる事によって, プラグインの編集と動作テストができるコマンドを用意する. また, GUI における編集ミスの為のリセットコマンドや, 他者とのプラグイン共有の為のコマンドも用意した. Rake で実装したコマンドは以下の通りである.

- edit (引数としてプラグイン名を入力する事でコードを編集する事ができ, 編集終了時に自動的にテストページを表示する.)
- check_log (Apache のログを表示する.)
- only_codes (テストページの作成無しにプラグインのコード編集する.)
- reset (Hiki ページ上の編集で起きたエラーに対して 1 つ前の編集を取り消す.)
- upload (自分のプラグインと動作に必要なファイルを公開サーバへ送信する.)

- free_plugin(edit_plugin.rb で編集できないという状況に陥った時に実行する事で編集を可能にする.)

2.2 Chefを用いたHiki環境構築

Chef を用いる事で本来環境構築において必要となる作業を大幅に削減する事ができる. また, 想定環境に合わせた Chef のコードを用意する事で環境構築におけるミスを減らす事ができる. 本研究では, Chef を用いた Hiki 環境構築法を考案することで, Hiki 環境構築における手間の削減を試みる.

2.2.1 Chef

Chef とはミドルウェアレベルにおける構成管理ツールであり, Chef の設定ファイルはプログラミング言語 Ruby で記述される. 従来, プログラミング言語は, システム記述には向かないとされるが, Ruby というプログラミング言語の文法規則をそのまま使ってシステムが記述されている (内部 DSL). Chef はサーバの環境を料理に見立てて, その名の通り Chef が料理を作るような動作で環境を構築する. Chef の用語として, サーバの環境を構築する為の手順書を Cookbook, あるいは Recipe と呼ぶとされている. Chef は更に以下に挙げるような特徴がある.

1. サーバの環境設定をするにあたって, 用意されている手順書を参照して作業を進めていくというのがよく使われる手法であるが, Chef ではその手順書をそのままコードとして記述できる. これにより手順書に修正が必要となった時のミスを少なくできる [5].

2. Chef では「特定のコマンドを使用して, 特定のソフトウェアをインストールする」という考え方でなく, 「特定コマンドが使用されている状態であり, 特定のソフトウェアがインストールされている状態にある」という考え方になっている [6].

3. Chef では冪等性が重要であると考えられている. 冪等性とは何度実行しても同じ結果が得られるという性質のことであり, Chef では冪等性が基本的に守られるため,

何度実行しても同じソフトウェアが複数インストールされるということは起こらない [7].

Server-Client 方式, Chef-Solo と Chef-Zero の 3 種類の方式が用意されているが, 本研究では, Web 上での Recipe 配布した上でユーザーが各自 Chef-Solo の実行により環境構築するという利用形態を取る事にする.

Chef とその他の構成管理ツールとの比較を表 2.1 に示す [8]. 代表的な構成管理ツールの中でも, Chef が Ruby の内部 DSL で記述される事による Ruby 経験者にとって可読性が高い点, 冪等性が保たれるという点に着目し, 本研究では使用ツールとして Chef を選択した.

表 2.1: 構成管理ツールの比較表.

	冪等性	言語	記述形式	開発時期	学習コスト
Chef	○	Ruby	内部 DSL	2009	高
Puppet	○	Ruby	外部 DSL	2005	高
Salt	○	Python	内部 DSL or YAML	2011	高
Fabric	×	Ruby	内部 DSL	2011	低
Ansible	○	Python	YAML	2012	低

Ruby における DSL

Domain Specific Language(DSL) とは, それぞれに応じた分野のタスクに特化した言語である. ある言語の文法に則っている内部 DSL と文法自体がオリジナルである外部 DSL が存在する. Chef のコードは Ruby の内部 DSL で書かれており, 1.2 節にもあるように Ruby の経験者にとって可読性が高いという利点を持っている. しかし, 元々完成されている言語の文法に則っているため, その言語の表現から外れてしまうような自由度の高い表現はできない. 内部 DSL と比較すると外部 DSL は構築難度が高く, 初めてコードを見る人にとっては理解しにくい記述になってしまう. しかし, 外部 DSL の利点としては自分で 1 から文法を作る事で自由度の高い言語の作成ができる. Chef が持つ環境構築に必要となる操作の概要を Recipe から速やかに理解できるという利点は内部 DSL の特性によるものである [9].

2.2.2 Git

Git とは変更履歴等の保存ができる分散型の管理ツールである [10]. 本研究では変更履歴を含むリポジトリを複製できるという機能を使用する事で, 公開サーバからの Git を利用した Hiki のダウンロード及び開発したプラグインのアップロードを公開サーバに置いてある Git リポジトリをバージョンアップしていき, 他者がそこから複製するという利用法をとる.

2.2.3 Chef によっておこなう操作

Hiki 環境構築の手順を記した Recipe を実行する事によって, Hiki と本研究で開発した追加機能の実装をおこなう. これにより, Hiki 環境構築にかかる手間を大幅に削減できる. また, 既に Hiki 環境が構築されていても, 追加機能の実装のみが適用されるようにする. Hiki が置かれているパスと公開サーバにおけるユーザー名についての編集が必要となる. Recipe の編集については, Hiki 上にデータを入力する事でプラグインによる自動編集の上でのダウンロードができるようにする.

西谷研究室では, ほとんどの学生が MacOS のバージョンは v10.9 Mavericks と v10.10 Yosemite を使用しており, 本研究では最新バージョンである v10.11 El Capitan における環境構築は想定しない. また, Hiki 環境構築には, 公開鍵を用いた公開サーバとの SSH 接続をおこなう. 本研究で扱う Hiki 環境構築法において必要となる操作は以下の通りである.

1. Hiki 環境構築がされているかを確認, 既に環境構築されているならば, 6. から実行
2. Apache で参照する Sites というディレクトリを作成
3. Hiki 自身の動作に必要なソフトウェアをインストール
   (a) Homebrew がインストールされてなければインストール
   (b) Docdiff がインストールされてなければ, gem を用いてインストール
4. Apache の設定及び,Git によるインストール後に実行するコマンドの宣言
   (a) Apache の個人設定ファイル (user_name.conf, user_name には実際のユーザー名が入る.) が無ければ作成

(b) Hiki のデータ操作に必要なコマンド, Apache の個人設定ファイルに追記する為のコマンドを宣言

(c) 上記のコマンドに加え, プラグインの共有に必要な git のコマンド, プラグインの権限の変更のコマンドを宣言

(d) Apache の再起動ができるように Apache の操作についても宣言

(e) まず, Apache の個人設定ファイルに追記する内容を Template を基に一時ファイルとして作成

(f) Apache の設定ファイル (httpd.conf) の内容を Template を基に編集

(g) Apache が読み込むディレクトリに関しての設定ファイル (httpd-userdir.conf) を Template を基に編集

5. Git による Hiki のダウンロード

(a) Git がインストールされていなければインストール

(b) 指定されているパスに公開サーバから git clone により, hiki-1.0 を指定の名前でダウンロード

(c) 宣言しておいた hiki-1.0 中のデータの権限の変更のコマンド及び Apache の個人設定ファイルへの追記のコマンドを実行

(d) hiki-1.0 と hiki-1.0/data の権限を変更

6. 本研究で実装する追加機能に必要となる設定

(a) diff-lcs がインストールされてなければインストール

(b) 設定に必要となるコマンドを宣言

i. 追加機能を使用する為に, プラグイン全ての権限を変更するコマンドを宣言

ii. 他者とのプラグイン共有の為に git リポジトリの設定のコマンドを宣言

iii. require するファイルを共有する為に Git リポジトリを設定するコマンドを宣言

(c) Template により, 追加機能の動作に必要なファイルを作成

i. CUI によるサポートの設定の為の Rakefile.rb を Template を基に作成

ii. GUI によるサポートの設定の為の edit_plugin.rb を Template を基に作成

iii. テストページの自動更新の為に使用される loadtext.rb を Template を基に作成

iv. edit_plugin.rb が require によって変更点の識別の為に使用する diff_lcs.rb を Template を基に作成

v. Rake のコマンドを実行により, Apache のエラーログを解析する際に実行される log_parse.rb を Template を基に作成

vi. それぞれについて, 宣言しておいたコマンドを実行

(d) 追加機能に使用するディレクトリ構造を用意

i. require するファイルを格納するディレクトリを作成

ii. edit_plugin.rb による編集の取り消しの為のディレクトリと各種ファイルを作成

2.2.4 テスト環境の構築

Hiki は, プラグインが require するプログラムが指定されたパスに存在しない等のミスにより, 閲覧すらできない状態に陥ってしまうケースがある. プラグインの問題による自分の Hiki の停止を避ける為にも, 他者のプラグインの共有及び引き継ぎの時には, 図 2.3 に示すように, 別環境へのプラグイン実装をおこない, 動作を確認した後に自分の Hiki に実装するべきであると考える.

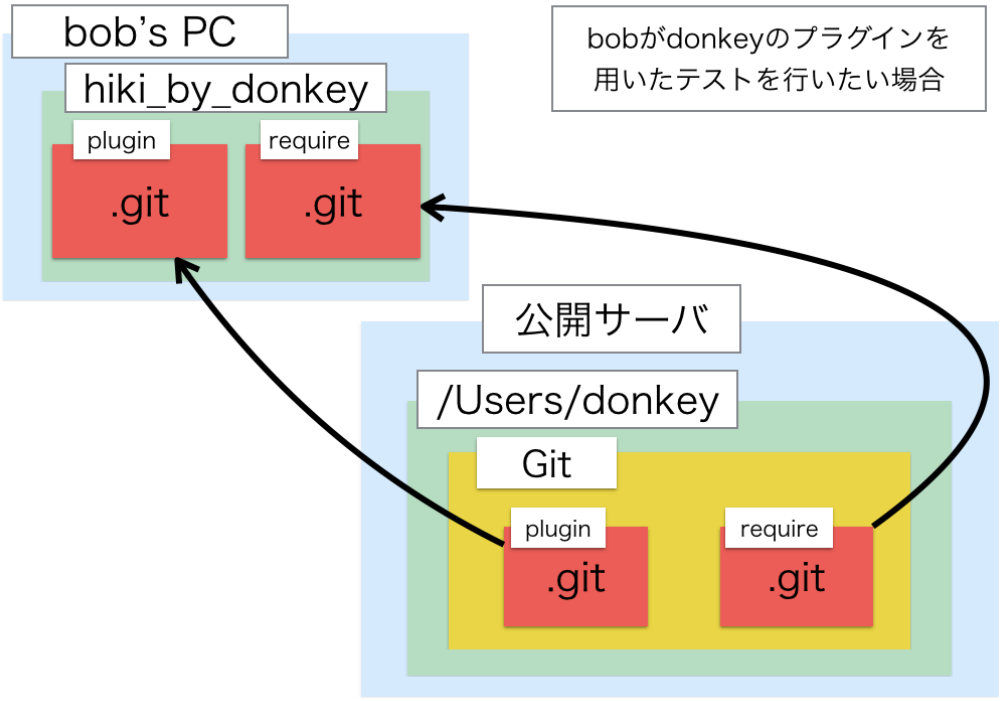


図 2.3: プラグインテストの環境構築のイメージ.

Chef による Hiki 環境構築法の考案により, Hiki 環境構築が迅速にできるようになる. この手法を応用する事で, 他者のプラグインをそのまま実装した状態で, 自分が利用する Hiki とは別の Hiki の環境構築をする事で開発の引き継ぎの際の誤動作を回避するための一時的に作業する環境を構築するための手法を考案する.

第3章　Hiki環境構築の具体的操作

3.1　Recipeの入手

図 3.1 のような Web 上から閲覧できる Hiki ページ上で公開サーバのユーザー名と Hiki のパスを入力する事で編集済みの Hikifor#{user_name} という Recipe をダウンロードできるようになっている.

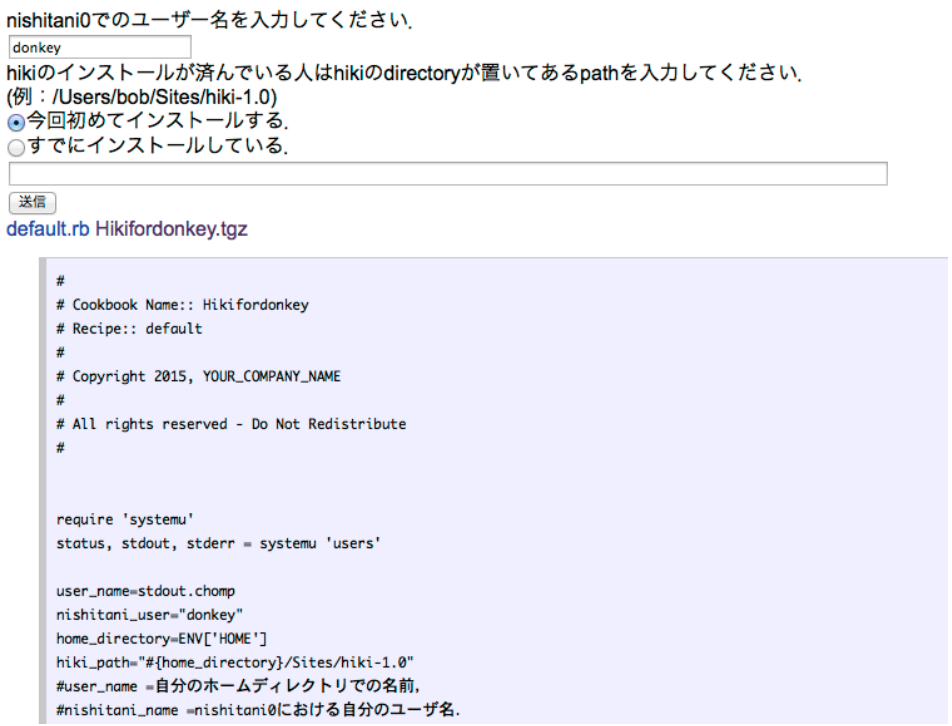


図 3.1: Recipe 配布の為のプラグイン.

まだ Hiki 環境構築をしておらず, Hiki のパスが存在しない場合には, ホームディレクトリの Sites というディレクトリに hiki-1.0 という名前で置かれるようになっている.

3.2 Recipeの実行

3.1節で挙げたページより, Recipeがtgzファイルでダウンロードされるので, 解凍した後に/var/chef/cookbooksに移動させ, Hikifor#{user_name}を実行する事で環境を構築する事ができる. 実際の環境構築のコマンドの例を以下に示す.

```
cp -r /Users/MorishitaShinya/Downloads/Hikifordonkey /var/chef/cookbooks/
```

#/Users/MorishitaShinya/Downloads/Hikifordonkeyの部分はターミナル上へディレクトリをドラッグ&ドロップする事で入力できる.

```
sudo chef-solo -o Hikifordonkey
```

3.3 Recipeの対象外の操作

プラグイン編集機能及び開発に必要となるプラグインのテストは可能となったが, Recipeで実行できるコマンド以外にHikiの初期設定が残ってしまう. この節では, 残された作業について述べる.

まず, Frontpageの作成と適用はGUIでの操作が求められるため, 自動化できない. しかし, この作業を自動化してしまうとHiki運用の1歩目を経験せずにHiki環境構築されてしまうため, この作業が自動化されていないのは問題ないと考える. ここでユーザーは, ガイドラインに則ってFrontpageの作成をおこない, ページ左上の管理ページへのリンクから, 適用プラグインを選択するという作業をする事となる. 作業として困難ではないため, 細かな説明は割愛するが, 以下に最初に適用しておくべきプラグインをまとめておく.

- attach.rb (ページ上にファイルのダウンロードのリンク付け，画像の表示等に使用される.)
- bbs.rb (Hikiページ上に掲示板を作成できる.)
- comment .rb(簡易なコメントフォームを作成する事ができる.)
- edit_plugin.rb (本研究にて作成したプラグイン開発環境を提供する為のプラグインである.)

- font.rb (文字の色付けや鎖線付けができるようにする.)
- loadtext.rb (ページの自動読み込みに使用されるプラグインである.)
- orphan.rb (ページの最初に簡易の目次を作る事ができる.)
- search.rb (キーワード入力によるページ検索できる.)

また, 上記の作業の後に hiki-1.0/data の内容が書き換えられるため, data ディレクトリ以下のファイルの権限の変更が必要となる. 操作をしないと, Hiki によるファイル作成等が正常におこなわれないようになってしまう. 権限の変更は以下のコマンドを実行する事で適用できる.

```
#hiki のパスへ移動している前提でのコマンドである.
sudo chmod -R 777 data
```

本研究で提供できるようになるテスト環境の使用の為に Hiki で開発するにあたって, 自分のプラグインを他者と共有できるように公開サーバへデータをアップロードしておく必要がある. データの共有に Git を用いるが, ここで公開サーバでの作業を Chef で自動化する事ができないため各自に Git リポジトリを作成してもらう事となる. 公開サーバで作業せず, 他者とデータ共有するため, git init のコマンドを実行する際に–shared 及び–bare のオプションをつける事とする. 必要となる作業は以下の通りである.

```
#公開サーバへログインして実行する.
mkdir Git
cd Git
mkdir plugin
cd plugin
git --bare init --shared
cd ..
mkdir require
cd require
git --bare init --shared

#ここから下のコマンドは自分の PC にて作業する.
rake upload
```

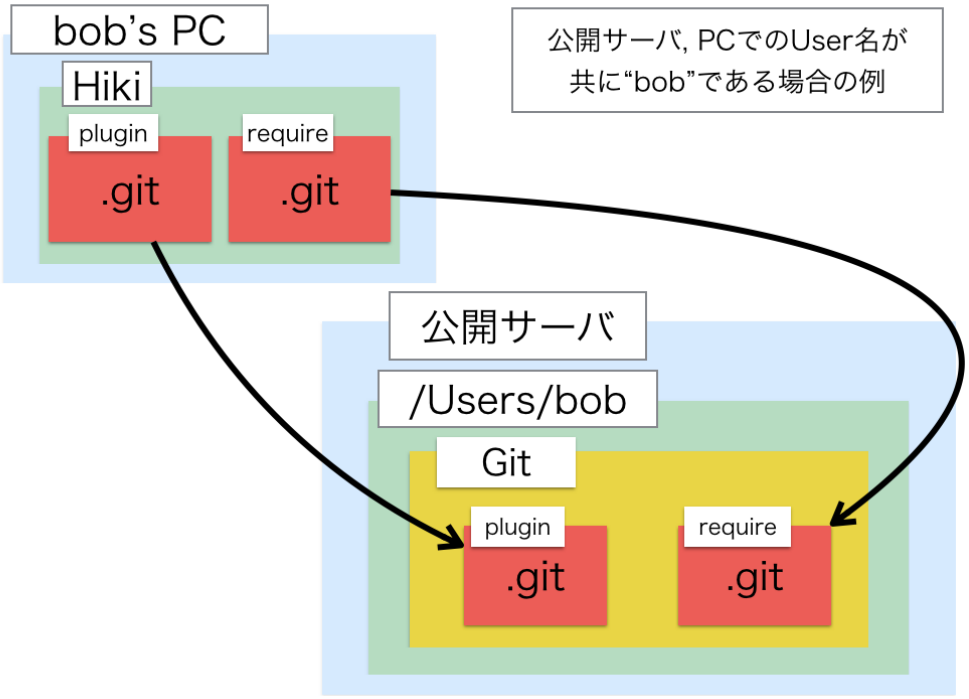


図 3.2: テスト環境構築に必要となるプラグインのアップロード.

コマンドの実行により図 3.2 に示したディレクトリ構造を構築でき, 公開サーバにプラグインをアップロードしておく事で, 開発者同士がお互いにプラグインを共有できるようになり, テスト環境をすぐに構築できるようになる.

3.4 テスト環境構築の手順

本研究で開発した TestSetting という Recipe を適用する事で, 指定した相手のプラグインを実装したテスト環境としての Hiki を構築する事ができる. TestSetting を適用するにあたって, 自分の公開サーバでのユーザー名に加え, プラグインを参照したい相手の公開サーバにおけるユーザー名を編集しておく必要がある. 動作としては, Hikifor#{user_name} と大きな違いは無く, 似た動作でもう 1 つの Hiki を hiki_by_#{reference_user} という名前で環境構築した後, プラグインを参照相手のものと置き換えられる. 実行コマンドの例を以下に示す.

```
cp -r /Users/MorishitaShinya/Downloads/TestSetting /var/chef/cookbooks/
#Recipe ダウンロードの後，ドラッグ&ドロップでパス指定できる．

sudo emacs /var/chef/cookbooks/TestSetting/recipes/default.rb
#nishitani_user と reference_user を編集する．

sudo chef-solo -o TestSetting
```

第4章　プラグイン開発の具体的操作

4.1　GUIでのサポート

4.1.1　ユーザーの行動遷移

Hiki 上での編集において, ユーザーが取る行動の遷移は以下の通りである.

1. Hiki ページの編集画面で{{edit_plugin}}と入力する事でプラグインを呼び出す
   (a) 新規作成する場合は, 使用するひな形の種類を選択した上でプラグイン名を入力して編集を開始する.
   (b) 既存のプラグインを編集する場合には, ラジオボタンにより編集プラグインを選択して編集を開始する.
2. ひな形あるいは編集前の内容がテキストエリア内に表示されるので編集する.
3. 編集が完了した時に実際に呼び出すプラグイン名を入力する.
4. 変更点, 変更後の内容, プラグイン適用のテスト結果が表示されるので編集を続ける場合には再度編集する.

4.1.2　編集画面の遷移

まず, edit_plugin を呼び出すと, 図 4.1 のような出力が得られる.

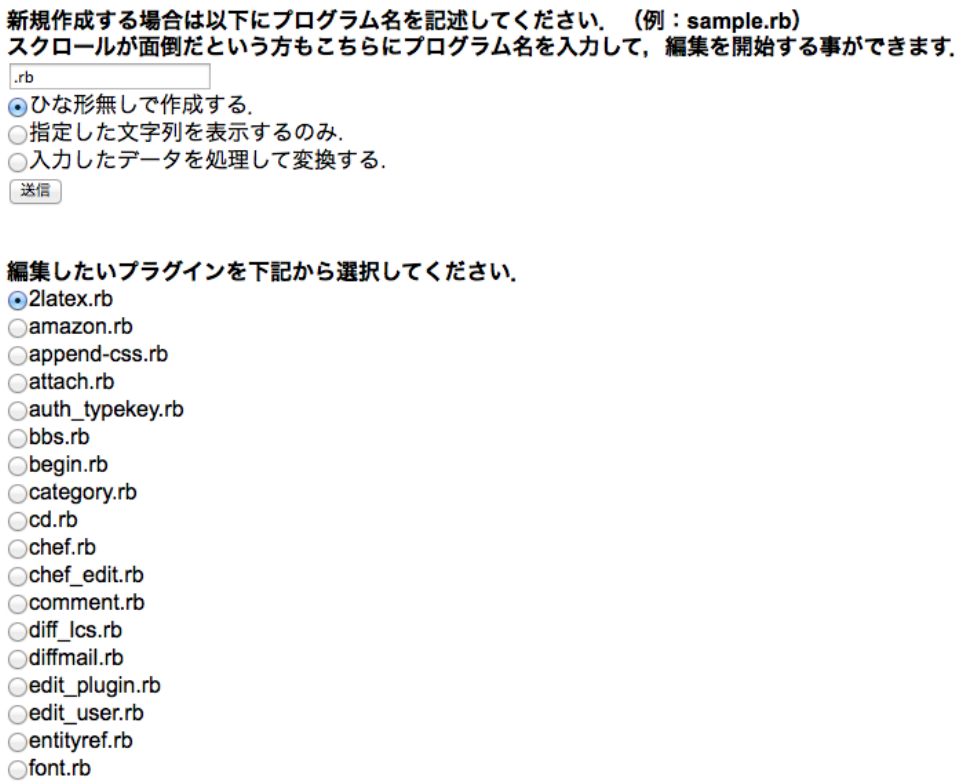


図 4.1: 編集ファイルの選択画面．

編集ファイルの選択画面として, 新規作成するプラグイン名の入力欄と使用するひな形を選択する為のラジオボタンが表示される．また, 既存のプラグインを編集する為のプラグイン選択用のラジオボタンが表示され, それぞれに対応した送信ボタンをクリックする事で編集画面へ遷移する．また, 新規作成するプラグイン名の入力欄に既存のプラグイン名を入力した場合にも既存のプラグインを編集できるようになっている．

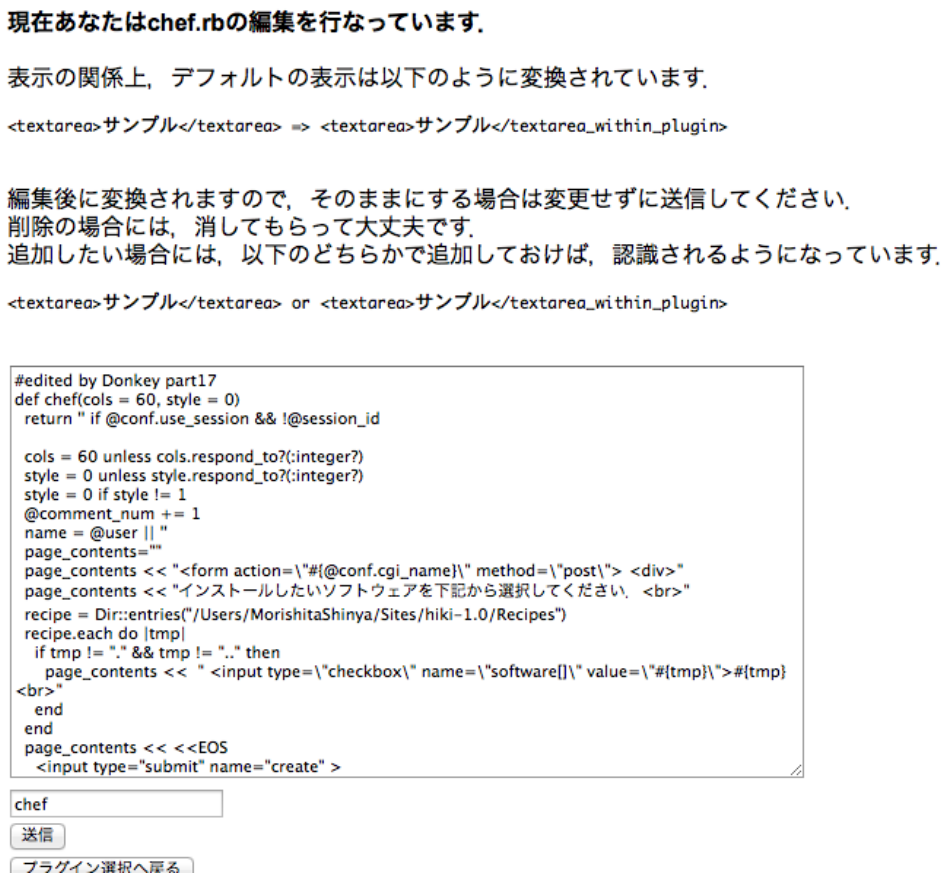


図 4.2: 1 度目の編集画面.

編集プラグインの選択後に表示される 1 度目の編集画面を図 4.2 に示した. 1 度目の編集画面ではテキストエリアにその時点の編集対象であるプラグインの内容が表示されるので, ユーザーはテキストエリア内を編集する. ここで問題となったのは, Hiki プラグインの内容に HTML の記述が含まれる事が多いため, プログラム内に`</textarea>`の記述があると正常な表示ができない事である. そのため, edit_plugin.rb による編集の際には`</textarea>`の記述を編集中のみ`</textarea_within_plugin>`という記述に変換し, 保存時に元に戻すという形で対応する事とした.

編集が終了した後に, テキストエリアの下にあるテキストボックスに呼び出したいプラグイン名を記述する事で, 指定したプラグインが次の画面でのテストで呼び出される. デフォルトの入力はプラグインのファイル名から.rb が取り除かれたものが入力されているので, 違う名前のプラグインを呼び出したい場合には, 前述の通りにプラグイン名を呼び出す事となる.

1度目の編集画面においてひな形の種類を選択した場合には対応したひな形がそれぞれデフォルトとしてテキストエリアに呼び出され, その内容で hiki-1.0/misc/plugin に保存される. 以下に今回用意した2つのひな形を示す.

```
def test_for_thesis2
  "sample by donkey!"
end
```

上記のひな形は呼び出すと, デフォルトで”sample by donkey!”と表示する.

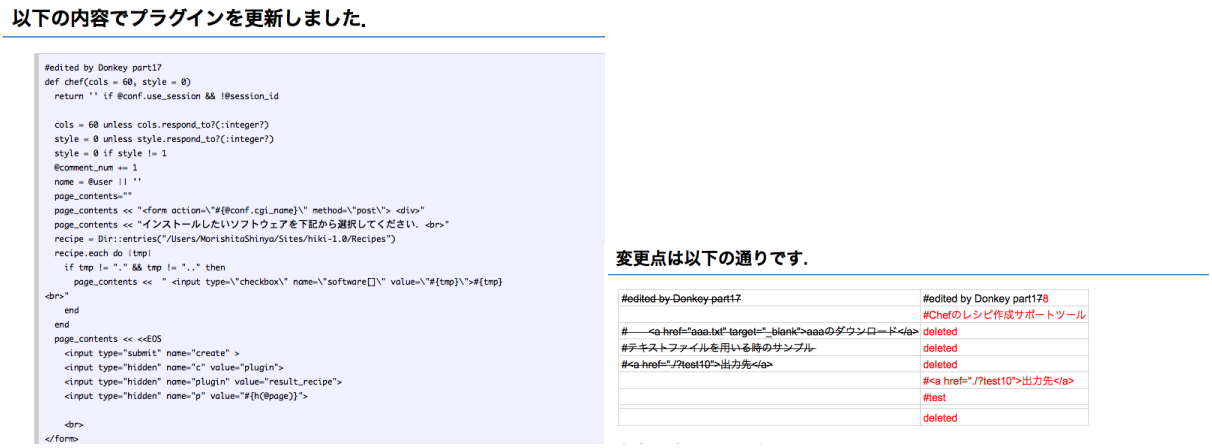
```
def test_for_thesis3(cols = 60, style = 0)
  return '' if @conf.use_session && !@session_id

  cols = 60 unless cols.respond_to?(:integer?)
  style = 0 unless style.respond_to?(:integer?)
  style = 0 if style != 1
  @comment_num += 1
  name = @user || ''
  page_contents=<<EOS
    <form action="./" method="post"> <div>
    <input type="text" name="param_name"  size="20" maxlength="20"><br>
    <input type="submit" name="create" >
    <input type="hidden" name="c" value="plugin">
    <input type="hidden" name="plugin" value="test_for_thesis3_after">
    <input type="hidden" name="p" value="plugin_test">
    <br>
</form>
 EOS

#ひな形の関係上，EOS の左にスペースが入ってますが削除して使用してください．
end

def test_for_thesis3_after
  md5hex = @db.md5hex( @page )
  params     = @request.params
  #  style      = params['style'].to_i
  content = ""
  content << ""
  content << "{{test_for_thesis3}}"
  save( @page, content, md5hex )
end
```

上記のひな形はテキストボックスを表示し, そこに入力したテキストを処理用の関数に受け渡す動作が記述されている.



(a) 更新後の内容の表示.　　(b) 変更点の表示.

図 4.3: 2 度目以降の編集画面.

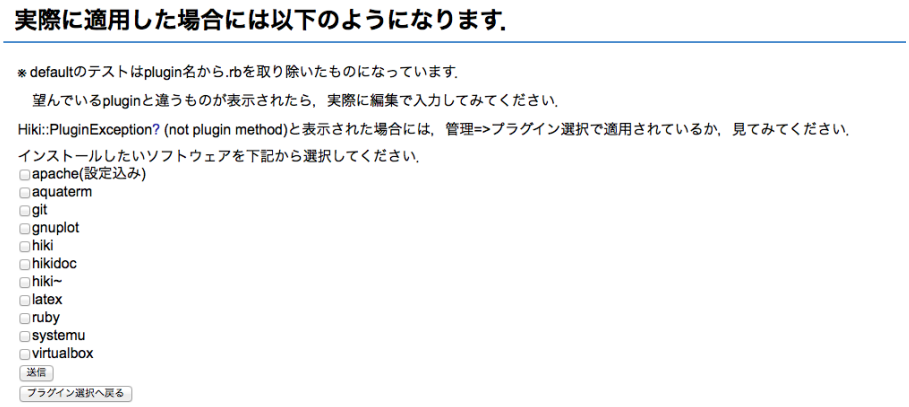


図 4.4: プラグインテストの表示.

1 度目の編集が終わり, 編集内容を送信すると, 2 度目以降の編集画面が呼び出される. 2 度目以降の編集画面では, 1 度目の編集画面の表示に加えて, 図 4.3 に示したような更新後のプラグイン内容, 元のプラグインの内容からの変更点が表示される. また, 図 4.4 のようにプラグインを実際に適用した時のテストが表示される. 新規作成の場合には管理画面から適用プラグインとして編集したファイルを追加で選択しなければならない.

編集を中断して他のプラグインの編集をしたい場合には”プラグイン選択に戻る”と書いてあるボタンをクリックする事で, 編集プラグインの選択画面へと戻る事ができるようになっている.

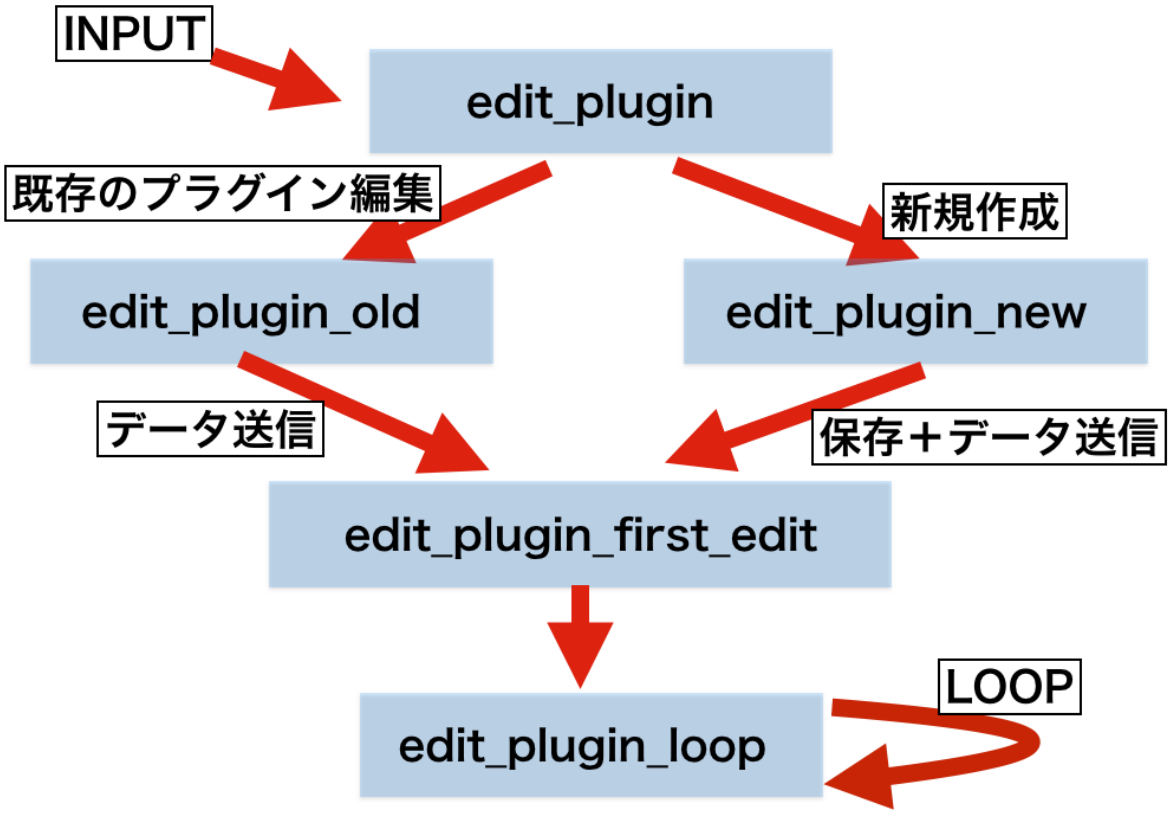


図 4.5: プラグイン間でのデータ受け渡しの流れ.

ページ移動しながら編集しているように見えるが, 実際には Hiki のフォーム送信の記述が不透明である事から実装は困難であったため, ページ移動は実装できていない. この一連の操作は同一のページ内が書き変わっていく事で擬似的なページ移動をしている. この擬似的なページ移動は edit_plugin に記述されている 7 つのプラグインが, 図 4.5 に示したようなデータの受け渡しをする事で実現している. 少し回りくどい呼び出し方をしているプラグインも存在するが, これはプラグイン内で扱う記法が HTML か Hiki 記法にのどちらかに統一する必要がある為である. 各プラグインはそれぞれ以下のような動作をしている.

- edit_plugin (図 4.1 で示している編集プラグインの選択画面を表示する.)
- edit_plugin_first_edit (図 4.2 で示している 1 度目の編集画面を表示する.)
- edit_plugin_old (編集プラグインの選択画面における既存のプラグインの編集について, edit_plugin_first_edit へのデータの受け渡す.)
- edit_plugin_loop (2 度目以降の編集画面における更新内容, 変更点, プラグインテストを表示する. また, データが送信されるたびに再帰的にページを更新し続ける.)
- return_button (編集プラグイン選択画面に戻るボタンを表示するプラグインであり, return_process の呼び出しの記述が書かれている.)
- return_process (edit_plugin5 によって呼び出され, ページ上で edit_plugin を呼び出すようにページを書き換える.)
- edit_plugin_new (編集プラグインの選択画面におけるプラグインの新規作成について, edit_plugin_first_edit へのデータの受け渡しと新規プラグインの保存をおこなう.)

4.2 CUI でのサポート

CUI でのサポートは Rake によって実現され, hiki-1.0 に置かれている Rakefile.rb の記述によって, 2.1.2 節で挙げたコマンド群を実行する事ができるようになっている. 編集用のコマンドである rake edit を実行すると emacs による指定したプラグインの編集ができる. また編集が終了すると同時に plugin_test というページが無ければ作成し, 編集したプラグインのテストが表示されるようになっている. 例えば, rake edit test と実行すると以下のような内容が plugin_test に書き込まれる.

```
{{loadtext(plugin_test)}}
{{test}}
```

また, テストページが自動的に開かれるようになっており, loadtext.rb による自動更新されるようにしているため, 速やかにプラグインの出力をテストできるようになっている.

実行可能な段階まで編集できそうにない場合には rake only_codes を実行する事でテスト無しで編集できるようになっている.

rake upload を実行する事で, require の内容と misc/plugin の内容が git push の実行により, 現在の自分のプラグインを公開サーバへアップロードする事ができる. rake reset では backup_plugin の中に格納されているデータを使用する事で edit_plugin.rb による 1 つ前の編集を取り消す事ができるようになっている. また, edit_plugin.rb での編集は, ターミナル上での Emacs 操作により作成したプラグインを編集できない仕様となっている. 編集を可能にしたい場合には rake free_plugin を実行する事で編集可能にする事ができる.

第5章　現状の評価

本研究で目的としていた環境構築における手間の削減, 及び Hiki プラグイン環境の整備は達成できていると考える. しかし, 以下に挙げる問題点が残されている.

- Hiki の表示の都合上やむを得ないテキストエリア内の変更, プラグインテストの為のデータアップロードや require するファイルの置き場所の指定等, 今後ユーザーに対しての習慣付けに期待しなければいけないところが多くあるのは問題であると考える.

- GUI によるプラグインの編集は通常のテキストエリア内でおこなう事となり, Emacs のような自動的なコードへの色付けがなされないため, Chrome 及び Firefox のアドオン等, ブラウザ上での編集に関するシステムを各々が準備しなければならない.

- 権限の関係上, edit_plugin.rb あるいは Rakefile.rb によって権限を変更されていないファイルは edit_plugin.rb による編集ができないという問題が残されている. rake free_plugin の実行によって編集可能な状態にできるが, ここでブラウザからターミナルへの移動が必要になるため, 根本的な解決にはなっていない.

開発からまだ日も浅く, テストケースも決して多いとは言えない為, 今後の運用を通して更なる改善を進めていく必要がある. しかし, Chef のコードを読む事で実行される動作を把握できるので, 今後本研究のシステムを運用していくにあたり, 容易に継続的な改善ができるようになっている.

第6章　総括

本研究では Chef の利用による Hiki 環境構築の自動化及び Hiki プラグイン開発環境の改善をおこなった. 本研究で得られた成果をまとめると以下の通りである.

1. GUI により編集を容易にする edit_plugin.rb という Hiki のプラグインを開発した. これにより, ターミナルとブラウザの間の移動の必要がなくなり効率的な開発ができるようになった.

2. Hiki プラグイン開発において必要となるコマンドの呼び出しを Rake で呼び出す為に Rakefile.rb を用意した. これにより, プラグインの編集, アップロード及び Apache のエラーチェック等のターミナル上での作業の効率が上がった.

3. Hiki 環境構築をおこなう Chef の Recipe を開発した. これにより, コマンド 1 つで Hiki 環境が構築でき, 研究準備の手間が削減された. 加えて, Recipe を読む事で動作内容を把握できるため, 今後運用を続けていくにあたっての環境構築手法の継続的な改善を効率的におこなえる.

4. テスト環境構築をおこなう Chef の Recipe を開発した. この Recipe の適用により, 指定したユーザーのプラグインを実装した Hiki を迅速に環境構築できるため, プラグイン内の記述により Hiki が閲覧不可の状態に陥るといったリスクを回避したプラグインのテスト環境を容易に構築できる.

謝辞

本研究の進行にあたり, 終始多大なるご指導ご鞭撻を頂いた西谷滋人教授に深く感謝致します. また, 本研究における, 知識の供給, テストへの参加をして頂きました, 西谷研究室の同輩, 先輩, 後輩の方々に心から感謝の意を示します. 本当にありがとうございました.

参考文献

[1] 「Hiki」, たけうちひとし, http://hikiwiki.org/ja/about.html, 2016/2/3 アクセス.

[2] 「comment.rb」, Hiki development team, http://hikiwiki.org/ja/comment.rb.html, 2016/2/8 アクセス.

[3] 「diff-lcs」, Austin Ziegler, https://rubygems.org/gems/diff-lcs/versions/1.2.5, 2016/2/8 アクセス.

[4] 「Ruby ベストプラクティス」, Gregory Brown, (オライリー・ジャパン 2010).

[5] 「今からでも始められる Chef 入門」, kasaharu, http://qiita.com/kasaharu/items/55a3000db31c52ce0bd7, 2015/9/3 アクセス.

[6] 「Chef 実践入門」, 吉羽 龍太郎, 安藤 祐介, 伊藤 直也, 菅井 祐太朗, 並河 祐貴, (技術評論社 2014).

[7] 「今更聞けない人の為の Chef 再入門」, School With, http://blog.schoolwith.me/chef-re-introduction/, 2015/9/10 アクセス.

[8] 「What is an Ansible?」, Shunsaku Kudo, http://www.slideshare.net/kudo.shunsaku/what-is-an-ansible, 2016/2/3 アクセス.

[9] 「Ruby で DSL」, Yukimitsu Izawa, http://www.slideshare.net/yizawa/rubydsl-25541986, 2016/2/2 アクセス.

[10] 「six apart ブログ」, にっく, http://blog.sixapart.jp/2014-03/mttips-02-what-is-git.html, 2016/2/3 アクセス.

付 録A　Hikifor#{user_name}

以下のプログラムは, Chefが実行する動作が記述されているファイルであり, 置くべきパスは/var/chef/cookbooks/Hikifordonkey/recipes/default.rb である.
この Recipe は TestSetting と合わせて,
http://nishitani0.kwansei.ac.jp/~donkey/hiki-1.0/?HikiStart から入手できるようにしている. 記述されているコードがどのような動作を意味するのかをコメントアウトで示している.

```
#
# Cookbook Name:: Hikifordonkey
# Recipe:: default
#
# Copyright 2015, YOUR_COMPANY_NAME
#
# All rights reserved - Do Not Redistribute
#
#公開サーバのユーザー名は"donkey", Hiki のパスはデフォルトのものを使用している.

require 'systemu'
status, stdout, stderr = systemu 'users'

user_name=stdout.chomp
nishitani_user="donkey"
home_directory=ENV['HOME']
hiki_path="#{home_directory}/Sites/hiki-1.0"
#user_name =自分のホームディレクトリでの名前,
#nishitani_name =公開サーバにおける自分のユーザ名.
#hiki_path=Hiki を置くパス.
```

```
#ホームディレクトリを環境変数から参照し，Hiki のデフォルトのパスを指定するようにしている．

if File.exists?("#{hiki_path}/hiki.cgi")==false then
#hiki.cgi の存在を使用して Hiki 環境構築がされているかどうかを判定する．

directory "#{home_directory}/Sites" do
  owner user_name
  mode 0755
end
#Apache が参照ディレクトリとする Sites というディレクトリを作成する．

bash "install homebrew" do
  user user_name
  cwd home_directory
  environment "HOME" => home_directory
code "ruby -e \"\$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)\""
  creates "/usr/local/bin/brew"
end
#Git のインストールに必要となる HomeBrew をインストールする．
#厳密にはインストール用のコマンドを実行するという記述になっている．

gem_package "docdiff" do
  action :install
  version "0.5.0"
end
#Hiki の動作に必要となる docdiff という Gem のパッケージをインストールする．

file "#{user_name}.conf" do
 path "/etc/apache2/users/#{user_name}.conf"
  mode "0644"
  action :create_if_missing
end
```

```
#Apache の設定ファイルである user_name.conf が無ければ作成する.

execute "mv" do
  command "mv data_init data"
  cwd hiki_path
  action :nothing
end

execute "cat" do
  command "cat /etc/apache2/users/#{user_name}_tmp.conf >>
 /etc/apache2/users/#{user_name}.conf"
  cwd home_directory
  action :nothing
end

execute "chown" do
  command "chown _www data"
  cwd hiki_path
  action :nothing
end
#Git によるダウンロードの後に実行する各コマンドを宣言しておく.
#ここでは実際に実行しない.

service "httpd" do
  supports :status => true, :restart => true, :reload => true
  action [ :start, :enable  ]
end
#Apache の再起動の為の Apache 呼び出しについての詳細を設定しておく.

template "#{user_name}_tmp.conf" do
 path "/etc/apache2/users/#{user_name}_tmp.conf"
  if node["platform_version"].to_f== 10.9
    source "httpd10_9.conf.erb"
  elsif node["platform_version"].to_f== 10.1
```

```
    source "httpd10_10.conf.erb"
  end
  owner  "root"
  mode 0644
  notifies :restart, "service[httpd]"
  variables({
    :home_name => user_name
            })
end
#user_name.conf に追記する内容を一時ファイルとして保存する.

template "httpd.conf" do
  path "/etc/apache2/httpd.conf"
  if node["platform_version"].to_f==10.9
      source "10_9.conf.erb"
  elsif node["platform_version"].to_f== 10.1
      source "10_10.conf.erb"
  end
  owner  "root"
  mode 0644
  notifies :restart, "service[httpd]"
end
#Template を基に httpd.conf を編集する.

template "httpd-userdir.conf" do
    path "/etc/apache2/extra/httpd-userdir.conf"
    source "httpd-userdir.conf.erb"
    owner  "root"
    mode 0644
    notifies :restart, "service[httpd]"
end
#Template を基に httpd-userdir.conf を編集する.

package "git" do
```

```
  action :install
end
#Git がインストールされていなければインストールする.

gem_package "diff-lcs" do
  action :install
end
#編集の変更内容の表示に使用する diff-lcs という Gem のパッケージをインストールする.

git "#{hiki_path}" do
    repository "#{nishitani_user}@nishitani0:/Users/bob/Git/hiki-1.0"
    reference "master"
    action :checkout
    user user_name
    notifies :run, "execute[mv]"
    notifies :run, "execute[chown]"
    notifies :run, "execute[cat]"
  end
end
#Git により hiki_path により指定したパスへ Hiki をダウンロードする.

directory "#{hiki_path}" do
  mode "0777"
end

directory "#{hiki_path}/data" do
  mode "0777"
end
#Hiki のデータの権限をそれぞれ変更する.

execute "chmod_plugin" do
  command "chmod -R 777 misc/plugin"
  cwd hiki_path
  action :nothing
```

```
end

execute "chown_plugin" do
  command "chown -R _www misc/plugin"
  cwd hiki_path
  action :nothing
end

execute "git_init_plugin" do
  command "git init"
cwd "#{hiki_path}/misc/plugin"
  action :nothing
end
#プラグインの権限の変更及び Git リポジトリ作成のためのコマンドを宣言する.

template "Rakefile.rb" do
  path "#{hiki_path}/Rakefile.rb"
  source "Rakefile.rb.erb"
  owner  user_name
  mode 0644
  notifies :run, "execute[git_init_plugin]"
  variables({
              :hiki_user => user_name,
              :hiki => hiki_path,
              :nishitani_name => nishitani_user
            })
end
#Template を基に CUI におけるサポートを記述した Rakefile.rb を作成及び編集する.
#また, この編集がおこなわれた時にプラグインが置かれているディレクトリに Git リポジトリを作成するコマンドを実行する.

template "edit_plugin.rb" do
  path "#{hiki_path}/misc/plugin/edit_plugin.rb"
  source "edit_plugin.rb.erb"
```

```
  owner  "_www"
  mode 0777
  notifies :run, "execute[chmod_plugin]"
  notifies :run, "execute[chown_plugin]"
  variables({
              :hiki_user => user_name,
              :hiki => hiki_path,
              :nishitani_name => nishitani_user
            })
end
#Template を基に GUI におけるサポートを記述した edit_plugin.rb を作成及び編集する.
#また，この編集がおこなわれた時にプラグインの権限変更についてのコマンドを実行する.

template "loadtext.rb" do
  path "#{hiki_path}/misc/plugin/loadtext.rb"
  source "loadtext.rb.erb"
  owner  "_www"
  mode 0777
  variables({
              :hiki_user => user_name,
              :hiki => hiki_path,
              :nishitani_name => nishitani_user,
              :home_dir => home_directory
            })
end
#Template を基にページの自動読み込みに使用する loadtext.rb を作成及び編集する.

execute "git_init_require" do
  command "git init"
   cwd "#{hiki_path}/require"
  action :nothing
end
#require するファイルをおくディレクトリに Git リポジトリを作成するコマンドを宣言する.
```

```
directory "#{hiki_path}/require" do
  owner user_name
  mode "0755"
  action :create
  notifies :run, "execute[git_init_require]"
end
#require するファイルをおくディレクトリを作成する.
#作成した場合には Git リポジトリの作成の為のコマンドを実行する.

template "diff_lcs.rb" do
  path "#{hiki_path}/require/diff_lcs.rb"
  source "diff_lcs.rb.erb"
  owner  user_name
  mode 0755
end
#Template を基に編集の変更点を表示する為の diff_lcs.rb を作成及び編集する.

template "log_parse.rb" do
path "#{hiki_path}/log_parse.rb"
  source "log_parse.rb.erb"
  owner  user_name
  mode 0644
end
#Template を基に Apache のログを解析する為の log_parse.rb を作成及び編集する.

directory "#{hiki_path}/plugin_backup" do
  owner "_www"
  mode "0755"
  action :create
end

file "edited.txt" do
  owner "_www"
  mode "0777"
```

```
  action :create
end

file "edited_plugin.txt" do
  owner "_www"
  mode "0777"
  action :create
end

file "pre_edit.txt" do
  owner "_www"
  mode "0777"
  action :create
end
#プラグインの編集内容の復元の為のディレクトリ構造を作成する.
```