平成18年度 修士論文

ユーザの位置情報を利用した moblog投稿・検索システムの設計と実装

電気通信大学 大学院情報システム学研究科

情報システム設計学専攻

0550035 前澤 直洋

指導教員 多田 好克 教授
前川 守 教授
大森 匡 助教授

提出日 平成19年1月30日

目 次

図 目 次

第 1 章

はじめに

近年、携帯移動端末の普及とその Web 利用環境の発展は目覚しく、今や誰でも、いつでも、どこからでも情報を扱うことができる環境にある。このようなモバイル環境において、そこから生まれる情報の特に重要な鍵となるのが、その情報が示す位置である。たとえば「今ここでこんなことが起こっている」、「このお店の料理がおいしい」という情報では、伝えたいことの重点はその情報の示す位置であり、一方、この情報を見たユーザが興味を示す点も、やはりその情報の指す位置であろう。たとえば見知らぬ土地に観光にきたとき、上記のような、その土地に特化した情報を知りたくなることがよくある。しかしそのような局所的な情報を、Web サイトから見付けるのに従来のキーワード検索を用いていては、特に移動中の多いモバイル環境では不便である。なぜなら、小型化された入力デバイスからのキー入力はそれだけでも手間が掛かってしまうというのに、移動中に画面の確認とキー入力を繰り返すことはさらに難しいためである。このようなモバイル環境では、今いる自分の位置情報を自動的に取得し、検索クエリとした周辺の地域情報検索を行いたいという要求がある。

本研究ではこのような要求を満たすため、情報端末として GPS 搭載携帯電話、情報媒体として Weblog を用い、位置と密着した情報の投稿と検索を実現するシステム、live-log システムの設計と実装を行った。

live-log システムは携帯電話と Weblog システムの間に立ち、情報を仲介する役割を担う。投稿時、携帯電話から Weblog システムへの投稿処理を請け負いながら、

データベースによって位置と情報を結びつける。このデータベースを利用することで位置情報をキーとした情報検索を可能にする。そして、その検索結果の提示として、本システムは位置と結び付いた Weblog 記事へのポータルを提供する。

本論文では、まず第 2 章より本システムの概要を見ていく。続いて、第 3 章と第、第 4 章で本システムの実装について議論する。第 5 章では今回開発したシステムの評価する。また、第 6 章では、関連研究・関連システムを述べ本システムとの比較する。そして、第 7 章で本論文をまとめる。

第 2 章

システムの概要

本章では、まず本システムの概要について述べる。

本システムはユーザのいる位置と、そこから発信される情報を結びつける Web サービスを提供する。

携帯電話から投稿された Weblog は特に moblog と呼ばれる。本システムでは GPS 搭載携帯電話から投稿された moblog に、位置情報を付加する。そして、位置情報を付加した moblog をユーザの位置情報から発見する。これにより、位置を中心とした情報共有サービスを提供する。本システムによる情報共有のイメージを図 2.1 に示す。この図のように、本システムは携帯電話と Weblog システムとの間に立ち、仲介役としてはたらくことで、地域に特化した情報の発信と検索を支援する。

本システムは携帯電話の Web ブラウザからアクセスされる Web アプリケーションである。携帯電話に標準搭載されている Web ブラウザで利用できるため、特にアプリケーションをインストールする必要はなく、どの機種、どのキャリアからでも利用することができる。動的に展開される Web コンテンツを対話的に操作することで、記事の投稿・検索だけでなく、編集や一時保存などさまざまな機能を利用することができる。

一方、moblog の投稿先である Weblog システムも汎用のものを対象とする。たとえば Movable Type や So-net blog など、一般的な Weblog システム・サービスを利用できる。

このように汎用の携帯電話と Weblog を利用できるため、より多くのユーザと

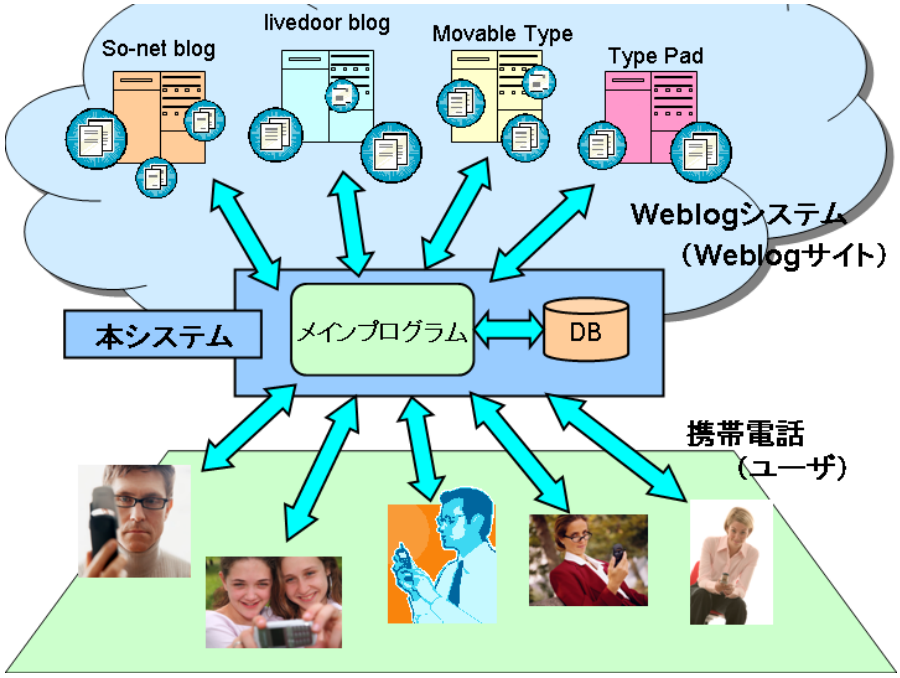

図 2.1: 本システムを中心とした情報共有イメージ

Weblog 記事との間で情報を共有することが可能となる。

特に、位置による情報検索の結果を、携帯電話用ポータルサイトとして提供する。ここに表示されるリンクを通して、ユーザのいる位置周辺で投稿された moblog 記事を含む Weblog サイトに訪れることができる。これにより、位置をきっかけとした横断的なコミュニケーション手段としても利用できる。

なお、以下では言葉の混乱を防ぐため、単に Weblog と呼んだ場合は、Weblog と moblog の両方を表すこととする。

2.1 位置情報を付加した Weblog 記事の投稿

ここでは本システムのメインな機能のひとつである、投稿について述べる。

本システムにから投稿した Weblog 記事の例を図 2.2 に示す。この記事にはタイトル、本文、写真の他に、この記事を書いた位置情報も示すことができる。ここに

図 2.2: 本システムから投稿した Weblog 記事

は緯度・経度だけでなく、その位置座標を中心とした地図も提示するため、視覚的にこの記事の指す位置を閲覧者に伝えることが出来る。

以下では図のような記事を作成する流れを説明する。

1. 位置情報の取得
2. 周辺スポットの選択
3. 投稿記事の記入
4. 投稿する Weblog の選択
5. 投稿

2.1.1 位置情報の取得

位置情報の取得も、Web ブラウザのインタフェースを通して、取得することができる。

本システムのメインメニュー (図 2.3) から「記事を書く」を選択すると、端末にある GPS が起動し、ユーザの現在の位置がシステムに送信される。

取得した位置情報はは位置確認画面 (図 2.4) で確認できる。

2.1.2 周辺スポットの選択

位置情報を確認すると、「スポット」という情報を選択する周辺スポット一覧画面 (図 2.5) が表れる。

本研究では、記事に与えるメタ情報のひとつとして、「スポット」という独自概念の情報を用意した。

スポットとは「場所の名前」でラベル付けした位置情報のことである。位置を表す場合、緯度・経度は的確にその位置座標を示してくれるが、それは人間にとっては分かりにくい。人間が位置を認識するとき、緯度・経度や住所ではなく、たと

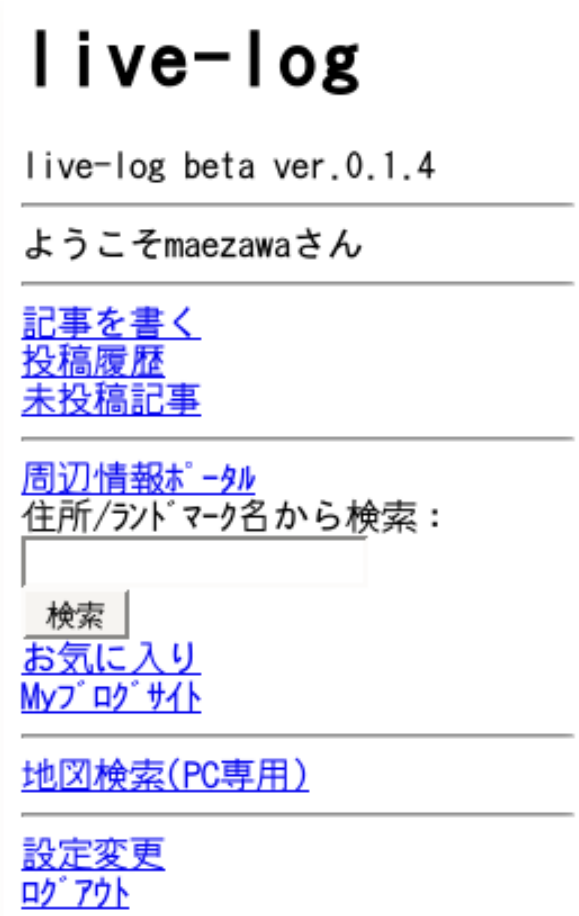

図 2.3: メインメニュー画面

図 2.4: 位置確認画面

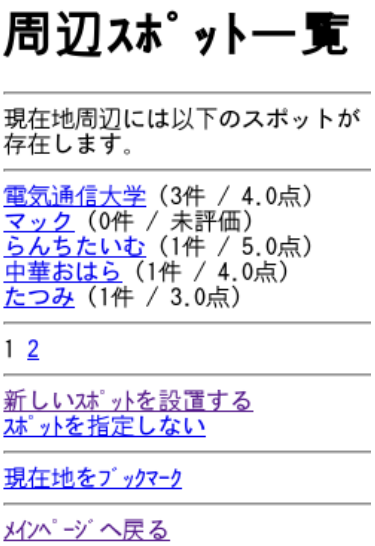

図 2.5: 周辺スポット一覧画面

えば「電気通信大学」あるいは「布田天神」など、 その位置が属している「場所の名前」であることが多い。そこで、投稿時、ユーザにいる位置に「場所の名前」を与えてもらい、検索時にそれを役立てようとするのが目的である。

スポットは他のユーザと共有することができる。投稿時、位置情報を取得すると、現在地から半径 1km 以内周辺の既存スポットが距離順で一覧表示される。そして、この中に今書こうと思っているスポットがあれば、それを選択してもらう。こうすることで、各「位置」に分散した記事をひとつの「スポット」に属させることができ、検索ヒット記事への誘導性の向上や、スポットを通したコミュニティの形成に役立てることができる。一方、目当てのスポットがなければ、新たにスポットを設置することも、全くスポットを指定しないこともできる。

2.1.3 投稿記事の記入

スポットを選択すると、次は投稿フォーム画面 (図 2.6) が表れる。

ここには、Weblog システムに投稿する記事そのものの情報であるタイトル、本文を記入する。

また、「ジャンル」「評価」という本研究が用意したメタ情報を選択する。

ジャンルはその記事の内容を表すカテゴリで、日記・グルメ・ニュース・レジャー・ショッピングから選択する。Weblog の方にも、記事を内容で分類する「カテゴリ」があるが、「ジャンル」は本システムが内部で情報を分類するためのメタ情報であり、Weblog の「カテゴリ」とは独立したものである。

「評価」はその記事が示す対象や場所に与える評価値を表し、5 段階評価、または「評価なし」を選択する。

2.1.4 投稿する Weblog の選択

すべての情報の入力を終えると、確認画面 (図 2.7) が表れる。

ここではこれまで書いた記事の確認と、投稿する Weblog 記事の選択が行える。

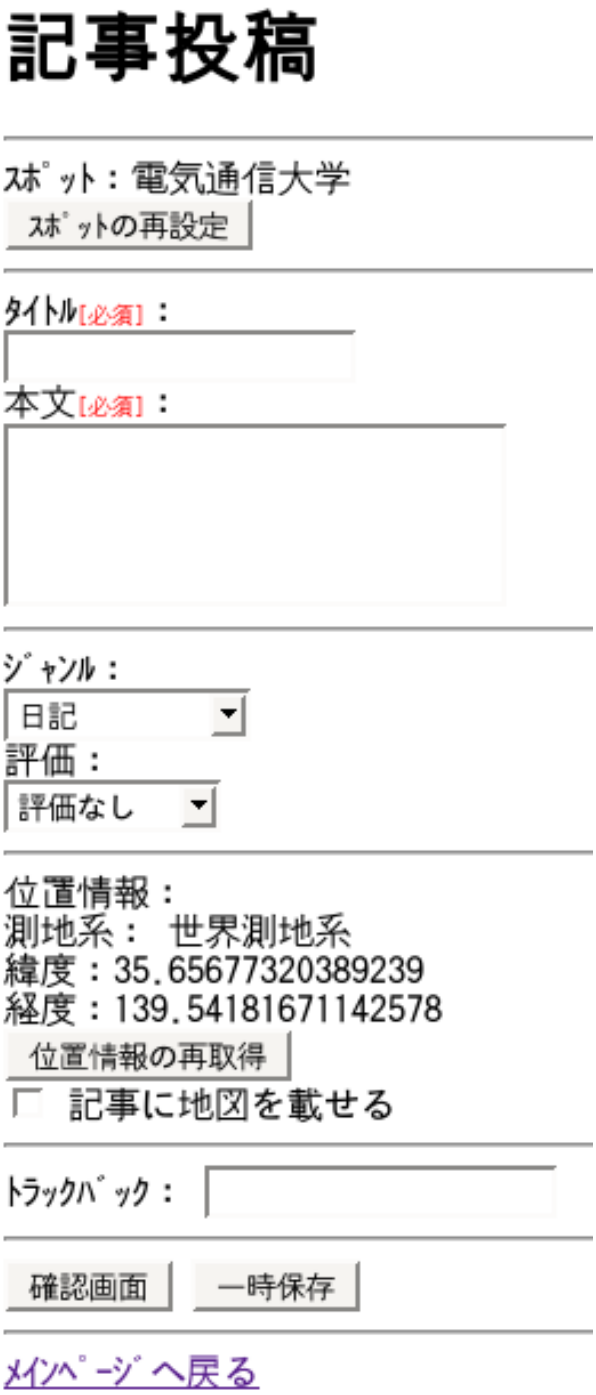

図 2.6: 投稿フォーム画面

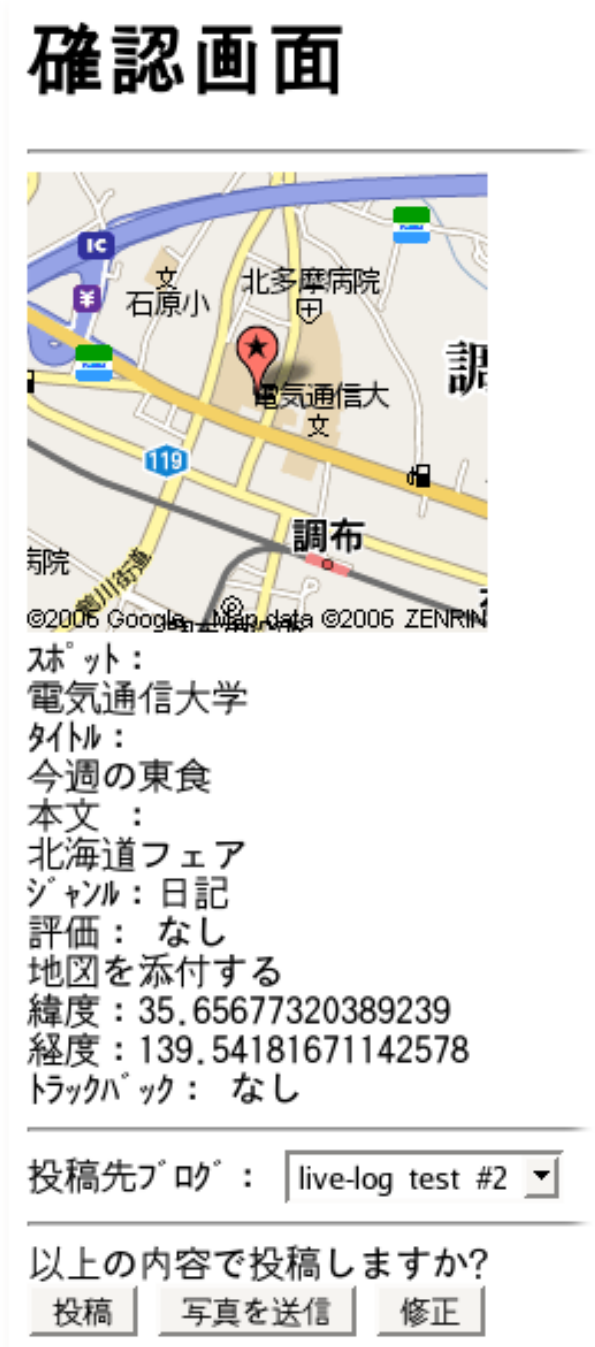
確認画面
IC
石原小
北多摩病院
電気通信大
119
調布
©2006 Google
©2006 ZENRIN
ｽﾎﾟｯﾄ：
電気通信大学
ﾀｲﾄﾙ：
今週の東食
本文 ：
北海道フェア
ｼﾞｬﾝﾙ：日記
評価： なし
地図を添付する
緯度：35.65677320389239
経度：139.54181671142578
ﾄﾗｯｸﾊﾞｯｸ： なし
投稿先ﾌﾞﾛｸﾞ ：
live-log test #2
以上の内容で投稿しますか?
投稿
写真を送信
修正

図 2.7: 確認画面

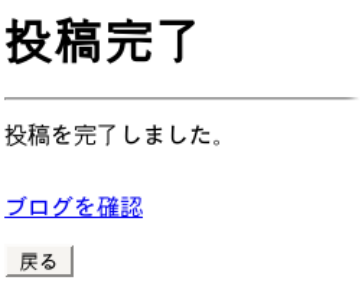

図 2.8: 投稿完了画面

本システムでは、ひとりのユーザに対し複数の Weblog を登録することができる。

この投稿先の Weblog を選択し、「送信」ボタンを押下すると、本システムが投稿処理を開始する。

2.1.5 投稿

そしてシステムが正しく処理を終えると、投稿完了画面が表示される。

2.1.6 写真付き Weblog の投稿

写真の付いた Weblog を投稿する場合は、画像ファイルだけ別途メールで添付して送信する。

確認画面で「写真の送信」ボタンを押下すると、メール送信画面 (図 2.9) が表示される。ここで表示されている「送信」リンクを選択すると、携帯電話のメーラが起動する。

このメーラに表示されるメールには本システムに必要な情報がすでに記述されている。メールを編集することなく、このまま画像ファイルを添付し、送信する。

本システムは、そのメールが届いた時点で投稿処理を開始する。

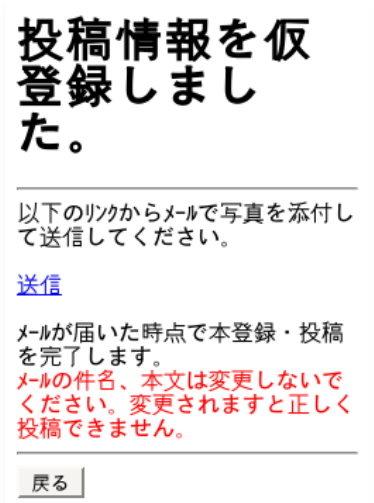

図 2.9: メール送信画面

2.2 位置情報が付加された Weblog の検索

この節では本システムのもうひとつのメイン機能である、検索について述べる。

本システムで投稿した記事は、本システムで検索を行える。その結果の提示方法として、本システムは投稿先 Weblog 記事へのポータルサイトを構築する。具体的には GPS で取得したユーザの現在地から、周囲 2km 以内に書かれた Weblog 記事へのリンクを一覧表示する。

図 2.10 に、ポータルサイトの画面を示す。表示された一覧は、その並び順として近距離順、新着順、高評価順が指定できる。また、ジャンルに絞ってフィルタリングするジャンル検索 (図 2.11) や、スポットに属した記事だけを提示するスポット検索 (図 2.12) が行える。記事投稿時に与えたさまざまなメタ情報は、このように情報のフィルタリングや並び順の要素として利用することができる。

本研究では提示方法を単に記事内容の表示ではなく、あえて各ユーザのもつ Weblog 記事へのリンクとして提供することで、「位置から Weblog に訪れる」という新しいコミュニケーション形成手段が生まれると考える。また、ただ並べるとい

全ｼﾞｬﾝﾙ

－ ズーム ＋

距離順 評価順 新着順

(A)：82m [ﾅﾋﾞ]
2006/12/14 22:34
[ﾚｼﾞｬ] うなずき人形…
★★★★☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

(B)：115m [ﾅﾋﾞ]
2007/01/28 21:11
[日] 多田研
★★★★★
投稿者：testuser
この記事にﾄﾗｯｸﾊﾞｯｸ

(C)：115m [ﾅﾋﾞ]
2006/12/14 18:31
[ﾆｭ] ベータテスト投稿テスト
投稿者：testuser
この記事にﾄﾗｯｸﾊﾞｯｸ

(D)：116m [ﾅﾋﾞ]
2007/01/29 15:48
[日] 電気通信大学
★☆☆☆☆
投稿者：testuser
この記事にﾄﾗｯｸﾊﾞｯｸ

(E)：208m [ﾅﾋﾞ]
2007/01/23 04:48
[ｸﾞﾙ] 電通マック
★★★☆☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

1 2 3 4

ジャンル検索
[日] 日記
[ﾆｭ] ﾆｭｰｽ
[ｸﾞﾙ] ｸﾞﾙﾒ
[ﾚｼﾞｬ] ﾚｼﾞｬｰ
[ｼｮ] ｼｮｯﾋﾟﾝｸﾞ

ｽﾎﾟｯﾄ検索

現在地をお気に入り登録

図 2.10: ポータルサイト画面

[ｸﾞﾙ]ｸﾞﾙﾒ

－ ズーム ＋

距離順 評価順 新着順

(A)：208m [ﾅﾋﾞ]
2007/01/23 04:48
[ｸﾞﾙ] 電通マック
★★★☆☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

(B)：366m [ﾅﾋﾞ]
2007/01/22 20:00
[ｸﾞﾙ] らんちたいむ
★★★★★
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

(C)：395m [ﾅﾋﾞ]
2007/01/22 20:04
[ｸﾞﾙ] たつみ
★★★☆☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

(D)：423m [ﾅﾋﾞ]
2007/01/22 20:05
[ｸﾞﾙ] 中華おはら
★★★★☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

(E)：569m [ﾅﾋﾞ]
2006/12/15 01:25
[ｸﾞﾙ] 日高屋
★★★★☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

ジャンル検索
[日] 日記
[ﾆｭ] ﾆｭｰｽ
[ｸﾞﾙ] ｸﾞﾙﾒ
[ﾚｼﾞｬ] ﾚｼﾞｬｰ
[ｼｮ] ｼｮｯﾋﾟﾝｸﾞ

ｽﾎﾟｯﾄ検索

図 2.11: ジャンル検索

電気通信大学

－ ズーム ＋

ｽﾎﾟｯﾄの詳細を見る
位置確認(ﾅﾋﾞ)

新着順 評価順

2007/01/29 15:48
[日] 電気通信大学
★☆☆☆☆
投稿者：testuser
この記事にﾄﾗｯｸﾊﾞｯｸ

2007/01/28 21:11
[日] 多田研
★★★★★
投稿者：testuser
この記事にﾄﾗｯｸﾊﾞｯｸ

2007/01/27 09:28
[日] あー…
投稿者：硬質化セラミック
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

2006/12/14 22:34
[ﾚｼﾞｬ] うなずき人形…
★★★★☆
投稿者：mae
この記事にﾄﾗｯｸﾊﾞｯｸ
[編集] [削除]

2006/12/14 18:31
[ﾆｭ] ベータテスト投稿テスト
投稿者：testuser
この記事にﾄﾗｯｸﾊﾞｯｸ

このｽﾎﾟｯﾄをお気に入り登録

▲ﾄｯﾌﾟ
周辺ｽﾎﾟｯﾄ一覧へ戻る
周辺情報ﾎﾟｰﾀﾙﾄｯﾌﾟへ戻る
ﾒｲﾝに戻る

図 2.12: スポット検索

うだけでなく、位置情報やスポット、ジャンル、評価などのメタ情報を有効的に用い、価値のある記事へのさらなる誘導性をはかっている。

2.3 その他の機能

本システムは投稿や検索以外にも、以下に挙げる様々な機能を提供している。

記事の編集・削除 本システムで投稿した記事を編集・削除できる。

メインメニューから「投稿履歴」ページを開くと、自分の書いた記事の一覧が表示され、そこから編集・削除を選択できる。また、周辺情報ポータルサイトに自分の書いた記事が含まれていた場合はその記事に「編集」「削除」リンクが表示され、そこからでも編集・削除操作を始めることができる。。

記事の一時保存 記事の記述中、投稿フォームの「一時保存」ボタンを押下すると、それまで入力していた記事情報を一時保存することができる。

本システムからの投稿はユーザの位置情報を利用するため、その場で記事を記入しなければ正しい位置情報を付加することはできない。しかし、実際のモバイル環境ではゆっくりと記事を記入できないことが多い。この機能を用いれば、先に位置情報だけ取得して、記事はあとでゆっくり書く、といった利用方法が可能である。

一時保存していた記事の再開はメインメニューの「未投稿記事」から行える。

お気に入り 位置、またはスポットをお気に入り登録すると、今後わざわざその場所に行かなくとも、いつでもその記事情報を閲覧することができる。

お気に入り一覧では登録した、位置・スポットの新着記事が表示されており、気になる地域の新鮮な話題をチェックすることができる。

トラックバック 投稿記事を指定の記事にトラックバックすることができる。

図 2.13: お気に入り一覧画面

図 2.14: ポータルマップ画面

トラックバックとは Weblog の特徴的な機能のひとつで、指定の Weblog 記事へリンクを張ったとき、その Weblog のユーザにリンクを張ったことを通知する仕組みである。また、その通知のことをトラックバック Ping と呼び、通知を送信する先をトラックバック Ping URL と呼ぶ。

本システムでは投稿フォームのトラックバック欄にトラックバック URL を直接記入するか、ポータルサイトからトラックバックしたい記事を選択することで、相手のユーザに自分の投稿記事を通知できる。

ポータルマップ　携帯電話からでは利用できないが、PC での検索用に、ドラッガブルな地図インタフェースによる、ポータルマップも提供する (図 2.14)。地図上には記事の示す位置にアイコンが配置されている。このアイコンをクリックするとウィンドウが開き、さらに情報を引き出すことができる。この情報の中に投稿先

Weblog 記事へのリンクが含まれる。

第 3 章

システムの設計

本研究では本システムを Web ブラウザからインターネットを介して利用可能な、Web アプリケーションとして設計した。本章ではまず本システムの構成イメージを示す。続いて、本システムの主な処理である投稿と検索において、本システムがどう処理を行うかについて述べる。そして最後に、それらを実現するための内部

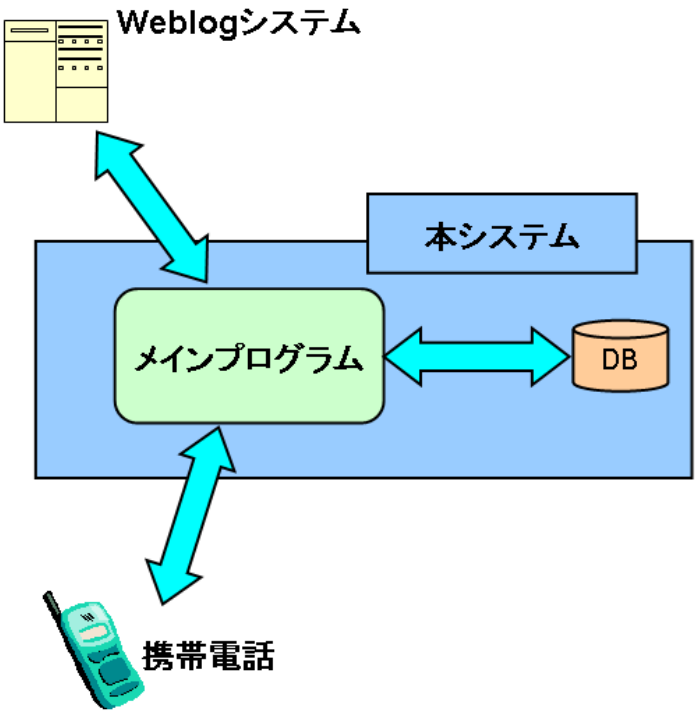

図 3.1: システム構成

構成の設計を論ずる。

本システムの構成を図 3.1 に示す。

携帯電話は GPS と Web ブラウザを搭載したもので、特にキャリア・機種に依存しない。Weblog システムは汎用の Weblog システム・サービスとする。本システムはこれら携帯電話と Weblog システムの間に配置され、そこで情報を仲介する。

さらに、本システムは内部にデータベースをもち、ここで投稿された記事情報を管理する。そして、このデータベースこそが位置と情報の結び付けを実現する部分である。

以下では、本システムのメインである投稿と検索の処理の流れを示す。

3.1 投稿処理の流れ

投稿時の処理の流れを図 3.2 に示す。

投稿時、本システムは携帯電話から Weblog システムへの投稿要求を処理しながら、データベースにて位置と情報を対応付けていく。

以下に投稿処理の流れを述べる。

1. 携帯電話より投稿記事と位置情報を受け取る
2. 記事の本文に位置情報を加え Weblog システムに投稿する
3. レスポンスとして投稿した記事に関する情報を受け取る
4. これら記事情報と位置情報を合わせて、データベースに保管する

3.2 検索処理の流れ

検索時の処理の流れを図 3.3 に示す。

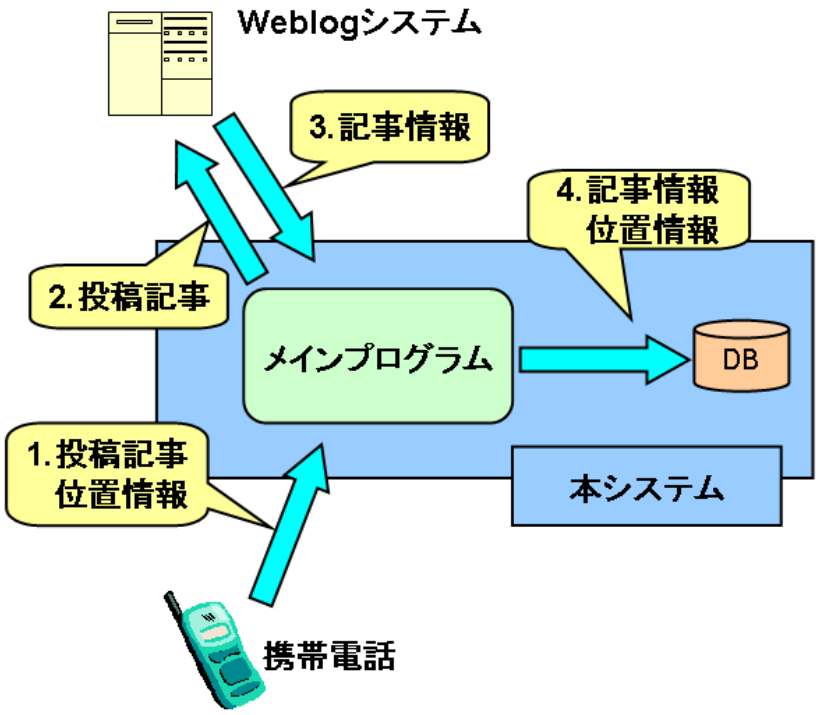
Weblogシステム
3.記事情報
4.記事情報
位置情報
2.投稿記事
メインプログラム
DB
1.投稿記事
位置情報
本システム
携帯電話

図 3.2: 投稿処理の流れ

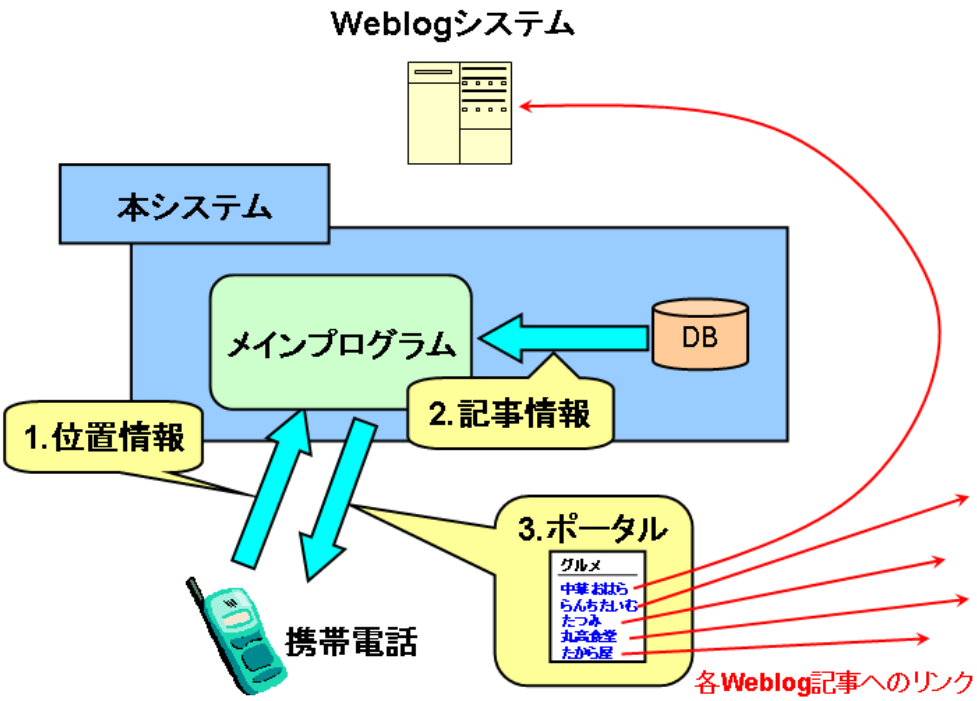
Weblogシステム
本システム
メインプログラム
DB
2.記事情報
1.位置情報
3.ポータル
グルメ
中華 おはら
らんちたいむ
たつみ
丸高食堂
たから屋
携帯電話
各Weblog記事へのリンク

図 3.3: 検索処理の流れ

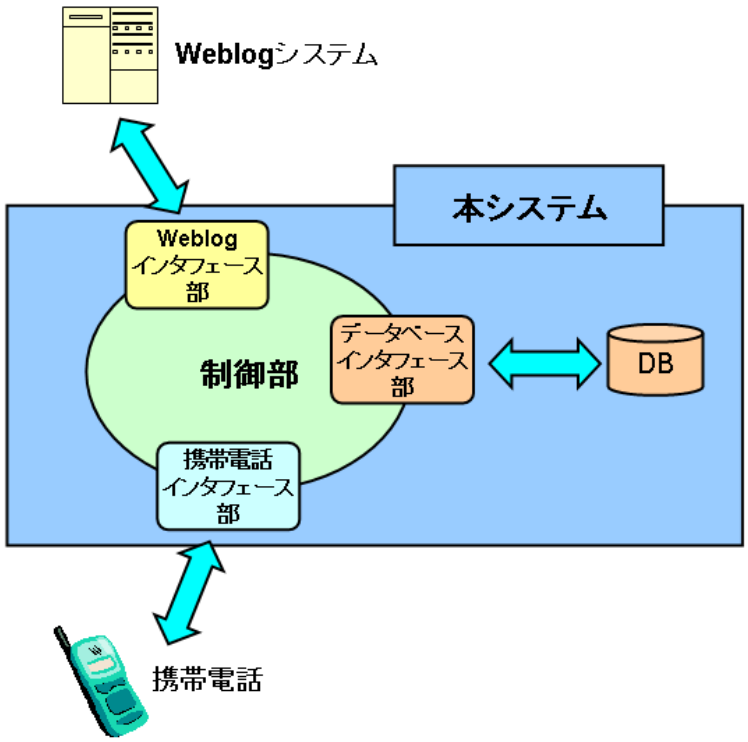

図 3.4: システム内部の基本構造

投稿処理によりデータベースに蓄積された記事情報を、位置情報をキーとして参照する。

以下に検索処理の流れを述べる。

1. 携帯電話から位置情報を受け取る

2. 渡された位置情報を基にデータベースを参照する

3. ヒットした記事情報からポータルサイトを構築する

こうして表示されたポータルサイトから気になる記事へのリンクを選択すると、投稿先の Weblog 記事サイトを開くことができるようにする。

3.3 内部構造

本システムの内部構造をモデル化した図を図 3.4 に示す。本システムは携帯電話、Weblog システム、データベースに対するそれぞれのインタフェース部と、それら 3 者間でやり取りされる情報を制御する制御部で構成される。

以下にはまず、制御部を主体として見た場合の各インタフェース部の役割を示す。

- 携帯電話インタフェース部

  入力： ユーザからの操作命令や情報を CGI パラメータとして受信

  出力： 携帯電話でも操作可能なユーザインタフェースを提供

- Weblog インタフェース部

  入力： 送信したリクエストに応じて、Weblog システムが返すレスポンスの受信とそこからの情報の抽出

  出力： Weblog システムへのリクエストの生成と送信

- データベースインタフェース部

  入力： データベースの情報をプログラムで扱える情報に変換

  出力： プログラムで扱われていた情報をデータベースに保管

各インタフェース部は、それぞれの先にある対象と制御部との間でやり取りされる情報の翻訳を担い、本システムと各相手との通信を実現する。また、このインタフェース部で携帯電話のキャリア・機種の違いや Weblog システムの違い、データベースを扱うデータベース管理システム (DBMS) の違いを吸収し、汎用性を高める。

続いて、制御部の役割を以下に示す。

- ユーザからのイベント処理

- 状態遷移の制御
- 携帯電話・Weblog システム・データベース間の情報流通制御

制御部は本システムの機能を実現する部分である。各インタフェース部を通して携帯電話、Weblog システム、データベースと対話しながらその状態に則した役割を遂行していくことで、第 4 章で示す本システムの機能を実現する。

第 4 章

システムの実装

本研究は以上の設計を基に、本システムを実装した。本章ではまず各インタフェース部の実装について論じ、続いて、本システムがもつ各機能の実現について議論する。

なお、以下に本システムの実装環境を示す。

- OS：Linux (Fedora Core 5)
- Web サーバ：Lighttpd 1.4.13
- DBMS：PostgreSQL 8.1.4
- 言語：Ruby 1.8.5(フレームワークとして Ruby on Rails 1.1.6)

また、動作確認を行った携帯電話と Weblog システムを以下に示す。

- 携帯電話： au W32H
- Weblog システム： livedoor Blog、So-net blog、Movable Type 3.3

4.1 携帯電話インタフェース部

携帯電話インタフェース部では、状態遷移に合わせて XHTML Basic[1] による動的コンテンツを構築することでユーザインタフェースを提供する。XHTML Basic は携帯端末向けに用意された XHTML のサブセットであり、現在主流の携帯電話の

Webブラウザ (WAP2.0ブラウザ) ならばほぼキャリア依存なく閲覧できる。ユーザからの操作や情報はCGIパラメータとして受け取り、それらは制御部が制御する。

ただし、写真付きの記事を投稿するときは、画像ファイルの転送にメールを用いる。現在の携帯電話に標準搭載の Web ブラウザでは、ファイルをアップロードする<input type="file" 〜 >タグを利用できるものがほとんどない。このため、Web ブラウザを利用したファイルの転送はできない。 その対策として、<a href="mailto:〜">タグによりメーラを呼び出し、画像ファイルをメールで添付・送信してもらうこととした。

4.1.1 位置情報の取得

GPS も XHTML Basic の記述により、ブラウザと連携して起動させることができる。GPS で得られた位置情報は CGI パラメータとして本システムに渡される。XHTML Basic に関する上の説明で、“ほぼ” キャリア依存しないと記述したのは、キャリアごとに XHTML Basic の独自拡張を設けているためであり、この位置情報取得の仕様もキャリア拡張のひとつであり、それぞれキャリアに依存する。携帯電話インタフェース部ではそのキャリアごとの仕様の違いを吸収し、汎用性を上げている。

本システムは現在主要の 3 キャリアである au、DoCoMo、SoftBank に対応した。各キャリアの位置情報取得の仕様は、どれも独自の URL から GPS を呼び出すことができ、指示した URL にパラメータが送信される仕組みになっている。また、そのとき設定する送信パラメータによって、位置情報の取得方法や、受信パラメータなどを指示できる。本システムではこの呼出し URL を<a>タグで記述する。

各キャリアの位置情報取得 URL を以下に示す。

au:<a href="device:gpsone?url=(system_url)&datum=0&unit=0&acry=0&number=0&ver=1">

DoCoMo：<a href="(system_url)" lcs>

SoftBank：<a href="location:auto?url=(system_url)" >

これらのリンクが選択されると “(system_url)” の部分に指定する URL に位置情報が送信される。つまり、ここに本システムの位置情報取得 CGI の URL を記述する。

また、受信するパラメータもキャリアによって異なる。送信されるパラメータの数や種類もだが、同じ内容を示す情報でもパラメータ名や値の表現方法が異なっている。

特に本システムが必要とする情報は測地系、緯度・経度、緯度・経度の値を表す単位である。

測地系とは緯度・経度・標高を表すために基準とする座標系のことで、日本では主に世界測地系と日本測地系が用いられている。

緯度・経度の表記方法は度単位で表すものと度分秒単位で表すものがある。たとえば同じ値でも度単位では “139.54484722”、度分秒単位では “+139.32.41.45” と表記される。特に度分秒単位では “E139.32.41.45” や “139/32/41.450” など様々な表記方法がある。

表 4.1 に、主要 3 キャリアによる位置情報パラメータの違いを示す。

各キャリアによる位置情報取得仕様の吸収

携帯電話インタフェース部では、以上に述べた各キャリアによる位置情報取得の仕様の違いを吸収する。

まず必要なのがキャリアの判別である。これができなければ、どのように位置情報取得機能を利用し、どのような情報が得られるかが分からない。キャリアの判別にはユーザエージェントの文字列をチェックする。ユーザエージェントは au ならば “KDDI”、DoCoMo ならば “DoCoMo”、SoftBank ならば “SoftBank” もしくは “Vodafone” で始まる。そして、そのキャリアに合わせて位置情報取得リンク、つまり、<a>タグの記述を変える。

続いて、受信パラメータの違いの吸収について述べる。

表 4.1: 各キャリアによる位置情報パラメータの比較

キャリア	パラメータ名	値
測地系		
au	`datum`	0：世界測地系　1：日本測地系
DoCoMo	`geo`	wgs84：世界測地系
SoftBank	`geo`	wgs84：世界測地系　tokyo：日本測地系　itrf：ITRF 系
表記単位		
au	`unit`	0：度分秒　1：度
DoCoMo	(なし)	(度分秒単位のみ)
SoftBank	(なし)	(度分秒単位のみ)
緯度・経度		
au	`lat`	unit=0：±dd.mm.ss.ss　unit=1：±dd.ddddd
	`lon`	unit=0：±ddd.mm.ss.ss　unit=1：±ddd.ddddd
DoCoMo	`lat`	±dd.mm.ss.sss
	`lon`	±ddd.mm.ss.sss
SoftBank	`pos`	[N|S]dd.mm.ss.ss[W|E]ddd.mm.ss.ss

本システムで最も重要なものが緯度・経度の値である。本研究では緯度・経度の値に影響を与える測地系と表記単位を統一し、本システム内で同等の値として処理できるようにする。

測地系は世界測地系を基本とする。世界測地系は日本測地系よりも地球規模で適合した国際的な測地基準系であり、日本でも測量法の改正により、こちらを用いるようになった。このため本システムでも世界測地系を基本とし、取得した値が日本測地系ならば世界測定系へ変換する。ただし、地図サービスや Geocoder など他の Web サービスを利用する際に、慣習的に日本測地系を採用しているサービスもあるため、必要があれば日本測地系へ変換する。

緯度・経度の表記単位は度単位を基本とする。これは、2 点間の距離を緯度・経度から計算する際、小数点で扱う方が容易だからである。このため、度分秒単位で受け取ったパラメータは度単位に変換する。ただし、これも同じく他の Web サービスによって表記方法が異なるため、その都度その Web サービスに合った表記に変換する。

4.1.2 他の投稿方式との比較

本研究ではユーザインタフェースに Web ブラウザ方式を採用したが、他の moblog サービスでは、メール方式やアプリケーション方式 [2, 3] を採っているものもある。メール方式はシステムが発行したメールアドレスに対し、件名に記事タイトル、本文に記事内容を記述し送信する方式である。アプリケーション方式は携帯電話にアプリケーションをインストールし、それを使って記事を投稿する方式である。

それぞれの方式には利点・欠点がある。

メール方式はキャリア・機種に依存せず、普段使い慣れている操作で投稿できるが、対話性に乏しく、投稿された記事の編集などの複雑な操作ができない。

アプリケーション方式は自由なインタフェースと高い機能を実現でき、さらに、携帯電話に備わった様々な機能や資源と連携することができる。しかし、各キャリ

アによって採用しているアプリケーション実行環境が異なるため、大きくキャリア依存してしまう。また、わずかな更新でもその都度アプリケーションのダウンロードが必要なため、ユーザの負担が大きく、細かな修正に即座に対応できない。

Web ブラウザ方式の欠点は連携利用できる端末機能が少ない、通信コストが高い、といった点が挙げられる。しかし、Web ブラウザは GPS やメーラと連携可能であり、メーラはカメラ、ファイルシステムと連携可能であるため、本システムに必要な資源は全て利用できる。通信コストに関しては、今後はパケット料金の定額制が主流になると見越し、特に問題視しない。そして、本研究では対話性と汎用性の両方を兼ね備えた Web ブラウザ方式を採用した。

4.2 Weblog インタフェース部

本システムでは、Atom Publishing Protocol[4] (AtomPP と略されるが、本論文では広く定着している AtomAPI と称す) を用いて、Weblog システムに記事の投稿・編集等の処理を要求する。AtomAPI は Weblog や Wiki などの Web 上のコンテンツを外部から操作できるアプリケーションレベルの通信プロトコルである。

他のプロトコルに XML-RPC というものもあるが、XML-RPC では Weblog システムごとに API が異なり仕様が統一されてないことや、パスワードが平文で流れてしまうことなどを考慮し、本研究では AtomAPI を採用した。

AtomAPI に対応している Weblog はいくつかあるが、本システムで動作を確認した Weblog システムは So-net blog、livedoor Blog、Movable Type 3.3 である。

4.2.1 AtomAPI による Weblog 間通信の実装

AtomAPI では URI で表したリソースに対し、各操作に応じた HTTP メソッドを送信する、REST アーキテクチャである。

Weblog に送信するリクエストは以下の 4 つの要素を組み合わせて生成する。

HTTP メソッド： AtomAPIでは要求する操作ごとにHTTPメソッドが割り当てられている。

URI： AtomAPIでは、各操作に対するリクエスト送信先URIを、その操作の名前(PostURI、EditURIなど)で表現し、これらのURIをサービスURIを呼ぶ。

認証： AtomAPIでは、毎回リクエストを送信するごとに、WSSE認証[5]による認証情報を必要とし、これをリクエストヘッダに添付して送信する。

データ： AtomAPIでやり取りされるデータはXML形式の文書であり、これをAtom Syndication Format(以下、Atomフォーマット)と呼ぶ。

以下ではこれら4つの要素について、詳細を述べる。

4.2.2 HTTP メソッドと URI

以下にAtomAPIで利用するHTTPメソッドと、サービスURIを示す。

HTTP メソッド

GET： リソースの取得

POST： リソースの作成

PUT： リソースの更新

DELETE： リソースの削除

サービス URI

FeedURI： Weblog記事の参照用URI

PostURI： Weblog記事の投稿用URI

EditURI： Weblog 記事の編集・削除用 URI

UploadURI： ファイルのアップロード用 URI

たとえば、FeedURI に対し HTTP GET リクエストを送信すると、その Weblog にある記事が参照できる。

サービス URI はユーザの管理する Weblog サイトごとに割り当てられる URI である。特に EditURI は Weblog サイトにある各「記事」に割り当てられる。

認証

AtomAPI ではリクエストを送信するたび、ユーザ名とパスワードを基にした認証情報を要求する。この認証方式として、現在ほとんどの AtomAPI 対応 Weblog システムでは WSSE 認証を採用している

WSSE 認証では、パスワード、ランダム文字列、作成日時を基にパスワードダイジェストを生成し、HTTP ヘッダに挿入する。パスワードダイジェスト作成に必要な要素を以下に定義する。

`Nonce`： ランダムな文字列。リクエストごとに生成する

`Created`： `Nonce` を作成した日時。ISO-8601 形式で表す

`Password`： Weblog システムで用いるユーザのパスワード

これらを順につないだ文字列を SHA1 でハッシュを生成し、さらにこれを Base64 でエンコードしたものを`PasswordDigest` とする。この作業を Ruby コードで示すと以下のようになる。ここで、`Base64::encode64()` は Base64 エンコードを行う関数、`Digest::SHA1.digest()` は SHA1 でハッシュ値を出す関数である。

```
PasswordDigest = Base64::encode64(Digest::SHA1.digest( Nonce + Created + Password ))
```

そして、これらの要素とユーザ名 (`Username`) を用い、以下の文を HTTP ヘッダに追加する。

```
X-WSSE: UsernameToken Username="(Username)" PasswordDigest="(PasswordDigest)"
Created="(Created)" Nonce="(Base64::encode64(Nonce))"
```

このようにして AtomAPI ではパスワードを暗号化し、セキュアな Weblog 間通信を可能にしている。

データ

AtomAPI でやり取りされるデータは Atom フォーマットと呼ばれる XML 形式の文書である。

以下に例として、投稿・編集リクエスト時の Atom フォーマットを示す。Atom フォーマットでは様々なタグ要素が定義されているが、投稿・編集時では以下のようなシンプルな構造だけ渡せば良い。

```
<?xml version="1.0"?>
<entry xmlns="http://purl.org/atom/ns#" xmlns:dc="http://purl.org/dc/el
ements/1.1/">
    <title type="text/html" mode="escaped">タイトル</title>
    <issued>2007-01-15T01:17:32</issued>
    <content type="text/html" mode="escaped" xml:lang="ja-JP">本文</content>
</entry>
```

この<entry>要素が 1 つの記事ということを表し、これを Atom エントリドキュメントと呼ぶ。ここに含まれる情報は、記事タイトルを表す<title>、投稿時間を表す<issued>、記事本文を表す<content>だけである (さらに、<issued>は省略もできる)。

本研究が情報媒体として Weblog を選んだ理由のひとつが、この少ない情報要素だけで Web コンテンツを構築できる点であり、複雑な編集作業のできない携帯電話に適した情報媒体であるといえる。

エンドポイント URI

ここで、これまでの説明からシステムだけでは用意できない情報、つまり、ユーザに要求する情報に関してまとめると以下のようになる。

- URI：サービス URI
- 認証：Weblog システムで使用しているユーザ名とパスワード
- データ：タイトル、本文

ユーザ名とパスワードはユーザが知り得る情報であり、タイトルと本文は投稿や編集のたびにユーザが入力する情報である。そして、サービス URI もユーザごとに割り当てられている URI であり、ユーザが管理せねばならない。つまり、AtomAPI を利用したいユーザはまずこれらサービス URI を知らなければならない。

そこで、ユーザに各サービス URI を通知するための仕組みとして、エンドポイント URI というものが用意されている。これは各 Weblog システム・サービスで管理するものであり、公開されている。この公開 URI に HTTP GET リクエストを送信すると、そのユーザの FeedURI、PostURI、UploadURI を知ることができる。なお、EditURI はその対象リソースが Weblog「記事」であるため、ここでは得られない。

レスポンスの解析

以上のようにリクエストを生成し、Weblog システムに送信すると、Weblog システムはその処理の結果をレスポンスとして返してくる。

このとき返信されるデータも、やはり Atom フォーマットである。そして、このデータの中には本システムが知りたい情報が多く記載されている。

Weblog インタフェース部では、DOM(Document Object Model) という XML 文書をアプリケーションから利用する API を用いて、Atom フォーマットに記載されている情報を抽出する。

そして、抽出したデータを制御部に渡していく。

4.3 データベース

以下では本システムのデータベースインタフェースとデータベースで管理する情報について述べる。

4.3.1 データベースインタフェース

データベースに保管されている情報をプログラミング内で操作できるようにするために、本システムのフレームワーク、Ruby on Rails に組み込まれている ActiveRecord を用いた。ActiveRecord は同名のデザインパターン、アクティブレコードを実装した ORM(Object / Relational Mapping) ライブラリである。これを用いることで、容易かつ効率的にリレーショナルデータベース上の情報を Ruby オブジェクトとして扱うことができる。

なお、この ActiveRecord では、データベースにあるテーブルに行を新規作成した場合、その行の “id” カラムに ID 番号が自動的に振られる。これは連続した整数であり、行を新規作成するたびに 1 ずつ増えた値が割り当てられる。本システムではこの ID を利用して各テーブル間の関係を表現する。 以降の説明で “〜ID” と表記した場合、特に断らない限りは “〜の情報を識別する ID 番号” のことを表す。

4.3.2 データベースで管理する情報

以下では本データベースが管理する情報について述べる。ただし、ここでは各情報の概観を示すだけに止め、実際にどのような項目を保管するかは節 4.4 の各機能の実装の中で示す。

ユーザ情報

ユーザ情報はさらに、ユーザの個別の情報を表すユーザ設定情報と、ユーザがもつ Weblog についての情報を表す Weblog 設定情報に分かれる。

ユーザ設定情報にはユーザ名、パスワードを含む。本システムはユーザ登録制を採り、ログインによってそのユーザと、ユーザのもつ Weblog を識別する。

Weblog 設定情報は、設定したユーザの ID、Weblog の名前、Weblog の URL、そして AtomAPI を使用するのに必要な (Weblog システムの) ユーザ名とパスワード、エンドポイント URI、FeedURI、PostURI、UploadURI である。

なお、ここに保管するパスワードは全て暗号化する。ただし、本システムで利用した PostgreSQL では 自動で暗号化・復号化する方法をもっていなかったため、システム側でその処理を行うことにした。

記事情報

記事情報は投稿された記事を表現する、本システムの最も核となる情報である。投稿された記事に関する多くの情報を管理し、ここで Weblog 記事と位置情報を対応付ける。

主な情報は Weblog 記事に関する情報 (投稿者名、投稿日時・タイトル・本文・投稿先 URL) や本システムが独自に用いる各種メタ情報 (ユーザ ID、スポット ID、ジャンル、評価、地図の添付)、そして緯度・経度である。

写真付き記事が投稿された場合は、さらにその写真画像のアップロード先 URL と、本システム内に保存したファイル名も保管する。

本文はポータルサイト構築のためには保管しておく必要ないが、編集・削除やトラックバックなどの処理のため手元に保管しておいた方が都合が良い。

一時保存記事情報

記事を一時的に保存する必要があるときは、記事情報を保管するテーブルとは別のテーブルに保管する。

このテーブルは写真付き記事を投稿するときにも利用する。本システムでは写真付き記事の投稿は別途メールで送信する仕様を採っている。このとき、メールが届くまでに一時的に保存するスペースが必要であるため、このテーブルを利用する。

スポット情報

スポット情報は投稿時にユーザに設置されたスポットに関する情報である。主にスポット名、住所、電話番号、関連 HP の URL、説明文、緯度・経度、を管理する。

4.4 各機能の実装

本節では、まずユーザアカウントの登録とログインに関して述べる。これは全機能を利用するうえで必要な、ユーザの識別を実現する。それから、本研究の目的である位置と密接に結び付いた Weblog 記事の投稿と検索、そしてその他の機能の実装について論ずる。

4.4.1 ユーザアカウントの登録とログイン

ユーザアカウントはユーザと、ユーザのもつ Weblog システムに関する情報である。ここには本システムがユーザに代わって Weblog システムに記事を投稿するために必要な情報が含まれている。

ログインは、システムがユーザを認識し、そのユーザ固有の情報を取得できる状態にするための作業がである。

ユーザアカウントの登録

表 4.2: データベースに保管するユーザ設定情報の項目

項目	備考	取得元
ユーザ ID	システムがユーザを識別する ID	システム
ユーザ名 (システム)	本システムで用いるユーザ名	ユーザ
パスワード (システム)	本システムで用いるパスワード	ユーザ

表 4.3: データベースに保管する Weblog 設定情報の項目

項目	備考	取得元
Weblog 設定 ID	Weblog 設定情報を識別する ID	システム
ユーザ ID	設定したユーザの ID	システム
ユーザ名 (Weblog)	Weblog システムに用いるユーザ名	ユーザ
パスワード (Weblog)	Weblog システムに用いるパスワード	ユーザ
エンドポイント URI	Weblog システムごとに用意されている	ユーザ
FeedURI	Weblog 記事参照用 URI	Weblog システム
PostURI	Weblog 記事投稿用 URI	Weblog システム
UploadURI	ファイルアップロード用 URI	Weblog システム
Weblog の名前	Weblog サイトのメインタイトル	Weblog システム
Weblog の URL	Weblog サイトの URL	Weblog システム

本システムを初めて利用する際、まずユーザアカウントを作成しなければならない。

ユーザアカウントの作成のため、本研究はユーザ設定作成フォーム、Weblog 設定作成フォームを提供し、この順番に情報を受け取る。表 4.2 と表 4.3 に本システムが求める情報を示す。取得元が “ユーザ” となっている情報が、各作成フォームにおいてユーザに入力を要求する情報である。

ユーザ設定作成フォームでは本システムに必要なユーザ固有の情報を要求する。ここで要求する情報は、本システムへのログインに用いるユーザ名とパスワードである。

Weblog 設定作成フォームでは、システムがユーザの代わり Weblog システムを操作するうえで必要な情報を登録する。ここで要求する情報は、Weblog システムで用いるユーザ名とパスワード、そして Weblog システムで用意されたエンドポイント URI だけである。残りの情報は本システムが Weblog システムに問い合わせ、取得する。

この問い合わせには、エンドポイント URI を利用する。節 4.2.2 に示したように、エンドポイント URI に対し HTTP GET リクエストを送信すると各種サービス URI が取得できる。このときの WSSE 認証情報にログイン名とパスワードを利用する。

以下にそのリクエストに対する Weblog システムからのレスポンスの例を示す。

```
HTTP/1.1 200 OK
Date: Mon, 22 Jan 2007 08:20:42 GMT
Server: Apache/2.2.2 (Fedora)
Pragma: no-cache
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

879
```

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://purl.org/atom/ns#">

   <link xmlns="http://purl.org/atom/ns#" type="application/x.atom+xml"
rel="service.post" href="http://blog.uec.ac.jp/mt/mt-atom.cgi/weblog/blog_id=2"
title="live-log テスト用ブログ"/>  —(1)
     <link xmlns="http://purl.org/atom/ns#" type="application/x.atom+xml"
rel="service.feed" href="http://blog.uec.ac.jp/mt/mt-atom.cgi/weblog/blog_id=2"
title="live-log テスト用ブログ"/>  —(2)
     <link xmlns="http://purl.org/atom/ns#" type="application/x.atom+xml"
rel="service.upload" href="http://blog.uec.ac.jp/mt/mt-atom.cgi/weblog/blog_id=
2/svc=upload" title="live-log テスト用ブログ"/>  —(3)
     <link xmlns="http://purl.org/atom/ns#" type="text/html" rel="alternate" href=
"http://www.spa.is.uec.ac.jp/~maezawa/livelog/" title="live-log テスト用ブログ" />
  —(4)

</feed>

0
```

このレスポンスには 4 つの<link>タグが記述されている。ここで注目すべき箇所はこの<link>タグ内のrel 属性である。このrel 属性の値によって、href 属性の示す URI の意味が分かる。つまり、rel 属性に注目すれば、抽出すべき URI が分かる。

以下にそれぞれのrel 属性が表すhref 属性の意味について示す。

service.post： PostURI(1)

service.feed： FeedURI(2)

service.upload： UploadURI(3)

alternate： Weblog サイトの URL(4)

このレスポンスから本システムが取得する情報は FeedURI、PostURI、UploadURI の 3 つのサービス URI と Weblog 名、Weblog の URL である。Weblog の名前は<link>タグのtitle 属性から取得する。これらを DOM を用いて取得する。

これら 5 つの情報を正しく取得できれば、続いて、これらをデータベースに保管する。このときデータベースに保管する各情報の項目を表 4.2、表 4.3 に示す。

データベースへの保管の成功をもって、ユーザアカウントの登録処理を完了する。

ログイン

本システムではユーザが利用する前にログインを要求する。ログイン画面よりユーザ名とパスワードを受け付け、「ログイン」ボタンが押下されると認証処理を開始する。

システムはまずユーザ名でデータベースを参照し、一致するユーザ情報を得る。そしてユーザ情報にあるパスワードと入力されたパスワードが一致すればログイン成功とする。

ログインが成功すると、システムはユーザ情報にあるユーザ ID をセッションに保存する。これによって、このセッションが続く限りはいつでもユーザ ID を利用できる。各機能の実行中にユーザ個別の情報が必要になった場合は、このセッションに保存したユーザ ID を使ってデータベースを参照する。このユーザ ID がシステムがユーザを識別する鍵となる。

4.4.2 位置情報を付加した Weblog 記事の投稿

この節では本システムのメイン機能のひとつ、位置情報付き Weblog 記事の投稿処理について述べる。

以下に、節 3.1 で示した投稿処理の流れを再掲する。

1. 携帯電話より投稿記事と位置情報を受け取る

2. 記事の本文に位置情報を加え Weblog システムに投稿する

3. レスポンスとして投稿した記事に関する情報を受け取る

4. これら記事情報と位置情報を合わせて、データベースに保管する

上記の 1. と 3. では本システムが必要とする情報を取得する。この中にはデータベースに貯めるべき情報も含む。表 4.4 に、これから取得する情報の項目と、その情報がどこから得られるのかについて示す。

投稿フォームからの情報受け取り

投稿時におけるユーザとのやり取り (節 2.1) によって、本システムは以下の情報を得られる。

- 位置情報
- スポット ID(スポット情報は ID で取得する)
- タイトル
- 本文
- ジャンル
- 評価
- 記事に地図を載せるかどうか
- トラックバック Ping URL
- 投稿先の Weblog 設定 ID(投稿先 Weblog 設定情報も ID で取得する)

各ステップでこれらの情報を受け取ると、制御部は逐次セッションに保存していく。そして、必要なときはセッションから情報を参照するようにする。

表 4.4: 投稿時、データベースに保管する記事情報の項目

項目	備考	取得元
記事 ID	記事情報を識別する ID 番号 システム	
ユーザ ID	投稿者のユーザ ID	セッション
投稿者名	Weblog で使っているユーザの名前	Weblog システム
投稿日時	日本時間	Weblog システム
タイトル		ユーザ
本文	位置情報を付加する前の記事	ユーザ
緯度	世界測地系、度単位	ユーザ
経度	世界測地系、度単位	ユーザ
スポット ID	「指定なし」ならば 0	ユーザ
ジャンル		ユーザ
評価		ユーザ
地図添付	記事に地図を添付するかどうか	ユーザ
投稿先 URL	投稿した Weblog 記事の URL	Weblog システム
EditURI	AtomAPI で用いる編集用 URI	Weblog システム
Weblog 設定 ID	投稿した Weblog の設定 ID	ユーザ

セッションに管理させたのは、記事の記入途中で位置情報の再取得を行ったとき、パラメータで管理すると以下の問題が起きるからである。キャリア (SoftBank を除く) の仕様では、位置情報を取得する際に指示する URL に、任意のパラメータを記述することはできない。つまり、本システムに渡されるパラメータには GPS からの位置情報しか含まれておらず、これまで入力された情報は全て失われてしまう。しかし、これら情報をセッションに保持しておけば、位置情報を取得し次の画面に遷移しても、セッションが持続している限りは情報が失われない。

ところで、セッションを用いることにはさらにもうひとつの利点がある。それは、投稿完了時にこれら全ての情報をセッションから消去するようにすれば、二重投稿を防止できる点である。Web ブラウザの「戻る」から戻って、もう一度「投稿」ボタンを押下しても、投稿すべき情報はもうセッションにはない。

さて、ユーザがすべての情報の入力を終え、「投稿」ボタンを押下すると、本システムは投稿処理を開始する。本システムの制御部は、これまで集めた情報を Weblog システムやデータベースに流通させていく。

Weblog システムへの投稿

制御部はまず Weblog インタフェース部に必要な情報を渡し、Weblog システムへの記事の投稿を行う。

ただし、ユーザから取得した記事本文の情報は基本的な内容しか書かれていない。まずは、図 2.2 で示したような位置情報を記載するために、実際の投稿の前に記事の調整を行う。

付加する位置情報は、緯度・経度、住所、地図である。

住所は、緯度・経度から住所へ変換する Web サービス [6] を利用し、丁目までの住所を取得する。

「記事に地図を載せる」にチェックが付いていた場合は、本システムが設けた添付地図用 Web サイトに通じる`<iframe>`タグを記事に埋め込む。この Web サイト

は Google Maps API[7] を用いて構築した。

なお、この地図の添付を任意で選択できるようにしたのは、<iframe>に対応していない Weblog システムがあるからである。この Weblog システムに投稿すると、<iframe>タグの内容がそのまま表示されてしまう。

Weblog に載せる記事が完成したら、Weblog システムに送信する投稿リクエストを作成する。

AtomAPI による投稿は PostURI に対して HTTP PUT リクエストを送信することで実現する。そして、このリクエストヘッダには WSSE 認証情報を付加しなければならない。これらを生成するのに必要な PostURI・ユーザ名・パスワードは、ユーザから受け取った投稿先 Weblog 設定 ID を用いてデータベースを参照し、取得する。

そして、送信するデータとして Atom エントリドキュメントを添付する。このときの Atom エントリドキュメントを以下のように構成する。

```
<entry>
  <title> ユーザから取得したタイトル</title>
  <issued> 投稿日時</issued>
  <content> 調整した記事本文</content>
</entry>
```

そして、これらを組み合わせて投稿リクエストを生成し、Weblog システムに送信する。

以下では、実際に送られるリクエストを例として示す。

```
POST http://nagashima.spa.is.uec.ac.jp.private/mt/mt-atom.cgi/weblog/blog_
id=2  HTTP/1.1
Host: nagashima.spa.is.uec.ac.jp.private
X-WSSE: UsernameToken Username="mae", PasswordDigest="BuM7mMxbUvzc3BCr7VcO
R/yze2w=", Nonce="MmE1YmMyZWJhZDQ4Y2RhNjY1NjVmMzg0NQ==", Created="2007-01-
```

```
22T07:32:10Z"
Content-Type: text/html
Content-Length: 901

<?xml version="1.0"?>
<entry xmlns="http://purl.org/atom/ns#" xmlns:dc="http://purl.org/dc/eleme
nts/1.1/">
    <title type="text/html" mode="escaped">タイトルはここに</title>
    <issued>2007-01-22T07:32:10Z</issued>
    <content type="text/html" mode="escaped" xml:lang="ja-JP">本文はここから


=== 位置情報 ===
緯度：35.6567732038924
経度：139.541816711426
東京都調布市富士見町二丁目
&lt;P&gt;&lt;IFRAME src=&quot;http://thompson.spa.is.uec.ac.jp:10080/map/mi
nimap?lat=35.6567732038924&amp;lon=139.541816711426&amp;category=0&quot; w
idth=300 height=300 frameborder=0 scrolling=NO&gt;&lt;/IFRAME&gt;&lt;/P&gt;

&lt;a href=&quot;http://www.navitime.co.jp/map/?datum=0&amp;unit=1&amp;lat=
35.65677&amp;lon=139.54182&amp;fm=0&quot;&gt; 携帯電話用ナビサイト（NAVITIME）
&lt;/a&gt;&lt;a href=&quot;http://thompson.spa.is.uec.ac.jp:10080/main&quot
;&gt;live-log&lt;/a&gt;
</content>
</entry>
```

Weblog システムからのレスポンスの受信

Weblog システムは投稿リクエストの処理を行った後、その結果をレスポンスとして返してくる。

このレスポンスには、HTTP コードによる成功・不成功の情報と、Atom エントリドキュメントによる投稿処理した記事の情報が含まれている。

この Atom エントリドキュメントに含まれる情報は、その対象の Weblog システ

ムによって異なるが、主に以下のような情報が返信される。

```
<entry>
    <title> 記事のタイトル</title>
    <content> 記事の本文</content>
    <author>
        <name> (1) 投稿者名</name>
    </author>
    <link rel="alternate" href=" (2)Weblog 記事の投稿先 URL"/>
    <link rel="service.edit" href=" (3) この記事の編集用 URI(EditURI)"/>
    <issued> (4) 投稿日時</issued>
    <modified> 最終更新日時</modified>
    <id> 記事を識別する文字列</id>
</entry>
```

本システムは上記レスポンスから (1) 投稿者名、(2) 投稿先 URL、(3)EditURI、(4) 投稿日時を DOM を用いて抽出する。

この中でも特に重要なのが (2) 投稿先 URL と (3)EditURI である。

投稿先 URL はその記事が閲覧できるサイトへの URL であり、ポータルサイト構築のためには必要不可欠なものだからである。

EditURI はその記事ごとに個別で割り当てられた編集用の URI であり、記事の編集・削除機能で利用するからである。

そして、これらの情報抽出の成功により Weblog への投稿処理が完了すると、制御部はデータベースの処理に移行する。

なお、先のリクエストに対する Weblog システムのレスポンスを以下に示す。

```
HTTP/1.1 201 Created
Date: Mon, 22 Jan 2007 07:21:59 GMT
Server: Apache/2.2.2 (Fedora)
Pragma: no-cache
Location: http://nagashima.spa.is.uec.ac.jp.private/mt/mt-atom.cgi/weblog/blo
g_id=2/entry_id=75
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

<?xml version="1.0" encoding="utf-8"?>
<entry xmlns="http://purl.org/atom/ns#">
  <title xmlns="http://purl.org/atom/ns#">タイトルはここに</title>
  <summary xmlns="http://purl.org/atom/ns#">utf-8</summary>
  <content xmlns="http://purl.org/atom/ns#" mode="escaped" type="application/x
html+xml">本文はここから

=== 位置情報 ===
緯度：35.6567732038924
経度：139.541816711426
東京都調布市富士見町二丁目
&lt;P&gt;&lt;IFRAME src="http://thompson.spa.is.uec.ac.jp:10080/map/minimap?lat
=35.6567732038924&amp;lon=139.541816711426&amp;category=0" width=300 height=300
 frameborder=0 scrolling=NO&gt;&lt;/IFRAME&gt;&lt;/P&gt;

&lt;a href="http://www.navitime.co.jp/map/?datum=0&amp;unit=1&amp;lat=35.65677&
amp;lon=139.54182&amp;fm=0"&gt; 携帯電話用ナビサイト (NAVITIME)&lt;/a&gt;
&lt;a href="http://thompson.spa.is.uec.ac.jp:10080/main"&gt;live-log&lt;/a&gt;
</content>
  <author xmlns="http://purl.org/atom/ns#">
    <name xmlns="http://purl.org/atom/ns#">mae</name>

    <url xmlns="http://purl.org/atom/ns#"></url>
    <email xmlns="http://purl.org/atom/ns#">maezawa@spa.is.uec.ac.jp</email>
  </author>
  <issued xmlns="http://purl.org/atom/ns#">2007-01-22T16:21:59+09:00</issued>
```

```
  <link xmlns="http://purl.org/atom/ns#" type="text/html" rel="alternate" href="
http://www.spa.is.uec.ac.jp/~maezawa/livelog/2007/01/post_31.html"/>
  <link xmlns="http://purl.org/atom/ns#" type="application/x.atom+xml" rel="serv
ice.edit" href="http://nagashima.spa.is.uec.ac.jp.private/mt/mt-atom.cgi/weblog/
blog_id=2/entry_id=75" title="タイトル"/>
  <id xmlns="http://purl.org/atom/ns#">tag:www.spa.is.uec.ac.jp:post:75</id>

</entry>
```

データベースへの保管

これまでの処理により、本システムが求める情報は全て集まった。そして最後に、これらの情報をデータベースに保管する。

データベースに保管する記事情報の項目は表 4.4 にて示した。

記事の本文は、節 4.3.2 で前述したように、ポータルサイト構築のためには必要ないがデータベースに保管しておく。なお、ここで保管する本文は投稿フォームに記入されたもので、Weblog 記事用に位置情報を付加したものではない。

投稿日時は Weblog システムが返したものを保管する。Weblog システムによっては日時を指定した投稿ができないものがあるため、本システムの方を Weblog システムに合わせる。タイムゾーンは日本時間とする。Weblog システムによっては協定世界時 (UTC) で返すものもあるが、このときは日本時間に変換する。

EditURI と投稿先 Weblog 設定情報 ID は編集時に利用するために保管する。

これら情報のデータベースへの保管の成功をもって、投稿処理は完了する。

4.4.3 写真付き記事の投稿

続いて、写真の載った記事投稿を実現するための実装について述べる。

写真付き記事を投稿する際、本システムでは画像ファイルを別途メールで転送する方式を採る。これは、携帯電話の Web ブラウザの仕様により、ファイルをアップロード することができないからである。

以下では本システムのメール方式の実装方針を論じ、それから実装を述べる。

メール方式の実装方針

携帯電話ではる<a href="mailto:〜">タグにてメーラと連携がとれる。このとき、body=〜を記述することでメール本文に載せる情報が指定できる。ここに、これまでの記事情報を記述することとする。

ただし、このときに以下の 3 つの問題を考慮しなければならない。

1 つ目は投稿者と投稿先 Weblog システムの識別の問題である。受信したメールがいったい誰から送られた記事なのか、どの Weblog システムに送信するものなのかが分からなければ、メールを受信してもどうすることもできない。

ここで、最も単純にメール本文にユーザ ID・Weblog 設定 ID を記載するようにしてしまうと、2 つ目の問題が発生してしまう。

2 つ目は改竄の問題である。メーラで開いたメールの本文は自由に編集することができるため、その内容は容易に改竄され得る。このため、先ほど述べたように単純にメール本文にユーザ ID などを記載するようにすると、ここを改竄されるだけで他のユーザになりすまされることがあり得てしまう。さらには、メールは送信履歴などに保存しておくことができるため、これを再編集すれば本システムを介さなくても投稿できてしまう。たとえば、本システムにログインするのが面倒だからといって、記事本文の箇所や位置情報の箇所を適当に編集したものを送信される、ということが起こり得る。特に本システムの核である位置情報を不正な値に設定されてしまうと、位置情報を利用するポータルサイトの信頼性が失われてしまう。そればかりか、悪意のあるユーザがいた場合、スパムのような記事をあちこちの位置に投稿されてしまう危険性がある。

これには本文を暗号化する、という解決策がある。これにより「投稿できる」か「全く投稿できない」かをはっきりさせることができる。これで「識別」も「改竄」もクリアできる。しかし、最後の問題は先ほどの「本システムを介さなくても投稿

できる」という点から発生する。

3つ目は多重投稿の問題である。メールは一度送信しても送信履歴に残しておくことができるため、送信ボタンを押すだけで何度も送信できてしまう。単純に届いた記事を処理するような実装では防げない。さらに、悪意のあるユーザにメールボムの要領で大量のメールを送りつけられると、システムが圧迫されてしまう危険性がある。

このため、「本システムを介してないメールは一切受け付けない」という仕掛けが必要である。

そこで本研究では、すべての問題を解決でき、かつシンプルな方法として、以下の手法を考えた。

まず、これまで入力された情報の全てをデータベースに一時保管する。このときの保管場所は記事情報とは別のテーブル、一時保存記事テーブルである。さらにこのとき、ランダムな文字列を生成し、これも一緒に保管する。

このランダム文字列こそが、一時的に保管した情報を引くためのキーであり、つまり、このキーだけをメール本文に載せれば良い。

そして、メール到着時にそのキーを基に、一時保存テーブルを参照すれば、投稿途中の情報を復帰させることができる。この情報の中にはユーザ ID、Weblog 設定 ID も含まれるため、「識別」の問題は解決できる。さらに、大元のデータはシステム内に閉じ込め、ユーザが触れる部分はただの文字列であり、しかも、少しでも改竄すると情報を手に入れることができないため、「改竄」の問題も解決できる。そして、投稿処理完了後に一時保存記事テーブルからそのキーの情報を削除すれば、いくらメールを再送してもそのキーの情報は存在せず、「多重投稿」できない。これで全ての問題はクリアできる。

本システムはこの方法でメール方式での投稿を実装した。

メール投稿方式の実装

上に述べた方針をもとに、メールによる写真付き記事の投稿を実装する。図 4.1 に、記事を一時保存してから再取得するまでの流れを示す。

記事情報の一時保存

ユーザから入力された情報をデータベースに一時保存するタイミングは、確認画面から「写真の送信」ボタンを押されたときとした。ユーザの操作が確認画面まで来たときには、必要な記事情報は全て入力されているからである。

「写真の送信」ボタンを押下されると、システムはまずランダムな文字列のキーを生成し、これまで入力された記事情報と共に一時保存記事テーブルに保管する。

具体的な保管項目を表 4.5 に示す。

そして、メール送信画面へと遷移する。ここにはメールでの投稿方法の説明と、<a href="mailto:〜">タグで囲った「送信」というリンクを表示する。この<a href="mailto:〜">タグのbody パラメータに、先ほど生成したキーを載せる。

ユーザはこの「送信」リンクからメーラを起動する。そして、そこからメールに写真画像を送添付し、本システムに送信する。

メールの着信

本システムのメールサーバにメールが届くと、procmail というメールの振り分けを行うソフトから、メール投稿プログラムを呼び出す。

メール投稿プログラム

メール投稿プログラムの処理の流れを以下に示す。

A. メール解析

1. メール本文と添付ファイルの分割

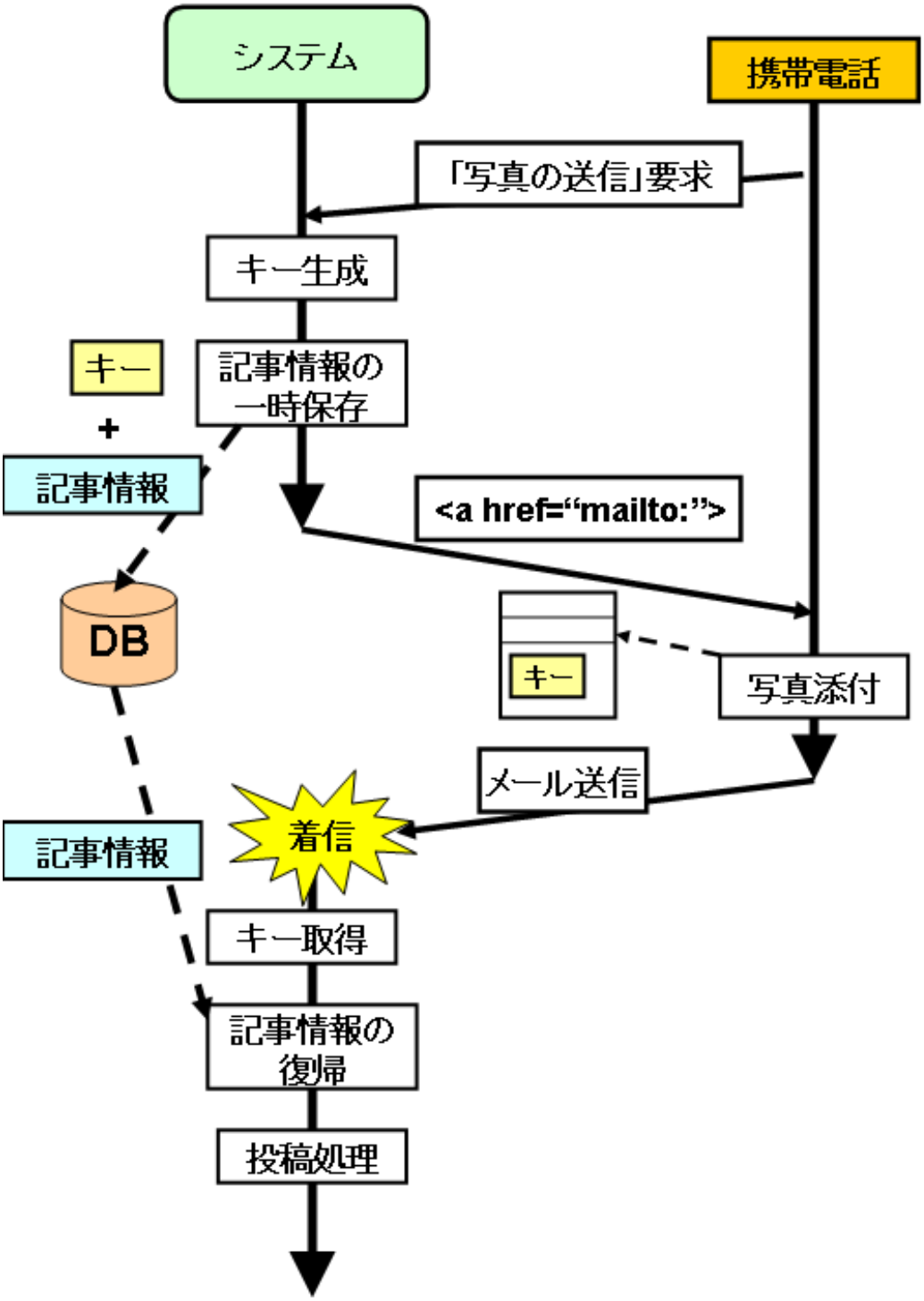

図 4.1: メール投稿方式における記事の一時保存と再取得の流れ

表 4.5: 写真付き記事投稿時、一時保存記事テーブルに保管する情報の項目

項目	備考	取得元
一時保存記事 ID	一時保存記事情報の識別 ID	システム
キー	ランダムな文字列	システム
ユーザ ID	投稿者のユーザ ID	セッション
保存日時	日本時間	システム
タイトル		ユーザ
本文		ユーザ
スポット ID	「指定なし」ならば 0	ユーザ
ジャンル		ユーザ
評価		ユーザ
緯度	世界測地系、度単位	ユーザ
経度	世界測地系、度単位	ユーザ
地図添付	記事に添付するかどうか	ユーザ
トラックバック Ping URL		ユーザ
Weblog 設定 ID	投稿する Weblog の設定 ID	ユーザ

2. 各パートから必要な情報を取得

B. 投稿

1. 画像ファイルを Weblog システムにアップロード
2. 画像ファイルを本システムに保存
3. Weblog 投稿する記事本文の調整
4. Weblog システムに記事を送信
5. データベースに記事情報を保管
6. 一時保存した記事情報をデータベースから削除

A. メール解析

まずは procmail より受け取ったメールを解析し、必要な情報を取り出す。

メール解析はまず本文パートと添付ファイルパートに分割することから行う。

本文パートではさらに本文を行ごとに解析し、キーを抽出する。そしてそのキーを基に、データベースの一時保存記事テーブルに保管した一時保存記事情報を取り出す。

添付ファイルパートではヘッダからコンテンツタイプと添付ファイル名、本文から Base64 エンコード された添付ファイルを取り出す。

B. 投稿

メール解析部で取得した各情報を使って投稿処理を行う。

1. 画像ファイルを Weblog システムにアップロード

まずは Weblog システムへの添付ファイル投稿から行う。ただし、AtomAPI で投稿できるファイル形式は画像ファイルであるのが一般的であるため、本システムは JPEG、GIF、PNG、BMP 形式のみを対象とした。

AtomAPI で Weblog システムに画像をアップロードするには UploadURI に HTTP PUT リクエストを送信する。

また、画像のアップロードの場合も、Atom エントリドキュメントによる記述で送信する。このときの Atom エントリドキュメントの構造は以下のようにする。

```
<entry>
  <title> 添付ファイル名</title>
  <issued> アップロード日時</issued>
  <content type="コンテンツタイプ"> Base64 エンコードされた
      添付ファイル</content>
</entry>
```

以下に画像アップロードリクエストを例として示す。

```
POST http://nagashima.spa.is.uec.ac.jp.private/mt/mt-atom.cgi/weblog/blog_id=2/svc=upload  HT
TP/1.1
Host: nagashima.spa.is.uec.ac.jp.private
X-WSSE: UsernameToken Username="mae", PasswordDigest="c9hNPlc/rSqwecCdbg1Cqr4vKkA=", Nonce="M
T
NmYjQ4ZGM3NTViYzBjMzA4NDYyYjc0MGY5OQ==", Created="2007-01-22T07:57:41Z"
Content-Type: text/html
Content-Length: 32377

<?xml version="1.0"?>
<entry xmlns="http://purl.org/atom/ns#" xmlns:dc="http://purl.org/dc/elements/1.1/">
    <title type="text/html" mode="escaped">061225_1914~0001.jpg</title>
    <issued>2007-01-22T07:57:41Z</issued>

    <content type="image/jpeg" mode="base64">
/9j/4QHcRXhpZgAATU0AKgAAAAgACgEOAAIAAAARAAAAhgEPAAIAAAAIAAAAmAEQAAIAAAAF
AAAAoAESAAMAAAABAAEAAAEaAAUAAAABAAAApgEbAAUAAAABAAAArgEoAAMAAAABAAIAAAIT
AAMAAAABAAEAAIdpAAQAAAABAAAAtsSlAAcAAABMAAABiAAAAAAwNjEyMjVfMTkxNH4wMDAx
                          (中略)
```

```
b1Y4JwSOOc15/wDt2roM3xbjuvAjySeHnheKxeQMC0ayMQfm5x83fmvEPEnia+1Sz03T7l2W
009WWGHIIRm2hj9SEQf8BFbvxb+J5+JV7pkqWkVjDaW+3yIslRI2N5BPOOBx25xxXxeA4flh
cwhjYfa5+bXTV3X9epz0YpxTt/Wv+Z//2Q==

    </content>
</entry>
```

上記リクエストに成功すると以下のレスポンスが返ってくる。ここで返されるAtomエントリドキュメントの構造は以下のようになる。

```
<entry>
  <title> アップロードされたファイル名</title>
  <link rel="alternate" href=" ファイルがアップロードされた
      URL"/>
</entry>
```

上記レスポンスの<link rel="alternate">タグのhref属性の値が、Weblogに保存された画像のURLであるのでこれを抽出する。

以下には、上記リクエストに対するWeblogシステムのレスポンスを示す。

```
HTTP/1.1 201 Created
Date: Mon, 22 Jan 2007 08:06:53 GMT
Server: Apache/2.2.2 (Fedora)
Pragma: no-cache
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=UTF-8

13c
<?xml version="1.0" encoding="utf-8"?>
```

```
<entry xmlns="http://purl.org/atom/ns#">
  <title xmlns="http://purl.org/atom/ns#">061225_1914~0001_10.jpg</title>
  <link xmlns="http://purl.org/atom/ns#" type="image/jpeg" rel="alternate" href="http://www.s
pa.is.uec.ac.jp/~maezawa/livelog/061225_1914~0001_10.jpg"/>
</entry>

0
```

2. 画像ファイルを本システムに保存

続いて、本システム用にも画像ファイルを保存する。このとき、どの携帯電話でも見られるように、横幅を 120 ピクセルに縮小して保存する。この縮小サイズの画像は編集処理のときや、地図インタフェースに使用する。

3.Weblog に投稿する記事本文の調整

続いて、Weblog 記事の投稿に移る。

まずは記事の本文に画像を埋め込む。本文の先頭に<img>タグを付け、src 属性の値に先ほど取得した画像 URL を設定する。

あとは通常投稿と同様に、本文の最後に位置情報として緯度・経度、住所、(記事に地図を載せるならば) 地図を付け加える。

4.Weblog システムに記事を送信

そして、AtomAPI による記事の投稿を行う。このとき送信するリクエストと、それに対するレスポンスは通常の投稿時のものと同じである。レスポンスから必要な情報を抽出する。

5. データベースに記事情報を保管

これまで集めた情報をデータベースに情報を保存する。

保存する情報は通常投稿時の内容 (表 4.4) に、Weblog にアップロードした画像 URL、本システムに保存したファイル名、画像ファイルのコンテンツを加えたものである (表 4.6)。

表 4.6: 写真付き記事投稿時、データベースに保管する情報の項目

項目	備考	取得元
基本記事情報	通常投稿時同じ内容 (表 4.4)	
画像 URL	画像のアップロード先 URL	Weblog システム
ファイル名	本システムに保存したファイル名	システム
コンテンツタイプ	画像ファイルのコンテンツタイプ	メーラ

6. 一時保存した記事情報をテーブルから削除

そして最後に一時保存記事テーブルからこの情報を削除する。

この一時保存記事情報の削除をもって、写真付き記事の投稿を完了する。

4.4.4 位置情報を付加した Weblog 記事の検索

この節では、投稿に並ぶもうひとつのメインの機能である、位置情報付き Weblog 記事の検索と、その提示方法について述べる。

以下に節 3.2 で示した検索処理の流れを再掲する。

1. 携帯電話から位置情報を受け取る
2. 渡された位置情報を基にデータベースを参照する
3. ヒットした記事情報からポータルサイトを構築する

1. 位置情報の取得

情報検索における位置情報の取得には、GPS による方法と住所やランドマークなどのキーワードから取得する方法がある。

キーワードから緯度・経度を検索するには、Geocoder という方法を用いる。本システムでは外部の Web サービス、Google Maps API の Geocoder を用いて、キーワード検索も行えるようにする。Google Maps API の Geocoder は JavaScript によるものと、HTTP リクエストによるものがある。携帯電話では JavaScript を利用できないため、本システムは HTTP リクエストで緯度・経度を取得する。

HTTP リクエストによる Geocoder は REST アーキテクチャの Web サービスである。この Web サービスを利用するためには、以下に示す URL に HTTP GET リクエストを送信する。

```
http://maps.google.com/maps/geo?q=(query)&output=(format)&key=(Google_Maps_API_Key)
```

上記の送信パラメータ “q” にキーワードを入力して送信する。また、レスポンスとして渡されるデータの形式は送信パラメータ “output” で指定できる。本システムではここに “xml”(XML 文書) を指定する。そしてこのリクエストを送信し、返信されたレスポンスから DOM を用いて緯度・経度情報を抽出する。

2. データベースの参照

取得した位置情報を基に、データベースから記事情報を参照する。

本研究では渡された緯度・経度から半径 2km 以内の記事情報を対象とした。これは、ユーザが提示されたものを見たときに「行ってみよう」と思える程度の距離として 2km が妥当だと考えたからである。

2 点間の距離はヒュベニの公式を用いて算出した。

3. ポータルサイトの構築

データベースから得た記事情報からポータルサイトを構築する。

以下ではポータルサイトで表示する情報について述べる。

地図

ポータルサイトのトップには地図を表示する。この地図にはユーザから取得した位置情報を中心にして、現在表示されている記事の指す位置にマーカーを配置させる。

これは Google Maps の地図データを利用して表示する。これは以下に示す URL を<img>タグに指定することで利用できる。

```
http://maps.google.com/mapdata?latitude_e6=(lat)&longitude_e6=(lon)&
  zm=(zoom)&w=(width)&h=(height)&cc=JP&min_priority=1&
  Point=b&Point.latitude_e6=(lat1)&Point.longitude_e6=(lon1)&Point.iconid=(iconID1)&Point=e
```

パラメータ “latitude_e6” と “longitude_e6” により中心の緯度・経度を指定する。このときの値は世界測地系・度単位表記であり、小数第 6 桁までの少数から、さらに小数点を取った文字列で記述する (例：“35.6561804163202” → “35.656180” → “35656180”)。

各位置座標に配置するマーカーは、“Point=b” 〜 “Point=d” で挟まれた部分で指定する。この部分を複数並べると、それぞれに対応したマーカーが地図に配置される。また、Point.iconidでは表示されるマーカーのアイコン画像を選択できる。本システムでは現在値には “★”、各記事には “A”、“B”、“C”、…とアルファベット順に表示されるものを指定した。

並び順

記事一覧の並び替えに、距離順・評価順・新着順を用意し、これらをリンクで選択できるようにした。このリンクの指す URL には、それぞれの並び順をパラメータとして記述してある。ここで指定された並び順によって、SQL でいう “order” を切替え、データベースを参照し直す。

記事一覧

ひとつのページに表示する記事数は 5 件程度とし、表示されない分はページ番号のリンクで補う。これは、携帯電話でのブラウジングでは、1 つのページにあま

り多くの情報を表示するとスクロール操作に不便だからである。

記事 1 件あたりに表示する情報は以下の項目である。

- 地図に表示されたマーカーのアルファベット
- 投稿日時
- クエリとして渡した位置情報からの距離
- ジャンル
- タイトル
- 評価 (「評価なし」を選択した記事には表示されない)
- 投稿者名
- 「この記事にトラックバック」リンク (節 4.4.8 にて後述)
- 携帯電話用ナビゲーションサイトへのリンク
- 「編集」「削除」リンク (閲覧しているユーザ = 投稿したユーザのときだけ表示。節 4.4.5 にて後述)

タイトルは投稿先 Weblog 記事へのリンクとして表示する。

携帯電話用ナビゲーションサイトは NAVITIME[8] の携帯電話用サイトを利用する。NAVITIME は携帯電話用のナビゲーションサービスを提供しており、このリンクからサービスを利用することができる。このサイトを利用して、気になった記事の示す場所へスムーズに移動できるようにした。

ジャンル検索・スポット検索

また、検索結果をジャンル・スポットで絞りこんだジャンル検索、スポット検索も用意した。ジャンル検索ではデータベースを検索する条件に、さらにジャンルを加える。スポット検索ではスポット ID だけをキーとしてデータベースを検索する。

4.4.5 投稿済み記事の編集・削除

投稿済み記事の編集・削除は、「投稿履歴」ページ、もしくはポータルサイトに表示された自分の記事から「編集」・「削除」リンクを選択することで、それぞれの操作を行えるようにする。

これらのリンクの指す URL には、その記事情報を指す ID がパラメータとして記述されている。このパラメータにより、システムがどの記事の編集・削除を要求されたのか識別できる。この ID を基にデータベースを参照し、投稿済みの記事情報を取得する。

以下では編集、削除のそれぞれに対するシステムの処理について述べる。

編集

編集では、はじめにまず投稿済み記事情報を投稿フォームと同様のフォームに記載し、 これを編集フォームとする。その後のユーザに行う操作は投稿操作とほぼ同じである。

ただし、編集中の記事が写真付きならば、写真の編集まで考慮しなければならない。このときは投稿フォームと確認画面の間に写真編集フォームを挟む。これについては後ほど述べる。

確認画面にて「編集」ボタンが押下されると、システムは編集処理を行う。以下は本システムの編集処理の流れである。

1. 投稿先の Weblog 記事の編集
2. データベースの更新

1. 投稿先の Weblog 記事の編集

まずは投稿済みの Weblog 記事を編集する。

AtomAPI による編集は EditURI に対し、HTTP PUT リクエストを送信する。EditURL は投稿済み記事情報に含まれている。

WSSE 認証に必要なユーザ名とパスワードは投稿済み記事情報内の投稿先 Weblog 設定 ID を使ってデータベースを参照することで得られる。

送信する Atom エントリドキュメントも投稿時のものと同じである。記事本文には投稿時同様、位置情報を付加する。

編集時の Weblog システムからのレスポンスには「成功」か「失敗」かの情報しか載っていないため、ここから新しく得られる情報はない。

2. データベースの更新

Weblog 記事の編集に成功すると、最後にデータベースの更新を行う。ここでは変更され得る以下の情報を更新する。

- タイトル
- 本文
- 緯度
- 経度
- スポット ID
- ジャンル
- 評価
- 記事に地図を添付するかどうか

このデータベース更新の成功をもって、編集処理を完了する。

写真の編集

写真付き記事を編集するときは写真の編集に関しても考慮しなければならない。確認画面に進む前に、写真を編集するかどうかユーザに選択させる。

写真付き記事の編集は以下の 3 パターンあり、このときの選択によって今後の処理が少し変わる。

(i) 本文のみの編集 (写真には変更なし)

(ii) 写真の削除

(iii) 写真の編集 (差し替え)

(i) と (ii) の場合は本システムの Web アプリケーション内で処理できるが、(iii) の場合は差し替える新しい写真をメールで送信してもらう必要がある。

(i) の場合、載せる写真は変更がないため、投稿時と同じ画像 URL を<img>タグに指定すれば良い。この画像 URL はデータベースから取得した投稿済み記事情報に含まれている。この<img>タグを付け加えれば、あとはこれを Weblog システムに送信すれば良い。

(ii) の場合は、<img>タグを付けないまま送信すれば良い。こうすれば Weblog 記事に写真画像が表示されない。ただし、Weblog にアップロードした画像の編集・削除は行わない。これは、AtomAPI による画像編集に対応していない Weblog が多いため、本システムでも対応しないこととした。それゆえ、(ii) を選択しても、Weblog 記事には写真は載らなくなるが、Weblog には画像ファイルが残ったままである。一方、本システムに保存したファイルは削除する。

(iii) の場合は、メールでの添付が必要となるため、一旦、一時保存記事テーブルに保存する。ここに保存する情報は写真付き記事投稿で保存した内容 (表 4.5) に、EditURI、記事 ID、システム内に保存した画像ファイル名を加えたものである (表 4.7)。そして、写真付き記事の投稿処理と同様、ユーザにメールによる画像ファイルの転送を要求する。

表 4.7: 写真付き記事編集時、一時保存テーブル保管する情報の項目

項目	備考	取得元
基本記事情報	写真付き投稿時と同じ内容 (表 4.5)	
EditURI	Weblog 記事の編集用 URL	データベース
記事 ID	編集する記事情報の ID	データベース
ファイル名	システムに保存した画像ファイル名	データベース

メールを受信したあとの処理も編集時とほぼ同じだが、最後にシステムに保存していた編集前の画像ファイルを削除する。

以上のようにして、写真の編集を実現する。

削除

削除処理も「削除」リンクが選択されるとその URL に記されたパラメータから記事情報を取得し、削除確認画面にその情報を表示する。そしてそ「削除」ボタンを押下されると、システムは削除処理を行う。

削除処理の流れも投稿処理とほぼ同じである。以下にその処理の流れを示す。

1. Weblog 記事の削除
2. データベースの更新

1. 投稿先 Weblog 記事の削除

AtomAPI による削除は EditURI に対し HTTP DELETE リクエストを送信する。このリクエストには Atom エントリドキュメントは必要ない。

また、写真付き記事を削除しても、本システム内の画像ファイルは削除するが、Weblog にアップロードした画像はやはり削除できない。

2. データベースの更新

Weblog 記事の削除が成功したら、データベースからこの記事を削除する。

このデータベースの削除によって、削除処理を完了する。

4.4.6 記述中記事の一時保存

記事の一時保存では、記事の保存先を投稿記事用のテーブルではなく、別に用意した一時保存記事用のテーブルとすることで実現する。

以下では記述中の記事の一時保存とその再開について、それぞれの実装を述べる。

一時保存

投稿フォームの「一時保存」ボタンが押下されると、システムは途中まで入力された情報を、全てデータベースに一時保存する。

表 4.8 にこのときに保管する一時保存記事情報の項目を示す。

このデータベースへの保管の成功をもって、記事の一時保存を完了する。なお、記事編集時の一時保存は認めない。

再開

再開は「未投稿記事」ページで表示される並未投稿記事記事一覧から、再開したい記事のリンクを選択する。

このリンクには一時保存記事 ID が記されており、この ID を基にデータベースを参照し、一時保存記事情報を取得する。そして、取得した情報は投稿画面フォームに復帰させる。

あとの流れは投稿と同じだが、最後に一時保存した記事情報をデータベースから削除する。

表 4.8: 一時保存時、データベースに保管する一時保存記事情報の項目

項目	備考	取得元
一時保存記事 ID	一時保存記事情報を識別する ID	システム
ユーザ ID	保存したユーザの ID	セッション
保存日時	日本時間	システム
タイトル		ユーザ
本文		ユーザ
スポット ID	「指定なし」のときは 0	ユーザ
ジャンル		ユーザ
評価		ユーザ
緯度	世界測地系、度単位	ユーザ
経度	世界測地系、度単位	ユーザ
地図添付	記事に地図を添付するかどうか	ユーザ
トラックバック Ping URL		ユーザ
Weblog 設定 ID		ユーザ

データベースの削除の成功をもって、一時保存された記事の投稿を完了する。

4.4.7 お気に入り

お気に入りの管理はデータベースでなく、ユーザごとに用意したファイルを用いることにした。このファイル名はユーザ ID を基に付けられている。これにより、ユーザの要求に対しどのファイルを操作すれば良いか、システムが識別できる。

位置とスポットを登録できるが、位置はその「緯度・経度」を、スポットはその「スポット ID」を登録する。また、参照する記事情報は位置の場合はその緯度・経度から半径 2km 以内にある記事情報、スポットの場合はそのスポットに所属する記事情報である。このように位置とスポットでは登録する情報、参照する情報に多少の差異はあるが、本システムではこの 2 つを同等に扱えるように実装した。

以下では、まずお気に入りを表現するファイルの中身について述べ、それからそのファイルを使ってどのようにお気に入りの登録、閲覧を実装したかについて述べる。

お気に入りファイルの情報

表 4.9 に、このファイルに保存するお気に入り情報の項目を示す。これらの項目の値を “:” で区切り、1 行単位でお気に入りを表現する。なお、お気に入り名に “:” が使われている場合は “\:” でエスケープする。

以下にお気に入りファイルの例を示す。

```
p:8:電気通信大学:0:0
l:0:調布駅\:南口:35.6518912928024:139.544477462769
```

お気に入りの登録

お気に入り登録は、位置情報取得時やスポット検索時などさまざまなタイミングで表れる「この位置をお気に入り登録」リンク、もしくは「このスポットをお気

表 4.9: お気に入りファイルで管理する情報の項目

項目	値
位置・スポットの識別記号	p：スポット l：位置
スポット ID	
お気に入り名	任意
緯度	世界測地系、度単位
経度	世界測地系、度単位

に入り登録」リンクの選択によって受け付ける。このリンクの指す URL には、位置を登録する場合は緯度・経度を、スポットを登録する場合はスポット ID をパラメータとして記述する。

このリンクを選択されると、お気に入り登録フォーム画面に遷移する。このフォームでユーザに要求する情報は、お気に入りの登録名である。選択された情報がスポットだった場合は、ここにスポット名を記述するが、これは任意で編集しても良い。

「登録」ボタンが押下されると、システムはお気に入り登録を行う。まず、前述した各情報項目を “:” でつなぎ、それをファイルに追記する。

正しく保存されればお気に入り登録は完了する。

お気に入りの閲覧

閲覧はまず、お気に入り一覧を表示するところから始まる。

お気に入りファイルを 1 行ずつ読み、さらにその行を “:” で分割し、各情報項目を取り出す (またこのとき、お気に入り名にエスケープ文字 “\:” があれば “:” に

戻す)。

もしその行、つまりお気に入りが位置だったら、その緯度・経度から半径 2km 以内で書かれた投稿記事の総数と最新投稿日時をデータベースから算出する。一方、そのお気に入りがスポットだったら、そのスポット ID のスポット情報から所属記事数と最新投稿日時を取得する。そしてそれぞれのお気に入り情報を最新投稿時間の新しい順にソートして一覧表示する。

お気に入りの名前はリンクになっており、このリンクが選択されると、そのお気に入りに所属する記事一覧を表示する。位置の場合は登録された緯度・経度から半径 2km 以内の記事情報、スポットの場合は登録されたスポット ID に所属している記事情報をデータベースから取得して、そのポータルサイトを構築する。

お気に入りの編集・削除

お気に入り一覧ではそれぞれのお気に入り情報に「編集」・「削除」リンクが付いている。

各リンクの指す URL には、その情報が書かれている行番号が記述されておりこれを基に何行目を編集・削除するのかを識別する。

お気に入り編集フォームでは「編集」ボタン、削除フォームでは「削除」ボタンを押下されると、システムはそれぞれの処理を行う。

まず、お気に入りファイルを読み込み、指定された行の書き換え、もしくは削除を行い、再びお気に入りファイルに書き戻す。

正しくファイルに書き込まれれば、お気に入りの編集・削除は完了である。

4.4.8 トラックバック

トラックバックは Weblog 記事が固有にもつトラックバック Ping URL にトラックバック Ping を送信することで実現できる [9]。

本システムでは、投稿フォームにてトラックバック Ping URL の入力を受け付け、もし入力があれば、その URL にトラックバック Ping を送信する。

このトラックバック Ping URL はユーザが自分で発見してコピー&ペースト、もしくは直接入力することができる。しかし、携帯電話でこのような操作は苦労するため、本システムでは自動検出の機能を設けた。

ポータルサイトから「この記事にトラックバック」を選択すると、投稿フォームに遷移する。そして、このフォームのトラックバック欄には、自動的に検出したこの記事のトラックバック Ping URL が記載されている。

以下ではトラックバック Ping URL を自動検出するための仕組みと、トラックバック Ping の送信方法について述べる。

トラックバック Ping URL の自動検出

Weblog 記事からトラックバック Ping URL を自動検出するための仕様として、TrackBack Auto-Discovery がある。TrackBack Auto-Discovery は Weblog 記事の HTML 内に RDF(Resource Description Framework) と呼ばれるデータを埋め込み、この RDF データ内にトラックバックに関する情報を記述することで実現する。

この RDF データは、たとえば、以下のようになる。

```
<!--
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:trackback="http://madskills.com/public/xml/rss/module/trackback/"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
<rdf:Description
    rdf:about="http://www.spa.is.uec.ac.jp/~maezawa/livelog/2007/01/post_33.html"
    trackback:ping="http://nagashima.spa.is.uec.ac.jp.private/mt/mt-tb.cgi/70"
    dc:title="今日の出来事"
    dc:identifier="http://www.spa.is.uec.ac.jp/~maezawa/livelog/2007/01/post_33.html"
    dc:subject=""
    dc:description="utf-8"
```

```
    dc:creator="まえざわ"
    dc:date="2007-01-22T20:00:36+09:00" />
</rdf:RDF>
-->
```

上のように、<rdf:Description />タグ内にWeblog記事に関するメタデータが記述されている。そしてこのタグ内の、trackback:ping属性の値にこのWeblog記事のトラックバックURLが指定されている。つまり、目的のWeblog記事のHTML文書から上記のようなRDFデータを見付け、その中にあるtrackback:pingの値を抜き出せば良い。

以下ではポータルサイトでの選択からトラックバック Ping URL 発見までの処理の流れを述べる。

1. 選択された記事の送信先URLをデータベースから取得する
2. このURLに対しHTTP GETリクエストを送信し、HTML文書を受信する
3. HTML文書から上記のようなRDFデータを検出する。ただし、HTML文書中には複数のRDFを記述できるため、RDFデータ中のcd:identifierの値と送信先URLとが一致するものを探す
4. 見付かったRDFデータの中からtrackback:ping要素の値を取り出す

以上により、ポータルサイトで選択した記事のトラックバックURLを自動検出することができる。

トラックバック Ping の送信

記事の投稿時、ユーザからの入力にトラックバック Ping URL があれば、本システムをトラックバックPingの送信処理を行う。

表 4.10: トラックバック Ping の送信パラメータ

名前	内容
url	Weblog 記事への URL(必須)
title	Weblog 記事のタイトル
excerpt	Weblog 記事の概要
blog_name	Weblog の名前

処理を行うタイミングは Weblog システムに投稿したあとで、Weblog システムからのレスポンス情報からトラックバックリクエストを作成し、指定のトラックバック Ping URL に Ping を送信する。

トラックバックは REST アーキテクチャのプロトコルを採用し、トラックバック Ping の送信はトラックバック URL に対する HTTP POST リクエストで行われる。このとき、表 4.10 に示すパラメータに投稿した記事の情報を指定する。また、Content-Type には必ずapplication/x-www-form-urlencoded を指定する。

以下にトラックバック Ping の例を示す。

```
POST http://nagashima.spa.is.uec.ac.jp.private/mt/mt-tb.cgi/70
Content-Type: application/x-www-form-urlencoded; charset=utf-8%

title=タイトル&url=http://www.spa.is.uec.ac.jp/~maezawa/myblog/&excerpt=本文&blog_name=My ブログ
```

このようにしてトラックバック Ping URL にリクエストを送信すると、レスポンスとして XML 文書が返信される。このレスポンスには成功、もしくは失敗 (エラーナンバー) の情報が記述されている。

このレスポンスを受信することで、トラックバック Ping の送信を完了する。

4.4.9 地図インタフェース

地図インタフェースは Google Maps API を含む JavaScript で実装を行った。

この地図インタフェースがブラウザから読み込まれると、システムはまず、データベースにある記事情報の全てを JavaScript 内に埋め込む。こうすることにより、JavaScipt のコード内で本システムの記事情報を扱うことができる。

この JavaScript コードには記事情報を表現するクラスが定義してある。そして埋め込んだ記事情報のひとつひとつに対しこのクラスのインスタンスを生成し、これを配列に格納していく。この配列は大域変数であり、ポータルマップ画面を開いている間は保持される。記事情報の集合をひとつの配列で管理することで、簡易的なデータベースを表現できる。

また、スポット情報を表現するクラスも用意し、スポットに対しても同じ操作を行う。

全ての情報の読み込み、格納をおえると、地図上に全ての記事情報をプロットしていく。このとき、その記事のジャンルごとにアイコンを変化させる。

このアイコンはマウスクリックのイベントに対し、情報ウィンドウを開く仕掛けにする。この情報ウィンドウに記述する情報は、タイトル、ジャンル、投稿者名、投稿日時、緯度・経度である。タイトルは投稿先 Weblog 記事へのリンクとする。また、写真付き記事の場合は、投稿時に本システムに保存した縮小サイズの写真を載せる。

メニューフォーム

地図画面の下にはメニューフォームを配置し、このメニューからジャンル検索、スポット検索を指定できる。

ジャンル検索

メニューのセレクトフォームからジャンルを選択されると、そのジャンルのアイコンだけを表示させる。

これは、一度地図上のアイコンを全て消し、記事情報を格納した配列の中からジャンルと一致したアイコンだけをひとつずつ配置し直していくことで実現できる。

スポット検索

メニューのスポットはボタンによって「表示」・「非表示」が切り替えられる。「表示」にした場合は地図上にスポットのアイコンを表示する。これは、スポット情報を格納した配列からすべてのアイコンを配置していく。

スポットアイコンはクリックイベントに対応し、そのスポットに所属するアイコンだけを表示させ、また、情報ウィンドウにスポット名と、その緯度・経度を表示する。

この実装もジャンル検索同様、一度全てのアイコンを消去し、スポット ID と一致したアイコンだけを配置させる。

記事一覧

地図の右には記事の一覧を用意する。ここには地図の中心から近距離順に記事が並んでいる。この一覧は地図の移動イベントによって変化する中心座標に合わせて、逐次ソートし直す。

第 5 章

評価

本章では本システムの評価を行う。まず、上に述べた実装の、投稿と閲覧に関する動作確認を行う。続いて、実際本システムを利用してもらったユーザのアンケートを示す。

5.1 動作確認

本章では本システムが正しく動作することを確認するため、図 5.1 に示す写真付き記事を投稿する投稿ユーザと、この記事を周辺情報ポータルサイトから見付け閲覧する検索ユーザの操作の流れを示す。

なお、本システムは携帯電話からアクセスされるものであるが、利用画面の取得の困難であるのと、1 ページの全体像を示したいため、以下の図では PC からアクセスした利用画面を示す。

5.1.1 投稿

投稿ユーザは、幕張メッセで開かれた『恐竜博』というイベントの会場から、図 5.1 に示す写真付き記事を投稿する。

ログイン

まずはログイン画面 (図 5.2) を開き、各欄にユーザ名とパスワードを記入する。

恐竜博

すげぇ。
こんなの生で動いてんだ・・・

=== 位置情報 ===
緯度：35.64774143376582
経度：140.03482818603516
千葉県千葉市美浜区中瀬二丁目付近

携帯電話用ナビサイト(NAVITIME)
live-log

投稿者：まえざわ 日時：2007年01月28日 21:57 | パーマリンク

図 5.1: 投稿する Weblog 記事

図 5.2: ログイン画面

そして「ログイン」ボタンを押下する。

メインメニューの選択

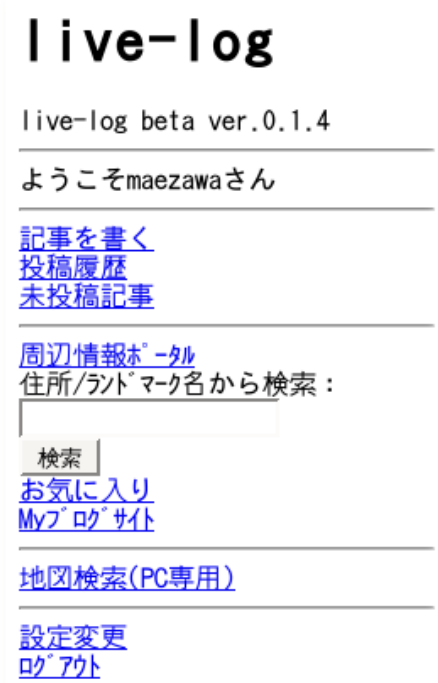

図 5.3: メインメニュー

続いてメインメニュー (図 5.3) から「記事を書く」を選択すると、携帯電話の GPS が起動し、「GPS 情報を通知します。よろしいですか?」などというメッセー

ジが表れる。ここで「はい」を選択すると、位置情報が取得され、本システムに送られる。

位置の確認

図 5.4: 位置確認画面

続いて位置確認画面 (図 5.4) が表示される。ここでは、先ほど取得した位置の結果を確認できる。表示されている位置情報を確認し、間違っていれば「再取得」を、

正しければ「決定して進む」リンクを選択する。

周辺スポット一覧の選択

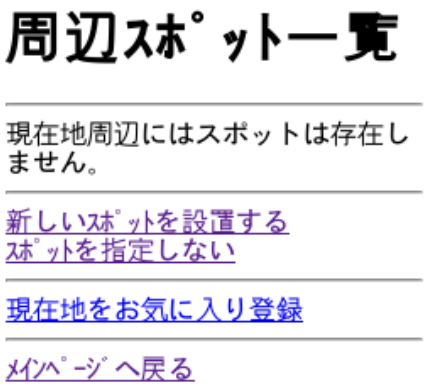

図 5.5: 周辺スポット一覧画面

次に、周辺スポット一覧画面 (図 5.5) が表示される。

ここには取得した位置近辺の周辺スポット一覧が表示されるが、この近辺にはなにもスポットが設置されていない。

そこで、新しいスポット「幕張メッセ」を設置することにする。「新しいスポットを設置する」リンクを選択する。

周辺スポット一覧の選択

「新しいスポットを設置する」リンクが選択されると、新規スポット作成画面 (図 5.6) が表示される。

ここに必要な情報を入力していく。まずは必須項目の場所名に「幕張メッセ」と記述する。住所欄は GPS から受信した位置情報を基に、本システムが自動で取得した住所である。あとの情報は任意であるため、このまま「確認画面」ボタンを押下する。

図 5.6: 新規スポット作成画面

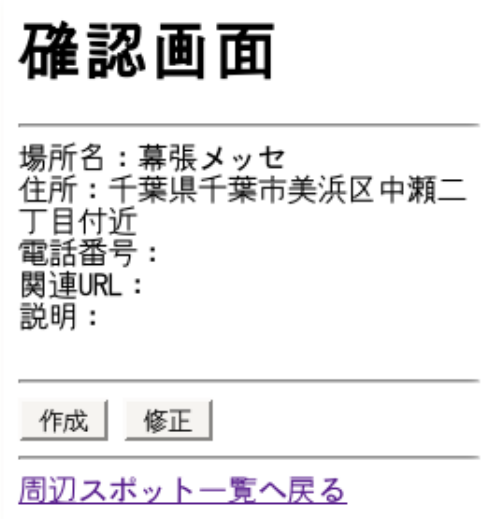

図 5.7: 新規スポット作成確認画面

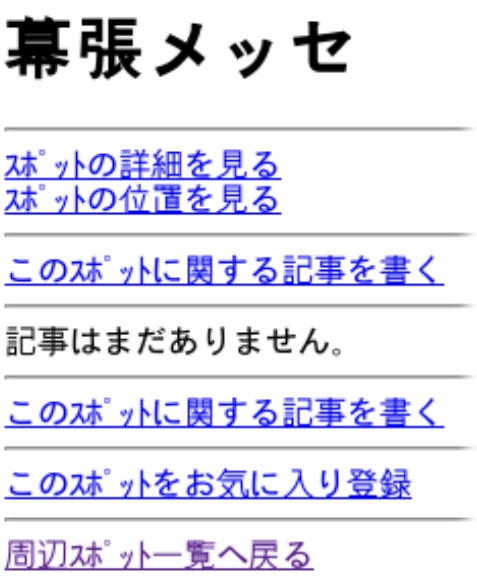

図 5.8: 「幕張メッセ」スポットに所属する記事一覧画面

そして、確認画面 (図 5.7) に進んだ後、「作成」を確定すると、新しいスポット「幕張メッセ」が作成される。

正しく作成されると「幕張メッセ」スポットに属する記事一覧画面 (図 5.8) を表示する。ただし、このスポットは先ほど立てたばかりなので、まだ誰もこのスポットに対し記事を投稿していない。

このまま「このスポットに関する記事を書く」リンクを選択し、これをスポットの決定とする。

記事の記入

所属させるスポットを選択すると、次に投稿フォーム画面 (図 5.9)) に移行する。

ここに、タイトルと記事を記入する。ジャンルは「レジャー」、評価は「4」に設定する。

また、投稿先の Weblog 記事に地図を載せるため、「記事に地図を添付する」チェッ

図 5.9: 投稿フォーム画面

クボックスにチェックを入れる。

そして、全ての記入を終えると「確認画面」ボタンを押下する。

記事の確認と投稿先 Weblog の選択

続いて、確認画面 (図 5.10) が表示される。この画面でこれまで記入した全ての情報を確認する。

また、「投稿先ブログ」セレクトボックスで記事を投稿する Weblog を選択する。

全ての情報を確認した後、続いて、写真を投稿するため、「写真の送信」ボタンを押下する。

メーラの起動と添付ファイルの送信

次に、写真の投稿画面 (図 5.11) に移行する。この画面には「送信」リンクが貼られており、ここをクリックするとメーラが起動する (図 5.12)。

メーラにはすでにメールアドレス、件名、本文が記述されており、あとは送信したい写真を添付して送れば良い。

これで投稿は完了である。

メインメニュー画面に戻り、「投稿履歴」を選択すれば、先ほど送信した記事が正しく投稿されていることが確認できる (図 5.13)。

5.1.2 検索

次は、上の例で投稿ユーザが投稿した記事を、海浜幕張駅にいる検索ユーザが周辺情報ポータルサイトから閲覧するまでの例を示す。

ログイン

ログインについては投稿時と同じである (図 5.2)。


確認画面
センター
ビー・エム
ダブリュー
千葉県精神科
医療センター
海浜幕
幕張メッセ
マリンスタジアム
アパホテル&リゾート
東京ベイ幕張
幕張
海浜公園
©2006 Google Map data ©2006 ZENRIN
ｽﾎﾟｯﾄ：
幕張メッセ
ﾀｲﾄﾙ：
恐竜博
本文 ：
すげぇ。こんなの生で動いてんだ・・・
ｼﾞｬﾝﾙ：ﾚｼﾞｬｰ
評価： ★★★★☆
地図を添付する
緯度：35.64774143376582
経度：140.03482818603516
ﾄﾗｯｸﾊﾞｯｸ： なし
投稿先ﾌﾞﾛｸﾞ： live-log test #2
以上の内容で投稿しますか?
投稿
写真を送信
修正

図 5.10: 確認画面

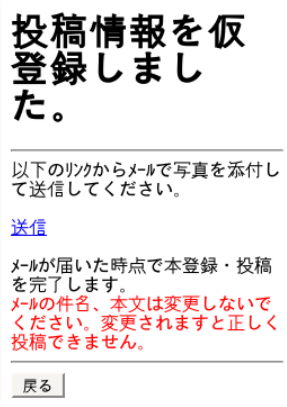

図 5.11: メール送信画面

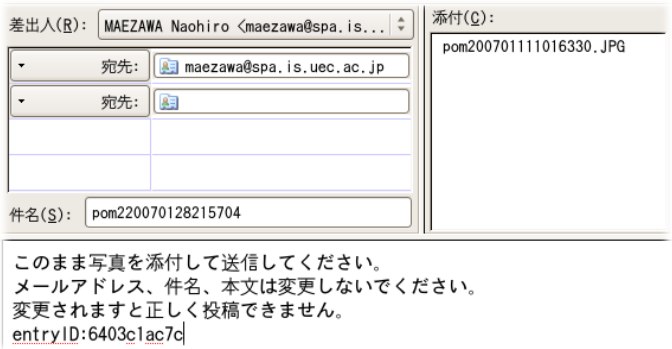

図 5.12: メールの送信

図 5.13: 投稿履歴画面

位置情報の取得

メインメニュー画面 (図 5.3) より、「周辺情報ポータル」を選択すると、GPSが起動し、位置情報の取得が開始される。

そして位置情報の送信を終えると、周辺情報ポータルサイトのトップページ (図 5.14) が表示される。

周辺スポット一覧

次に、検索ユーザはスポット検索を選択することとする。

「周辺スポット一覧」リンクを選択すると、現在地から周囲 2km 以内のスポット情報が表示される (図 5.15)。

この図では先ほどのユーザが設置した「幕張メッセ」スポットが確認できる。

続いて、「幕張メッセ」というタイトルを選択すると「幕張メッセ」スポットに所属する記事一覧が表示される (図 5.16)。この図では、先ほどのユーザが投稿した記事情報が確認できる。

この記事情報のリンクをクリックすると先ほどのユーザが投稿した Weblog 記事を閲覧することができる。

以上より、投稿ユーザは本システムから図 5.1 の Weblog 記事を投稿でき、閲覧ユーザは周辺情報ポータルサイトから、投稿ユーザの投稿した記事を発見できた。

これにより、本システムが正しく動作したことを確認できた。

5.2 アンケート 評価

本研究が実装したシステムを実際のユーザに利用してもらい、アンケートによる評価を行った。このアンケートでは 5 人のユーザに 6 つの質問をし、各質問に対

図 5.14: 周辺情報ポータルサイト・トップ画面

図 5.15: 周辺スポット一覧画面

図 5.16: 「幕張メッセ」スポットによるスポット検索

する 5 段階評価とそのコメントを答えてもらった。

以下にその結果を示す。

5.2.1 投稿の評価

質問 1 情報発信者の立場で、投稿した Weblog 記事を見て、自分の伝えたい位置の情報を閲覧者に良く示せてるか

この質問では、位置に特化した情報を発信したい投稿者にとって、本システムが Weglog 記事に与えた位置情報が、はたして要求に答えているかどうかを確認した。

以下に、評価の平均点と主なコメントを記す。

平均点：4.2

- GPS 機能により場所の情報を正確かつ簡単に載せることができる (評価：5)
- 縮尺の指定ができないようなので、込み入った場所の指定で問題が出そう (評価：4)
- 見やすくできていて良いと思う (評価：4)

以上を見るに、十分満足のいく結果が得られた。本システムを使用した場合、投稿者が閲覧者に伝えたい位置情報を十分提供できるといえる。

5.2.2 閲覧の評価

質問 2 情報閲覧者の立場で、本システムで投稿した他者の Weblog 記事を見て記事の指す位置が直感的に分かり易いか

この質問では、閲覧者が投稿された記事の位置情報を見て、その記事の指す位置を十分把握できるかということを確かめた。

以下に評価の平均点と主なコメントを記す。

平均点：4.4

- 地図の縮尺が少し不満。広い場所を指定している場合に、その場所の全景が見えなくて少し判り辛い (評価：4)
- やはり縮尺の指定が欲しい (評価：4)

結果としては十分満足な結果が得られた。しかし、表示する地図の縮尺についての意見が多かった。縮尺は Google Maps API のコントローラで調整できるのだが、やはり一番始めに目に映るものが分かりやすくなければ、記事の表現する位置はすぐに伝わらないと考える。今後は投稿時にユーザが縮尺を選べるような実装を考慮したい。

5.2.3 検索の評価

質問 3 周辺検索ポータルサイトの「全ジャンル」で表示されているものを見て、現在地と各記事の位置関係が空間的に (距離や方向、位置的に) 把握しやすいか

「全ジャンル」とはスポット検索やジャンル検索などの情報の絞り込みを、全く行わない検索方法を指す。この質問では、検索結果として提示される地図や距離などにより、各記事が示す位置が空間的に把握できるかを確かめた。

以下に評価の平均点と主なコメントを記す。

平均点：3.8

- GPS からその場とお店の距離まで計算してくれ、近い店を出せるのはよいと思う (評価：5)

- 同じような場所で書かれているようなので、ある程度バラけるように表示して欲しい (評価：2)
- 地図をスクロールできないという不満はあるが、記事の位置関係は把握しやすい (評価：4)

概ね満足な結果が得られた。この質問も地図に関する意見が多かった。ポータルサイトで表示される地図は Google Maps API のように操作できない。しかし、地図のスクロールや縮尺の自動調整などは実装次第では可能であり、今後の実装では改善していきたい。

質問 4　「全ジャンル」を見てからジャンル検索での各ジャンルの絞り込みや、スポット検索、並び順の変更などを使ってみて、これが「求めている・価値のある情報発見への近道」としてより良い誘導性を発揮しているか

この質問では、本研究が用意したジャンル検索やスポット検索、並び順の変更などから、有効な記事の発見性を高めるかどうかを尋ねた。

以下に評価の平均点と主なコメントを記す。

平均点：3.75

- ユーザインタフェースとしては良いと思う。スポット検索ではスポット名 (場所名) だけでは本当に探している場所なのかどうか不安になるので、いくつかの Weblog 記事の先頭だけでも見えると分かり易いし、面白いかもしれない (評価：4)
- グルメで検索したとき、表示されているのが Weblog のタイトルだけでは何

が食べられる店か分かり辛いかもしれない。グルメでも和食、洋食などのより細かいジャンルがあると良い (評価：3)

- (評価情報の値に対して、)5 段階評価の値だけでは情報の有用性を判断するのが困難な場合が生じると思われる。グルメでは味と価格を評価したり、投稿者の年齢を表示するなどして、情報を判断する基準を増やせば有用性があがるのではいか (評価：4)

現状の検索手法でも概ね満足な結果が得られた。しかし、さらなる発見性向上のための面白いアイデアが多く得られ、今後の改善点として考慮していく。

5.2.4 以上を踏まえた本システムの評価

質問 5 本システムが地域を中心とした情報共有の手段として利用できるか

この質問では、第 2 章で提唱した位置を中心とした情報共有のイメージが、実装したシステムで実現可能かどうかを確認した。

以下に評価の平均点と主なコメントを示す。

平均点：4.4

- 非常に有効だと思う。スポットが増えてしまうと煩雑になってしまう可能性もあるが、利便性は十分にある (評価：5)
- できると思う。ただし、ユーザインタフェースをもう少し改善してほしい (評価：4)
- 新着順で最近あった出来事を探すことができるため、地域の行事などを知るよい手がかりが得られた (評価：4)

十分に実現可能だという結果が得られた。ユーザの数を増やしていくためのユーザインタフェースの向上や、ユーザが増えたときに起こるであろう煩雑さの対策など今後のシステムが発展していくうえで必要な改良点の指摘も得られた。

5.2.5 横断的なコミュニケーション

質問 6 本システムの位置をキーとしたポータルサイトは、他のブロガーとの横断的なコミュニケーション手段として活かせるか

横断的なコミュニケーションとは、たとえばあるユーザが別のユーザの Weblog を訪れ、コメントなどを残していく、というようなコミュニケーション形態のことをいう。横断的なコミュニケーション手段は、そのコミュニケーションのきっかけ作りや活性化をはかる方法のことを示す。たとえばソーシャル・ネットワーキング・サービスでの友達の日記の更新通知や、Weblog のトラックバックなどが横断的なコミュニケーション手段であるといわれている。この質問では本システムが、位置をきっかけとして他者の Weblog に訪れる、というポータルサイトとしての役目をもつことができるかを確認した。

以下に評価の平均点と主なコメントを示す。

平均点：4.0

- 場所が近いブログを勝手にトラックバックしてくれるならば横断的なコミュニケーション手段になると思うが、現時点では地図からしかリンクが張れていないのでそこまで到っていないと思う (評価：3)
- スポットの話題を通することにより日記の書き込みなどしやすくなると思う (評価：4)
- ブログ記事に場所が関連することを書く機会は割りとあるので、地図がある

というのは良いかと (評価：4)

十分満足な結果が得らたが、トラックバックの有効利用など、改良点はまだまだありそうだということが分かった。

第 6 章

関連研究

上松らが開発した場 log システム [10] は本システムと同じくデータベースによって Weblog 記事と位置情報を結びつける。外部の Weblog に登録でき、Weblog システムとの間のプロトコルは XML-RPC を利用している。投稿方式はメール方式を採用しているが、同研究で開発した RSS に位置情報を記述する拡張プラグインを Weblog システムに導入することで、システムの専用クローラに情報を回収させることもできる。この場合の投稿は場 log システムを介さなくとも良い。収集した情報提示も本システムと同様であり、ある地点の位置情報を与えると、その地点周辺でかかれた情報を距離順、または時間順に並べて提示する。また、PC 向けの地図インタフェースも提供する。

さらに、上松らは場所にタグ付けを行うことで、場所を中心としたコミュニケーションを支援するシステム [11] を開発した。このシステムでは本研究のスポットと似た概念の情報を位置情報に与えている。また、場 log と組み合わせることで、場所から Weblog 記事を発見する機能をもつ。

お散歩マニア [12]、ちず窓 [13] は地図を利用した Weblog ポータルサービスである。地図上には各記事の示す位置にジャンルごとのアイコンが配置されている。そして、それらをクリックすることでその位置に関する情報を閲覧することができる。地図は Ajax 技術を使っているため携帯電話での閲覧はできない。投稿は moblog に対応しておらず PC からのみである。タグ作成フォームにて位置情報を含むタグを生成し、それを Weblog の記事に貼り付けると記事にドラッガブルな地図を掲載

することができる。投稿が完了あとで、クローラがその記事を回収し、データベースに登録する仕組みになっており、これによりポータルを構築できる。

マップコミ [14] は地図上に口コミ情報を書き込めるシステムである。これも先のお散歩マニアやちず窓と同じく、地図上の位置にジャンルごとに区別されたアイコンが表示され、それをクリックすると詳細を見ることができる。また、この地図も Ajax 技術を用いているため携帯電話では閲覧できない。異なる点は情報媒体は Weblog ではなく、投稿された情報はシステム内のみに納められる点である。また、既存情報と同じ場所の情報はその下に追記ができる。投稿は主に投稿フォームにて行うが、au の GPS 搭載携帯電話にも対応しており、EZ ナビウォークの現在地メール機能によりメール方式で投稿することができる。

NAVIBLOG 社は携帯電話用 Web ブラウザによる地図インタフェースを実現し、視覚的に周辺情報を検索できる NAVIBLOG MOBILE[15] を開発した。地図インタフェースから記事を投稿すると、投稿された位置のある範囲にクリッカブルなアイコンが配置され、このアイコンから情報を引き出すことができる。ただし、外部の Weblog への投稿はできず、システム内だけに情報を閉じている。

表 6.1 これらの既存システムと、本システムとの比較を示す。

表 6.1: 本システムと既存システムとの対比

		場 log[10]+文献 [11]	お散歩マニア [12] ちず窓 [13]	マップコミ [14]	NAVIBLOG[15]	本システム
投稿	投稿方式	メール Weblog システムから直接 *	Weblog システムから直接 *	Web ブラウザ (PC) メール (携帯電話)	Web ブラウザ	Web ブラウザ
	携帯電話対応	○	×	○	○	○
	外部 Weblog への投稿	○	○	×	×	○
検索	提示方式	地図、一覧	地図、一覧	地図、一覧	地図	地図、一覧
	携帯電話閲覧	○ (一覧のみ)	×	×	○ (地図のみ)	○
	キーワード検索	○	○	○	×	○
	外部 Weblog へのリンク	○	○	×	×	○
その他の機能	場所による情報共有 (スポット)	○	×	△ (記事に追記)	△ (記事に追記)	○
	ジャンルによるカテゴリ分類	×	○	○	○	○
	評価値の付加	×	×	×	×	○
	携帯電話からの記事編集	×	×	×	○	○
	記事の一時保存	○ (端末で保存)	×	○ (端末で保存)	×	○
	トラックバック Ping の送信	×	○ (Weblog の機能)	×	×	○
	位置情報の登録 (お気に入り)	×	×	×	○	○

*クローラが回収

第 7 章

おわりに

本研究では、位置と密着した moblog の投稿と検索を行うシステムの設計と実装を行った。

本システムは Web アプリケーションであり、携帯電話に搭載されている標準の Web ブラウザを使って、対話的に操作できる。投稿先 Weblog システム・サービスも汎用的なものを使えるが、AtomAPI に対応しているものとする。本システムは携帯電話と Weblog に間に立ち、各機能の処理を仲介しながら、内部にもつデータベースによって位置と情報を結び付けていく。

投稿処理ではユーザから受け取った記事情報と位置情報を使って、Weblog システムへの投稿とデータベースへの保管を行う。このとき、さまざまなメタ情報を与えることができる。このメタ情報は、記事の表現力の向上や検索時の支援に利用することができる。そして、本システムから投稿した Weblog 記事には地図などを使った位置情報が付加されている。

検索処理ではユーザから受け取った位置情報をキーとして、データベースを参照する。そして、その検索結果をポータルとして提示する。ポータルには与えられた位置情報から周囲 2km 以内の記事情報の一覧が表示される。その記事情報のリンクを選択することで、投稿先の Weblog 記事を開くことができる。また、投稿時に与えたメタ情報を絞り込みや並び順の要素として利用することにより、価値ある情報への誘導性を向上させている。

また、投稿や検索以外でも、投稿済み記事の編集・削除や、記述中の記事の一時

保存、検索した位置をブックマークできるお気に入り登録、地図インタフェースから情報検索を行える地図ポータルなど、さまざまな機能を実現した。

本システムは内部のデータベースによって位置と情報の結び付けを実現する。また、携帯電話・Weblog システム・データベースに対するそれぞれのインタフェース部をもち、これによって各相手との通信を可能にする。そして、それらへの情報の流通や状態遷移を制御する制御部によって、各機能を組み立てていった。

そして、この実装の動作確認を行い、投稿、検索の処理を正しく実行していることを確認した。また、実際にユーザに使用してもらい、アンケートに答えてもらった。その結果、本システムが位置を中心とした情報共有サービスとして、十分に有用性があることを確認した。

今後の課題として、まず、検索対象の範囲を広げたいと思っている。本システムを介していない Web コンテンツも検索対象とした、より汎用性の広い位置特化情報の検索を提供したい。

さらに、インタフェースの強化である。特に携帯電話でも対話的に扱える地図インタフェースを開発していきたい。携帯電話でも十分に表現力のあるインタフェースを模索しながら、さらなる情報の発見性の向上を目指したい。

謝辞

本研究を遂行するにあたっては、いろいろな方々にお世話になりました。

まず、指導教員の多田好克先生には日頃から熱心なご指導、そしてご鞭撻を賜わりました。また、ご多忙中にもかかわらず論文の草稿を丁寧に読んで下さり、大変貴重なご助言をいただきました。ここに厚く御礼申し上げます。

また、佐藤喬助手には日頃から研究方針や研究方法に多くのご指導をいただきました。また、ご多忙にもかかわらず論文の草稿を丁寧に読んで下さり、大変貴重なご助言をいただきました。ここに厚く御礼申し上げます。

そして、本研究が行なえたことは、研究方針や方法論について議論をし、共に研究生活をおくってきた多田研、そして村山研の学生諸氏のおかげでもあります。最後に、これらの皆さんに感謝いたします。

参考文献

[1] XHTML Basic, http://www.w3.org/TR/xhtml-basic/

[2] 水島 壮太, 小檜山 賢二: 「GPS モブログ」による社会調査プラットフォームの研究開発, 情報処理学会研究報告,Vol.2005, No.28, 2005-MBL-32, pp.247–254 (2005).

[3] 平田 真章, 千葉 雅裕, 仁田 壮一: 携帯電話向けブログクライアントの開発, 情報処理学会研究報告,Vol.2005, No.71, 2005-HI-114, pp.25–30 (2005).

[4] J. Gregorio, BitWorking, R. Sayre: The Atom Publishing Protocol, draft-ietf-atompub-protocol-12.txt, IETF Internet-Draft, (2006).
http://www.ietf.org/internet-drafts/draft-ietf-atompub-protocol-12.txt

[5] IBM, Microsoft, VeriSign: Web Services Secuirty,
http://www-128.ibm.com/developerworks/library/specification/ws-secure/

[6] Nakamura-KU ADDICT, http://www.knya.net/archives/2005/07/rest.html

[7] Google Maps API: http://www.google.com/apis/maps/documentation/

[8] NAVITIME: http://www.navitime.co.jp/

[9] Six Apart: TrackBack Technical Specification,
http://www.sixapart.com/pronet/docs/trackback_spec

[10] Hiroki Uematsu, Kosuke Numa, Tetsuro Tokunaga, Ikki Ohmukai, and Hideaki Takeda: Balog : Location-based Information Aggregation System. In Poster Proceedings of 3rd International Semantic Web Conference (ISWC2004), (2004).

[11] 上松大輝, 沼 晃介, 濱崎雅弘, 大向一輝, 武田英明: タグ付けされた場所に基づいたコミュニケーション支援, 第19回人工知能学会全国大会論文集, Vol.JSAI05, pp.35–37 (2005).

[12] BLUEPAGE: お散歩マニア, http://www.osanpomania.net/

[13] ちず窓製作委員会: ちず窓, http://chizumado.jp/

[14] デジタルナビ: マップコミ, http://mapcomi.jp/

[15] NAVIBLOG: NAVIBLOG MOBILE, http://www.naviblog.jp/