MiSpider に基づく Web ブラウジング支援エージェントの実装

Implementing a Web Browsing Support Agent Based on MiSpider

深萱裕二郎 † 大囿忠親 † 伊藤孝行 † 新谷虎松 †
Yujiro FUKAGAYA Tadachika OZONO Takayuki ITO Toramatsu SHINTANI

† 名古屋工業大学大学院 工学研究科 情報工学専攻
Graduate School of Engineering, Nagoya Institute of Technology
{fukagaya, ozono, itota, tora}@ics.nitech.ac.jp

本稿では，我々が開発した Web 上におけるエージェント環境 MiSpider を用いた，Web ブラウジング支援を行うエージェントの実装について述べる．MiSpider を用いることで，エージェントはセッション間で内部情報を保持したままユーザのブラウジングしているページに移動し，さまざまな Web ブラウジング支援を行うことができる．本稿では特にブックマークエージェントについて述べる．

1 はじめに

近年，WWW の発展により，様々な情報が Web 上で公開され，Web ブラウジングの機会が増加した．そのため，ブラウジング支援の重要性が高まっている．ブラウジングを支援する方式として，ユーザがブラウジングしているページについて移動するエージェントによる方法であれば，ユーザにとって利用しやすい．しかし，従来の方法ではセッション間の情報の受け渡しが困難であり，エージェントが情報を保持したまま，Web ページ上を移動することができない．MiSpider では，以上の問題について解決を行う．

MiSpider におけるエージェントは，ユーザが閲覧ページを移動した時に，セッション間で内部情報を保持したまま移動できる永続性を持つ．また，Web 上の任意のエージェント間で通信を行うためのメッセージパッシング能力を持つ．このような，エージェントの機能を利用することで，ユーザにさまざまな Web ブラウジング支援を提供することができる．

ブラウジング支援としては，ユーザが頻繁に利用するサイトに，どのような環境からでも，容易にアクセスできる必要がある．また，ユーザが求める情報の検索を容易にする必要がある．前者を解決するために，ユーザのブックマークの情報 (ディレクトリ構造，ページ名，URL) を保持するエージェント (ブックマークエージェント) を実装する．後者を解決するために，ユーザが検索サイトに送ったクエリを保持するエージェント (クエリエージェント) を実装する．

ユーザは既存の Web ブラウザを用いて，特別なソフトウェアなしで MiSpider を利用できる．そのため，インターネットに接続されている環境であれば，世界中からエージェントを利用することができる．

第２章では，関連研究との差分について述べる．第３章では，MiSpider の構成，機能について述べる．第４章では，MiSpider を利用した Web ブラウジング支援として，特にブックマークエージェントについて述べる．最後に第５章で本論文をまとめる．

2 関連研究

2.1 Ajax

Ajax[3] とは，Web ブラウザに実装されている JavaScript の HTTP 通信機能を使い，Web ページのリロードを伴わずにサーバと XML 形式のデータのやり取りを行い，処理を進めていく対話型 Web アプリケーションの実装形態である．実際には，Ajax という技術が存在しているわけではなく，DHTML (JavaScript + CSS) や XMLHttpRequest などのクライアント側の技術にサーバ側の Web アプリケーションとを加えた実装形態のことを指す．Ajax では，指定した URL から XML ドキュメントを読み込む機能を使い，ユーザの操作や画面描画などと並行してサーバと非同期に通信を行なうことで，サーバの存在を感じさせないシームレスな Web アプリケーションを実現する．

Ajax と MiSpider を比較すると，アプローチとして，JavaScript, XML, DOM, XMLHttpRequest などの技術を組み合わせて利用する点が共通点である．差分となる点は，Ajax ではアプローチ，モデルの提案がなされている．一方，MiSpider ではフレームワーク

として提供するインフラを利用することで MiSpider の機能を利用した開発ができる．また，Ajax ではインタラクションの向上を主な目的としている．Web サーバ側の処理の間にユーザを待たせないようにして，リアルタイムでユーザのアクションを反映する．一方，MiSpider ではインタラクションに加えて，ユーザ間のコミュニケーションを可能にすることも目的としている．既存サイトへ適用する場合，MiSpider を利用することで，既存のソースコードと独立して機能を追加することができる．

3 Web 上におけるエージェント環境 MiSpider

MiSpider は永続性，メッセージパッシング，GUI などの API を提供する．開発者は，提供された API を利用して，容易にエージェント開発を行うことができる．

ユーザは，Web ブラウザから本システムにログインする．ログイン時に利用するエージェント名，パスワード，および閲覧する Web ページの URL を入力する．ユーザは，指定したエージェントの機能をブラウジング中に利用することができる．

処理の記述には JavaScript を用いる．JavaScript は，Web ページ内の情報の取得が容易である．また，Web 上におけるユーザの操作に応じた処理を記述できるため，Web 上におけるエージェント開発に適していると考えることができる．

図 1 に MiSpider の構成を示す．MiSpider は，Base Agent と Page Agent から構成される．ユーザが，Web ブラウザから，Web プロキシ（CGI として実装される）を経由して，ある Web ページにアクセスする．対象となる Web ページは，Web プロキシを通過する時に，Base Agent によって，MiSpider のエージェントが埋め込まれる．すなわち，JavaScript として，ページにエージェントが埋め込まれる．これが，Web ブラウザに送信され，Page Agent がエージェントを動作させる．

図 2 に MiSpider の構成図を示す．MiSpider は，主に JavaScript と CGI から構成される．Web ブラウザから，CGI によるプロキシを通して Web ページにアクセスする．プロキシでは，キャッシュした元の HTML ソースに JavaScript の埋め込み，およびリンク先をプロキシ経由に書き換える処理を行い，Web ブラウザに表示する．埋め込んだ JavaScript ファイルには (a) 閲覧ページの読み込み時の処理，(b) 閲覧ページ移動時の処理，(c) エージェントが実行する処理を記述する．(a) において，MiSpider へのログイン時に指定したエージェント *Agent* の Web ページへの埋め込み，および XML ファイル (*Agent*.xml) から *Agent* の情報の取得を行う．取得した XML ソースを JavaScript のオブジェクトに変換し，(c) の *Agent* の処理を実行する．(b) において，現在の *Agent* の情報を XML ソースに変換し，セーブを行う CGI に送信する．セーブを行う CGI では，受け取った *Agent* の情報を XML ファイル (*Agent*.xml) に書き込む．(a) および (b) の処理によって永続性を実現している．

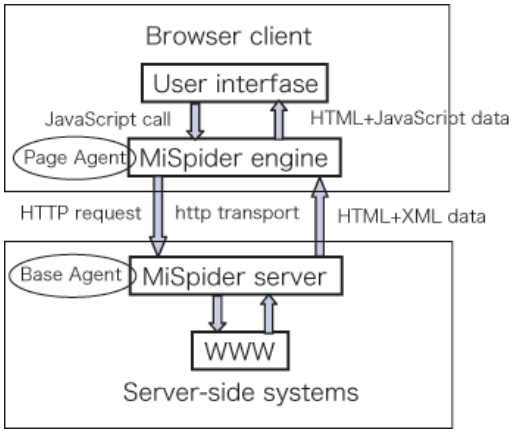

図 1: システムの構成

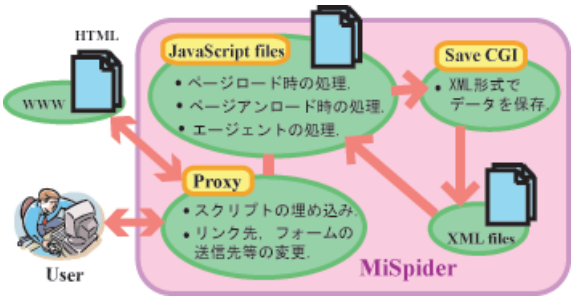

図 2: システムの実装方式

3.1 メッセージパッシング

Web 上の任意のエージェントとメッセージ通信を行うことを考える．そのときの問題点として，閲覧ページの更新がエージェントに及ぼす影響が挙げられる．従来の通信方法では，通信時に閲覧ページのリロードを行わなければならない．そのため，Web ページ上のエージェントの情報が初期化される．このようなことを防ぐために，Web ページの更新が不

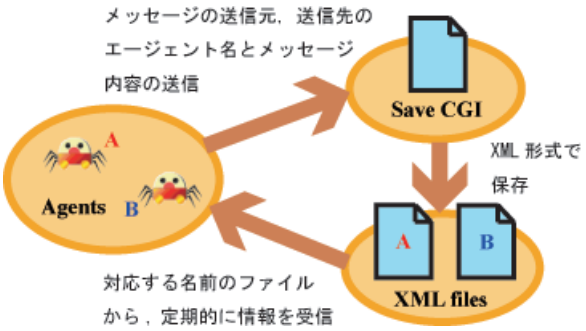

図 3: メッセージパッシング

```
[JavaScript]
kbase.hoge_site.items[0].title = "item0";
kbase.hoge_site.items[0].url = "http://www.hoge.co.jp/item0.html";
kbase.hoge_site.items[1].title = "item1";
kbase.hoge_site.items[1].url = "http://www.hoge.co.jp/item1.html";

[XML]
<kbase>
 <knowledge>
  <name>hoge_site</name>
  <items>
   <item>
    <title>item0</title>
    <url>http://www.hoge.co.jp/item0.html</url>
   </item>
   <item>
    <title>item1</title>
    <url>http://www.hoge.co.jp/item1.html</url>
   </item>
  </items>
 </knowledge>
</kbase>
```

図 4: JavaScript のオブジェクトと XML の対応例

要な通信方法が必要である．

図 3 にメッセージパッシングの仕組みを示す．*agent-A* が *agent-B* へメッセージを送る場合，*agent-A* は CGI に *agent-B* の名前とメッセージ内容を送信する．そして，CGI では，送信先のエージェントの名前のファイル（*agent-B*.xml）に XML 形式で保存する．各エージェントは，対応する名前の XML ファイルにアクセスして，定期的に情報を更新する．このときに，*agent-B* は *agent-A* からのメッセージを取得する．

メッセージの送受信には，XMLHttpRequest オブジェクトを用いる．XMLHttpRequest オブジェクトは，Web クライアントが直接 XML データを送受信することを可能にする．また，その際にページのリロードを行う必要がない．そのため，リロードを行う必要なしでメッセージパッシングを行うことができる．

3.2 永続性

ユーザが閲覧ページを移動した時に，エージェントの情報が初期化される．閲覧ページを移動した後

図 5: ブックマークエージェント

も継続的にエージェントに処理を行わせるためには，前のページをアンロードする時に，実行状態を退避し，新しいページをロードする時に実行状態を復帰する手法が必要である．

MiSpider では，プロキシを通して Web ページにアクセスする．proxy では，JavaScript の埋め込みやリンク先の書き換えなどを行う．埋め込まれた js ファイルには，ページ読み込み時の処理と，閲覧ページ移動時の処理を記述する．解決手法は以下の通りである．ページをロードする時に，XML ファイルにアクセスする．取得した XML ソースをパージングして，JavaScript オブジェクト *kbase* に変換する．そして，*kbase* から実行状態を取得して，エージェントの処理を開始する．ページをアンロードする時には，*kbase* を XML ソースに変換し，セーブを行う CGI に送信する．XML を用いることで，JavaScript のオブジェクトや配列などのデータ構造を表現できる．XML の木構造の節はタグ名が複数形の時は配列，単数系の時はオブジェクトに対応する．木構造の葉は文字列として扱われる．JavaScript のオブジェクトと XML の対応関係の例を図 4 に示す．

4 MiSpider を用いた Web アプリケーション

4.1 ブックマークエージェント

ブックマークエージェントにより，家と職場といった異なる環境から共通のブックマークを利用することができる．既存のブックマーク共有サービスとして Yahoo!ブックマーク [4]，Book まーく [5]，はてなブックマーク [6] などがある．MiSpider におけるブッ

クマークエージェントの差分として，エージェントがブラウジングページについて移動するため，ユーザのブラウジングを中断する必要なしに，ブックマークの利用や新しいブックマークの登録を手軽に行うことができる．その際に特別なソフトウェアを必要としない．また，メッセージパッシングによって，特定のエージェントを利用するユーザとブックマークを共有できる．そのため，ユーザはブックマークの利用に関して，本人のみの利用，特定ユーザとの共有，ソーシャルブックマークのような全体への共有というように，共有の範囲を細かく決めてブックマークを利用することができる．

図 5 にブックマークエージェントの利用画面を示す．メニューボタンでブックマークエージェントを選ぶと，ブックマークの追加ボタンとブックマークの一覧が表示される．ブックマークの一覧中から，ブックマークに登録したタイトルをクリックすることで，登録したページへ移動する．メニューボタンを再度押すことで表示は消える．

ブックマークエージェントの実装

ブックマークエージェントの実装について以下に述べる．作成したプログラムは，プロキシで埋め込まれるファイルに追加することで利用できる．

ブックマークの登録

```
function saveBookmark() {
  var bk = new Object();

  // Bookmark 名の入力 (デフォルト値: ページタイトル)
  bk.title = prompt("ブックマークの名前を入力してください", document.title);
  bk.url = url;    // url: 現在閲覧中のページの URL
  // type='page': ページを表す，'dir':ディレクトリを表す
  bk.type = 'page';

  if(bk.title && bk.url) {
    kbase.bookmark.items.push(bk);
    saveCondition();
  }
}
```

新しいブックマークのオブジェクトを作成し，ブックマークの配列の最後に入れる．saveCondition ではオブジェクト kbase を 3.2 章で示した方式で XML に変換されて，XML ファイルに保存される．

ブックマークの表示

```
function showBookmark() {
  var displayHTML = openTags;

  var bks = kbase.bookmark.items;

  for(var i=0; i<bks.length; i++) {
    if(bks[i].type == 'page') {
      displayHTML += '<div onclick="location.href=\''
        +bks[i].url+ '\'">' + bks[i].title + '</div>';
    } else if(bks[i].type == 'dir') {
      displayHTML += '<div onclick="showDir(\''+i+
     '\')">' + bks[i].title + '</div>';
    }
  }

  displayHTML += closeTags;
  document.getElementById(agent_disp_id).innerHTML
    = displayHTML;
}
```

ブックマークを利用するための HTML を作成する．kbase.bookmark.items に格納された要素を順に見て，ページを表すときはクリック時に登録したページを開く．ディレクトリを表すときはクリック時に下位構造を表示する．作成した HTML を，絶対位置指定した DIV タグの中に入れる．

特定ユーザとの共有

```
var req;        // XMLHttpRequest Object

// req の定期更新 (間隔: updateSpan msec)
setTimeout("loadXMLDoc()", updateSpan);

/*** メッセージの受信 ***/
function receiveMsg() {
  // ローディングが完了し、ステータスが OK のとき
  if (req.readyState == 4 && req.status == 200) {
    // XML リソースを javascript の配列に変換
    var ary = getXMLData(req.responseText);

    for(var i=0; i<ary.length; i++) {
      kbase[ary[i].name] = ary[i];
    }

    if(kbase.msgs) {
      // 確認メッセージを作成
      var confMsg = makeConfMsg(kbase.msgs);
      var res = confirm(confMsg);
      // 'OK' ならばメッセージを処理する
      if(res) {
        procMsg(kbase.msgs);
      }
      kbase.msgs = new Array();
      saveCondition();
    }
  }
}

/*** メッセージの送信 ***/
function sendMsg(msg) {
  var to = msg.to;
  // メッセージを XML に変換
  var msgXML = js2XML(msg);
  // 送信相手の XML ファイルの更新
  sendSaveCGI(to, msgXML);
}
```

メッセージパッシングを利用して，特定ユーザとのブックマーク共有を行う．メッセージの送信では，

メッセージの内容として登録したブックマークの内容を持たせる．loadXMLDoc では XMLHttpRequest を利用して，利用しているエージェント名と同名の XML ファイルにアクセスする．receiveMsg は req.onreadystatechange イベントで呼び出される．受信側では，新しいメッセージを受け取った場合には，確認ダイアログを表示する．OK が押された時に，受け取ったメッセージを順に処理し，キャンセルされた場合は無視する．最後に受け取ったメッセージを消去する．

全体への共有

ブックマークの内容を HTML ファイルとして書き出しを行い，MiSpider を利用していない Web 利用者からも，登録したブックマークを利用できるようにする．

4.2 クエリエージェント

ユーザが検索サイトへクエリを送信する際に，クエリをエージェントの情報として保存する．保存したクエリは，メニューから選択して，フォームに入力できる．これによって，クエリの再利用性が高まり，ユーザは複数サイトでの検索が容易になる．

実装方式としては，ブックマークエージェントと同様の方法で実装することができる．検索サイトで送信されたクエリを保存することが，ブックマークの追加の部分に相当する．利用するクエリの選択が，利用するブックマークの選択に相当する．選択時にページを開く処理が，選択時にフォームにクエリを挿入する処理に相当する．

4.3 チャット機能

メッセージパッシングを用いて，チャットのようなコミュニケーション手段を容易に提供することができる．従来の Web チャットでは，新しいメッセージを取得してもリロードを行うまで情報が更新されない．MiSpider を利用することで，リロードなしでリアルタイムに情報の更新が行われる．

4.4 MiSpider のインターフェイス

ブラウザの閲覧ページに追加で情報を表示させることは，ユーザのブラウジングの妨げになる恐れがある．そこで，エージェントの表示ボタンを押すことで，情報の表示／非表示の切り替えを行う．また，移動ボタンを押すことで，エージェントの表示位置の移動を行うことができる．また，表示させる項目が多くなり，Web ブラウザの画面上に収まらない場合がある．そのような場合には，一度に表示させる量を絞り，切り替えボタンで表示項目を切り替えるようにしている．

5 おわりに

本論文では，Web 上におけるエージェント環境 MiSpider の実装，および MiSpider に基づくブラウジング支援エージェントの実装について述べた．MiSpider が提供するメッセージパッシング，永続性などの機能を利用することで，効率の良い Web アプリケーション開発ができる．

今後の課題として，サーバへの負荷の問題が挙げられる．本システムでは，クライアント側からサーバに対して定期的に，データの更新チェックを行っているため，利用者の増加に伴い，サーバの負荷が大きくなる恐れがある．更新のチェック間隔を長くすることによって，サーバへの負荷も軽減すると考えられるが，長過ぎてはリアルタイム性が損なわれる．以上のようなトレードオフの調整を行う必要がある．また，送受信する情報に関しては，差分情報のみを送るような設計が必要となる．

他の課題として，セキュリティの問題が挙げられる．現在はメッセージを受け取った際に，メッセージの処理を行うかどうかの確認を行うダイアログを表示させるという簡易な方法をとっているが，さらに考慮を行う必要がある．

参考文献

[1] Y. Fukagaya, T. Ozono, T. Ito and T. Shintani, ”MiSpider: A Continuous Agent on Web Pages ,” In the Proc. of the 14th World Wide Web Conference(WWW2005), pp.1008-1009, May. 2005.

[2] D. Kotz and R. S. Gray, ”Mobile agents and the future of the internet.”, ACM Operating Systems Review, pp. 7 -13, August 1999.

[3] ajax: a new approach to web applications http://www.adaptivepath.com/publications/essays/archives/000385.php

[4] Yahoo!ブックマーク http://bookmarks.yahoo.co.jp/

[5] Book まーく http://www.bookmark.ne.jp/

[6] はてなブックマーク http://b.hatena.ne.jp/