XLink データの登録方法について

大坂　哲司
株式会社フジミック（ＸＭＬコンソーシアム基盤技術部会兼務）

概要:　XLink の第三者リンクは、リンクデータをリソースの外で管理する機構で、HTML のハイパーリンクにはない機能である。本稿では XLink の登録を行うシステムを一般的なブラウザ上で JavaScript のみで実装を試み、その登録データを第三者リンクとして管理する手法を提案する。その結果、XLink 登録を利用して、既存のページから自分専用のページリンクしたコンテンツの方法を見出した。また、ブラウザ上におけるフォントメトリックス、ブラウザ上でマウスドラッグしたテキストの XPointer ロケーション情報取得についても報告する。

XLink Implementation by HTML-DOM Operation

Tetsushi Osaka　(Fujimic Inc.)

Abstract:　The third party link of XLink is the mechanism in which link data is managed outside a resource, and is the function which is not in the hyperlink of HTML. The XLink registration system in this paper is tried on a popular browser, implementation is tried only by JavaScript, and the technique of managing the registration data as a third party link is proposed. Consequently, XLink registration was used and the method of the contents which carried out the page link only for themselves from the existing page was found out. Moreover, fontmetrics on a browser, XPointer location information acquisition of the mouse dragged text on the browser is reported.

１　はじめに

XLink はリソースとリソースを関連付けるリンク機構を有し、W3C は 2001 年 6 月 W3C 勧告として XLink1.0(XML Linking Language)を公開した[XLink] [DuCharme]。XLink は他の ML(Markup Language)とは異なり、属性中心の ML である。そのため、他の ML との親和性が高く、XLink 勧告後に策定された SVG などの XML 規格は、XLink の機能を取り入れているものが多くなっている。XLink によって実施されるリソースの関連付けは、サーバーサイトでも利用できるし、クライアントサイトにおいても利用することできる。XLink が有する多彩な機能をフルに享受するためには、Web ブラウザ上で XLink が作動することが望まれる。しかし、現時点で Amaya など一部のブラウザに XLink の機能が搭載されているが、それらに搭載されている機能は単純リンクを主で、拡張リンクを含む XLink の機能がフルに搭載されているものはまだない状況にある。

我々は本研究に先立ち[大坂]、XLink の有用性を検討するため、一般的なブラウザ上において JavaScript による実装を試みた。その概略は、次の通りである。JavaScript で作成した XLinkAPI は、インターネットエクスプローラ(IE5.0 以上)上で、XLink 仕様に沿って記述した XML ファイルを読み込み、その記述に従い XLink を実現した。XLink 実装に当たり、XLink の連鎖が大きな焦点であったが、XLinkAPI を動的に HTML に埋め込む処理の実現により、リンクの連鎖が可能となった。また 1 対多のリンクや双方向リンクの実現を含め、拡張リンクや第

三者リンクなどの XLink のリンク技術実装を行った。その結果、HTML で表現されている A 要素によるハイパーリンクを超える XLink リンク機構の有効性について報告した。

XLink の大きな特徴として、リンク情報を別途管理することができる第三者リンクが挙げられる。通常の HTML は、文書内にハイパーリンクが書き込まれ、リンク元の文書とリンク先の文書は、それぞれ単独で存在する。リンク元の文書作成時において、リンク先の文書は間違いなく存在していると思われる。しかし、サイトの運用において、リンク先の文書が別フォルダに移動されたり、リンク先の文書が削除されたりすることによって、broken link が生じることがある。リンク情報が一塊になっている第三者リンクは、リンクの管理を容易にする環境を提供するものである。RSS(RDF Site Summary)はリソース自身の管理に優れている[神崎]。これに対して XLink はリソース自身の管理並びにリソース間の関連も管理できるため、参照元がないリソースや参照先がないリソースの発見を容易にサポートすることができる。XLink の第三者リンクの有効性は、このような要求に対応できるものである。

このように XLink は我々に HTML のハイパーリンクを超える有用性をもたらすものであるが、通常のページ作成ツールのように簡単にリンクを埋め込むことができればすばらしい。しかし XLink を簡単に埋め込むことのできる登録・編集ツールは大変少ない[XLink][X2X]。XLink ファイルはエディタを使って直接ファイルを作ることもできるが、リンクの記入ミスを生じることも少なくない。また、XLink では xlink:href で記述されるリソースと xlink:label で示されるラベル情報を一元的に管理することが要求され、対応するシステムも必要となる。本稿では、JavaScript による XLink 実装を通して得られた技術に基づいて、ブラウザ上に表示されたページやその中のテキストをリソースとし、別に表示したページやその中のテキストを関連付ける XLink の登録機能の実装を試み、大きな知見を得た。その詳細について以下の通り報告する。

２　XLink データの登録

XLink ではブラウザ上に開いたページ全体またはそのページに表示されている文字列がリソースとなる。本稿における XLink データ登録のシナリオは次の通りである。２つのフレーム上に開いたページに対して、それぞれのページ上からリソースとなりテキストをマウスドラッグにより選択し、両リソース間のリンク関係、リンク時の表示方法を選択して、XLink データとして登録するものである。この XLink データ登録の開発並びに作動環境は Windows でインターネットエクスプローラ(IE5.0 以上)を用いた。XLink データ登録の実装方法を①ドラッグしたテキストの抽出、②テキストオブジェクトのフォントメトリックス、③フォントサイズとフォント高の関係、④ドラックしたテキストの判定と XPointer 情報取得、⑤XLink データの登録手順について、順を追って説明する。

２．１　HTML 文中からドラッグしたテキストの抽出

ドラッグイベント(ondragstart)を検知し[MSDN1]、そのイベント nodeE を関数 openMenu に渡す。この関数では、マウスイベントが生じたノード node を nodeE.srcElement から取得すると共に、ドラッグした文字列 text を nodeE.dataTransfer.getData(“Text”)から取得する[MSDN2]。ここで用いられた dataTransfer オブジェクトは、drag-and-drop イベントによってクリップボードに格納されたテキストを、getData メソッドを通してそのテキストを取得するものである。これらの処理によって得られた文字列 text が、HTML 文書中のどこにあったか、その座標情報などを知る手段をインターネットエクスプローラは用意していない。但し selection.createRange()を利用し innerHTML テキストの文字列処理により、ドラッグしたテキストの検索も可能である[LL2000]。

ドラッグされた文字列 text は必ずノード node 中にあるが、node 内のどこにあるかは現時点で不明である。文字列 text がノード node 中に１つしかない場合もあれば、複数ある場合も考えられる。前者のケースでは、ノード node 中にある文字列 text を検索し、その文字列に対して修飾処理[大坂]を行えばよい。しかし後者のケースでは、複数行に渡り、複数個の文字列 text があり、どの文字列がドラッグされたものであるか、判定を行う必要がある。

この判定の結果、その文字列に対して修飾処理が行える。この処理を行うために、次節で述べるテキストオブジェクトのフォントメトリックスによる表示位置の座標計算とドラッグしたマウス座標との比較によって可能となる。

２．２　テキストオブジェクトのフォントメトリックス

ブラウザ上に表示されているオブジェクトの座標をすべて把握できなければ、検索で求められた文字列がドラッグした文字列 text であるか否かの判定を行うことができない。表示されているオブジェクトの座標は、ノード node のプロパティやスタイル属性から求めることができる。オブジェクトの開始座標は node.offsetLeft、node.offsetTop から、オブジェクトの大きさは node.offsetWidth、node.offsetHeight から求めることができる。またテキストオブジェクト中の文字列の表示位置は、フォントファミリー、フォントサイズ、フォントスタイル、フォントウェイト、レタースペースなどの CSS スタイル属性から得られる情報を用いて計算することができる。CSS スタイル属性は node.style.fontFamily によってフォントファミリーが、node.style.fontSize によってフォントサイズが取得できる。これら CSS スタイル属性に基づいたテキストオブジェクトのフォントメトリックスが把握できれば、テキストオブジェクト中にある任意の文字列に対して、その表示位置が計算可能となる。

ブラウザに表示されるテキストは見易くするため、プロポーショナルスタイルでそれぞれの文字が表示される。一般に同じ全角文字でも、漢字に対してカタカナやひらがなは、やや小さ目に表示される。また同じカタカナでも文字によって大きさが異なる。ここですべての文字に対して、それらのフォントメトリックスを掌握することは不可能である。本稿では、任意の文字列の表示長（ブラウザ上で表示されているテキストの長さ）を求めるため、非表示でその文字列をブラウザ上にレンダリングし、そのテキストオブジェクトから求めた offsetWidth を表示長とした。フォントメトリックス計算のためのスクリプトコードはリスト１に示した通りである。

```
//フォント情報とテキストtxtからテキストのfontMetrics (ｐｘ) を求める
//HTML上のノードvnodeの下にこれらの情報を追加する
//docsは対象とするDOMを指定する
//フォントサイズfs (ｐｘ) は整数または実数であること
function  fontMetrics(ff,fs,ft,fw,ls,txt,vnode,docs){
    if(!txt)return 0;
        var   mNode=addTextNode("div","","","",vnode,docs);
            mNode.style.visibility="hidden";    //非表示でノードを追加する
            mNode.style.width=fs*txt.length;   //文字列が入る大きさを確保
        var   tNode=addTextNode("span","","",txt,mNode,docs);
            if(ff)tNode.style.fontFamily=ff;      //フォントファミリの設定
            if(fs)tNode.style.fontSize=fs;        //フォントサイズの設定
            if(ft)tNode.style.fontStyle=ft;       //フォントスタイルの設定
            if(fw)tNode.style.fontWeight=fw;    //フォントウェイトの設定
            if(ls)tNode.style.letterSpacing=ls;//レタースペースの設定
        var  len=tNode.offsetWidth;               //文字の表示長の取得
            mNode.removeNode(true);           //追加されたmNodeの削除
        return  len;
}
```

リスト１　フォントメトリックス計算

まず、docs で指定された DOM にあるノード vnode 配下に、DIV 要素を追加し、そのノードを mNode とする。この要素の中にテキストを記入して、そのテキストの長さを求めるため、その表示モードは非表示とする。またテキストが入る十分な幅を確保するため、DIV 要素の幅をフォントサイズに文字列長を乗じた値を採用する。この指定がないと、ブラウザの表示幅を越え 2 行、3 行で表示されるテキストの表示長は、すべてブラウザの表示幅の値となる。この要素の中に、テキストをセットする SPAN 要素を設け、このノードを tNode とする。この要素に対して、フォントメトリックスを求めるテキストのデフォルト CSS スタイル属性をセットし、テキストを埋め込む。このノード tNode の幅は、設定されたフォント情報に従ったテキストの表示長が tNode.offsetWidth から求められる。このコード中の関数 addTextNode はテキストや属性を持つノードの生成が簡単に行える関数である[大坂、野村]。この関数を通して、容易に文字列の表示長を求めることができる。

長い文字列は１行に入らず、必ず改行を伴うものである。改行を含めたフォントメトリックスを考慮しなければ、ドラッグした文字列がそのテキストオブジェクトのどこにあるか判定できないものとなる。この改行に関する情報は調査してもその具体的な情報を得られなかったため、テキストオブジェクトのプロパティから算出することにし

た。複数行に渡るテキストオブジェクトの offsetHeight プロパティを調査したところ、テキストオブジェクトの高さは、1 行のテキストオブジェクトの offsetHeight の行数倍であることが分かった。この結果は、ノーマルな line-height における行ピッチとなるもので、本稿では、１行分の offsetHeight の値をフォント高と呼ぶ。ADDRESS 要素、BR 要素、CENTER 要素、DIV 要素、HR 要素、TR 要素などの要素は改行を伴うもの（これらを改行要素と呼ぶ）であるが、その改行ピッチは各テキストオブジェクトの node.offsetLeft、node.offsetTop の値から、このフォント高と同じ値であった。一方、BLOCKQUOTE 要素、H１～6 要素、P 要素、PRE 要素などの要素は、改行と共に、他のテキストオブジェクトと区別するためのマージン（空白行）が設けられる（これらの要素を改空行要素と呼ぶ)。このマージンはフォントサイズに関わらず、19px であることが判明した。これにより改空行要素の次に出現する要素の縦軸上の位置は、改空行のｙ座標に対して、フォント高＋19px を足したｙ座標となる。また、改空行要素はそのオブジェクトの前後にこの 19px のマージンを有するものであるが、改空行要素が連続している場合、このマージンはダブらないようなルールがあることも判明した。

上述の結果、フォントメトリックスに関するすべての情報が、これで揃った。しかし、ここで１つの問題が生じた。それは HTML-DOM 上で見出すことができる座標の単位である。通常 BODY 要素で指定される単位は絶対単位の PT（ポイント）によって表記されている（1in=72pt)。一方その他の要素では画面解像度に依存する相対単位の PX（ピクセル）によって表記または記述されることが多い。ここで PT と PX 混在の単位系では処理が複雑になるため、PX の単位系に統一した。

２．３　フォントサイズとフォント高の関係

フォントサイズ(単位は px、pt)とフォント高(px)の関係は、リスト２の HTML により、表１に示すような結果を得た。この表に示された関係を解析した結果、リスト３の関数 getFontHeight、getFontSize によって算定可能であることを見出した。

表１　フォントサイズとフォント高の関係

font-size(px)	height(px)		font-size(pt)	height(px)
6	6		6	9
7	7		7	10
8	9		8	12
9	10		9	13
10	11		10	14
11	12		11	16
12	13		12	18
13	14		13	19
・・・・・	・・・・・		・・・・・	・・・・・
37	41		37	55
38	42		38	57
39	43		39	58
40	45		40	59

リスト２　フォントサイズからフォント高への変換 HTML

```
<HTML>
<HEAD><TITLE>フォントサイズ</TITLE>
<script language="JavaScript">
//      px">あい を pt">あい に置換する
function getSize(){
var txt="font-size   font-height\n";
        var node=document.body;
        var child=node.childNodes;
        for(i=0;i<child.length;i++){
                if(child[i].nodeName=="SPAN"){
                        var font_size=child[i].style.fontSize;
                        var height=child[i].offsetHeight;
                        txt+=font_size+"        "+height+"px\n";
                }
        }
alert(txt);
}
</script>
</HEAD>
<BODY style="font-size:12pt; font-family:ＭＳ Ｐゴシック;"
onload="getSize()">
<span style="font-size:6pt">あいうえお</span><br/>
<span style="font-size:7pt">あいうえお</span><br/>
                        ・
                        ・
<span style="font-size:39pt">あいうえお</span><br/>
<span style="font-size:40pt">あいうえお</span><br/>
</BODY>
</HTML>
```

リスト３　フォントサイズ、フォント高の変換スクリプト

```
//フォントサイズ(pt, px)からフォント高(px)を求める
function getFontHeight(fs){
        var fm=fs;
        if(fs.indexOf("pt")>0){//単位がptの場合
                fs=fs.substring(0,fs.indexOf("pt"));
                size=parseInt(fs);
                fm=Math.floor(size*1.5);
                if(Math.floor((size-4)/6)==Math.ceil((size-4)/6))fm--;
        }
        else if(fs.indexOf("px")>0){//単位がpxの場合
                fs=fs.substring(0,fs.indexOf("px"));
                size=parseInt(fs);
                fm=size+Math.floor(size/8);
        }
        return fm;
}

//フォント高(px)からフォントサイズ(px)を求める
function getFontSize(fh){
        return Math.ceil(fh*8/9);
}
```

２．４　ドラッグしたテキストの判定と XPointer のロケーション情報取得

ドラックした文字列 text とテキスト中の文字列の判定の前準備として、ドラッグイベントが生じたブラウザ上の座標 x、y をそれぞれ nodeE.offsetX、nodeE.offsetY から求める。この座標とブラウザ上に表示されている文字列の座標を比較して、ドラッグした文字列の判定を行う。ドラッグイベントを生じたノード node（表示座標 XX、YY で大きさ width、height を有する要素）中に、ドラッグした文字列 text が必ず含まれるが、ドラッグされていない文字列 text も含まれることも考えられる。

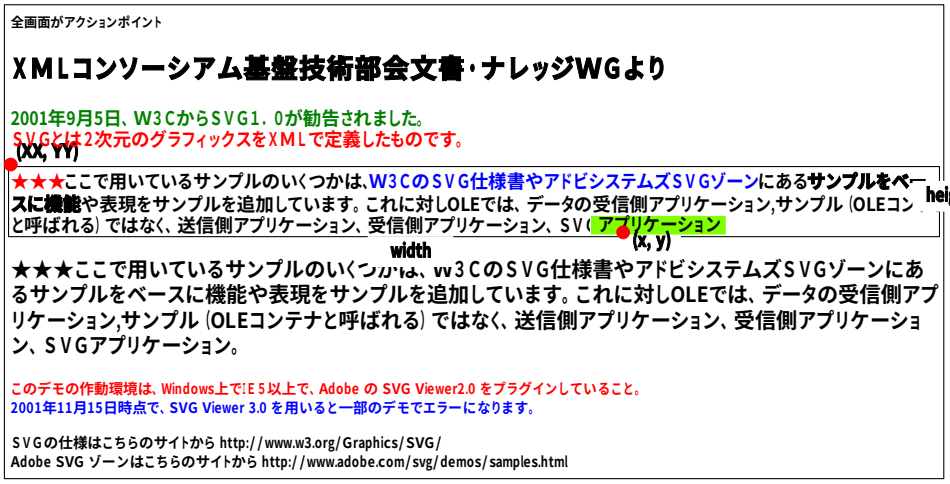

図１　マウスドラッグのサンプル画面

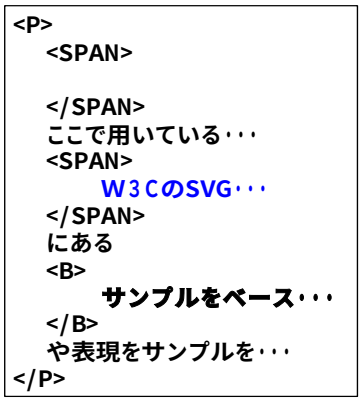

図２　ドキュメントの構造

図１においてドラッグした文字列を“ アプリケーション ”とする（図中で緑色背景箇所の文字列）。図２に示すように、この文字列を含む親ノードは P 要素で、このノードには、3 個の要素と 3 個のテキストノードがある。各テキストノードの検索よって“ アプリケーション ”と言う文字列は、P 要素の 6 番目の子ノードであるテキストノード中にあることが見出される。またこのテキストノードには“ アプリケーション ”と言う文字列が 4 個存在し、この中にドラッグした文字列がある筈である。

テキストオブジェクト中からドラッグした文字列を求める手順は次の通りである。①ドラッグイベントが生じた座標 x、y とドラッグテキスト text（このテキストの表示長を m0 とする）を、並びにイベントが生じたノード node を得て、その node のブラウザ上での開始座標 XX、YY（初期値を X、Y として保存）とその大きさ width、height を取得する。②node が有する CSS スタイル情報を求める（フォントサイズ fs、フォント高 fh の単位は px に統一する）。③node の DOM 構造を読み取り、その子ノードに対して、以下の処理を行う。④子ノードが改行要素であれば、XX=X、YY+=fh として改行を行う。また改空行要素であれば、XX=X、YY+=fh+19 として空行を伴う改行を行う（ただし当該子ノードの 1 つ前の兄要素が改空行である場合、YY-=19 として空行重複を避ける）。⑤ノード node の子ノードが要素であれば、その要素の表示長 len を求め、XX+=len とする。もし XX が X+width を越えるならば、この文字列の前に改行ポイントがある。この文字列までの文字列について二分木検索によって求められた改行ポイントまでの長さを m1 とする。XX-=m1、YY+=fh として改行処理を行う。⑥ノード node の子ノードがテキストノードの場合、そのテキスト中にドラッグした文字列 text が含まれているか、判定する。⑦そのテキストノード中に text が含まれない場合、④と同様の処理を行う。⑧そのテキストノード中に text が含まれる場合、まず m2=XX とする。⑨その文字列を含む文字列の表示長を m2 に加える。⑩もし m2-m0 が X+width を越えるならば、この文字列の前に改行ポイントがある。改行ポイントまでの長さを m1 とする。m2-=m1、YY+=fh として改行処理を行う。⑪ドラッグした位置 x、y がそれぞれ m2-m0～m2、YY～YY+fh の範囲内にあるか、判定を行う。⑫こ

の判定に適合しない場合、次のテキスト候補に対して⑨の処理を行う。もしすべての候補が判定に適合しない場合、XX=m2 とし③の次のノードへ進む。⑬この判定に適合すると、この文字列はドラッグした文字列そのものとなり、この文字列に修飾処理を行う。このドラッグ文字列が 2 行に渡って折り返し表示されている場合、その上の行、下の行に分割表示されている文字列に対して、それぞれ部分マッチングによる判定を実装した。

以上の結果より、図１にあるようにドラッグした文字列“アプリケーション”は、上述の処理によりドラッグ座標 x、y は文末にある“アプリケーション”の表示座標中にあること分かり、この文字列に対して修飾処理を行うことができた。この修飾処理と連動して XPointer 情報を取得し、フレーム menu 中のリソース欄に XPointer の string-range 関数で表されるリソース名、及びタイトル欄にドラッグしたテキストが表示される。

現在、上述の改行処理において、半角文字列に見られるワード単位での改行ルールや句読点などの禁止則への対応はなされていない。

２．５　XLink データの登録手順

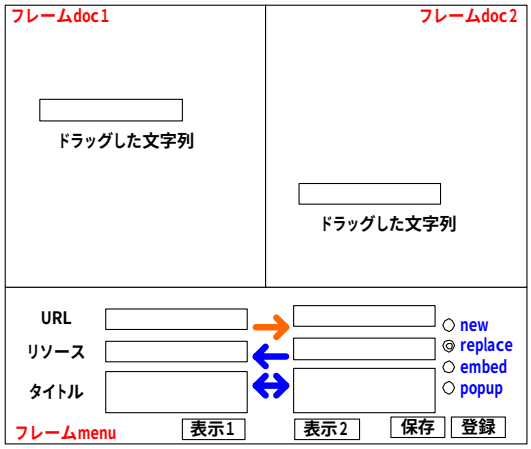

図3　XLink登録画面

登録の流れは次の通りである。まず XLink 登録用 HTML をアクセスすると図 3 が表示される。ブラウザ上に３つパートからフレームが表示される。それらの内、ブラウザ下部に表示されるフレーム menu は、XLink 登録を制御するフレームである。ドキュメントを表示するフレーム doc1、doc2 がブラウザ上部に並んで表示される。フレーム menu の左側の URL 欄に参照するページの URL を入力し、ボタン” 表示１”をクリックすると、フレーム doc1 に指定したページが表示される。表示されたページ中のテキストをドラッグすると、リソース欄に xlink:href に対応する Xpointer 形式の URI が表示される。またタイトル欄は xlink:title に対応するもので、これらの記入により、１つのリソース（このリソースをリソース１と呼ぶ）の記述が完了する。もう１つのフレーム doc2 に同様の操作を行い、リソースを定義する。ここで定義したリソース間の関係をフレーム menu 中央の矢印によって指定する。ここで指定できる関係は１方向並びに双方向が指定できる。更に参照表示形式を同フレーム右側のチェックボックスにより選択する。これらの指定によりリソース１からリソース２へのリンク関係とリンク時の表示形式が定まり、これらのデータはボタン“保存”により、フレーム menu 内にある XLink データ(DOM)に保存される。更に続けて別のリソースに対しても同様の処理を行う。最後にボタン“登録”によって XLink データはサーバーに登録される。これらが XLink 登録処理の流れである。

この処理における大きなポイントは、①HTML 表示完了の検知、②表示完了後、表示された HTML 中へのマウスドラッグ処理を行うスクリプトを埋め込み、③XLink データの生成（xlink:label 値の生成）である。①②は、フレーム doc1、doc2 に表示される HTML には、そのテキストをドラッグするなどの処理機能はない。この処理機能を埋め込むために、表示完了検知、スクリプト埋め込み処理は必要である [大坂]。③における留意点は xlink:label の値が他のリソースと明らかに識別できること、同じリソースに対して重複しないことである。この留意点に対して、xlink:label 値は、登録画面の前に行うユーザー認証のユーザーID と日付 yyyymmddhhmmsstt を 1 つにしたユニークな値とし、xlink:href 値とペアで管理した。

3　実装の評価結果と制限

本稿における XLink データ登録はブラウザだけの処理では完結しない。HTML などのコンテンツはサーバー上にあり、XLink データもまたサーバー上に登録される。XLink データはユーザー認証に基づき、ユーザー所定のフォルダに XLink データが保存される。図 3 の登録画面によって 2 つのリソースに対してリンクを定義し、その XLink データをブラウザからサーバーへ、第三者リンクを定義される XLink ファイルとして保存される。参照時この XLink ファイルを参照画面にロードし、その XLink データに従ったリンクが連鎖し、対応する HTML を表示させることができた。また第三者リンク形式を利用して、既存のページを基にして新たに関係付けられた自分専用のコンテンツが出来上がった。この結果は、リストとして並べられているブックマークと比べ、大変使い勝手のよいものであった。

XLink ファイルをエディタで編集し、リンク関係を壊して第三者リンクによる broken link 検出の有効性を調べた。本稿では、リソースからリソースリストを作成し、リソースとラベル値をペアで管理している。このリソースリストに従い、アーク情報を求める。アーク情報中の xlink:from、xlink:to の値がリソースリストのラベル値にないアークは、broken link とし、アークリストに加えない。このようにして出来上がったアークリストは存在するリンクしかない状態となる。この過程で、リソースがおかしいもの、アークがおかしいものが検出する手段を見出し、broken link の管理可能であることが示唆できた。但し、リソースそのものの存在は別途サーバープログラムによる検証が必要である。

本稿では JavaScript のセキュリティによる大きな制限がある。図 3 の上で作動する XLink 登録用 API が組み込まれていないドメインへのアクセスはできず、利用の範囲は自分達が管理するイントラネットなどに限られる。また図 3 中で存在する URL が正しく入力される場合、問題が生じない。しかし誤った URL を入力するとエラーコード 404 で生じる表示エラーが表示され、以後そのフレームは制御不能となる（HTTP 通信を伴わないローカルファイルの表示処理ではエラーリカバリーが可能）。これらの制限は ActiveX コントロールを用いると回避できるものであるが[LL2000]、JavaScript のみでの実装を試みようとする本稿では、ActiveX による処理は利用しなかった。

4　応用事例（アノテーションの登録）

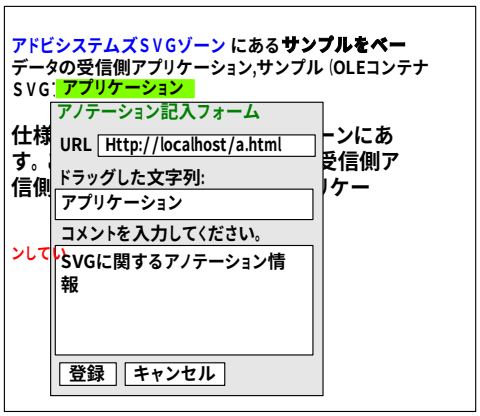

図4　アノテーションの登録画面

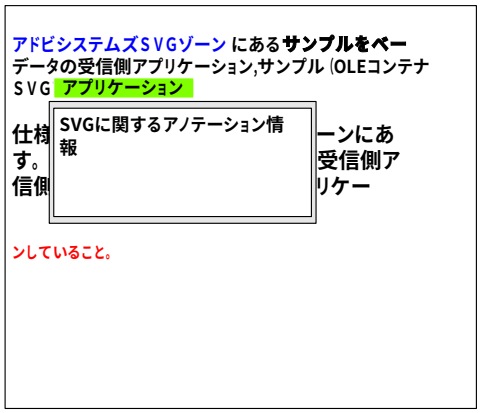

図5　アノテーションの参照画面

XLink の応用事例としてアノテーションがある。ユーザーはページを見ているとき、そのページ中で気に入った箇所、疑問を抱いた箇所などに対して、自分なりのアノテーションを付与できるようにするため、図 4 に示すように機能を用意した。画面上でアノテーションを付与するテキストをマウスドラッグする。この操作によって、ドラッグした文字列にテキスト修飾（ここでは緑色の背景色を付与）を施し、アノテーションを記入する画面が HTML 上にポップアップ表示される。ポップアップ画面に表示される URI は、表示し

ているページの URL を document.location.href により取得したものと、ドラッグしたテキストのページ上の位置を Xpointer の string-range で記述したものと合わせたもので、これが XLink のリソース元となる。コメント記入欄に記入したものがドラッグしたテキストへのアノテーション情報となる。この入力テキストがローカルリソースと定義され、リンク先のリソースとなる。ボタン"登録" によってアノテーションが登録される。登録されるデータはリンク元となるリソースとリンク先となるアノテーション情報である。このリソース元からリソース先へのリンクを xlink:type="arc"でアークを記述し、アノテーション表示のため、xlink:show="popup"と言うポップアップ表示モードを用意した。アノテーション情報は XLink データの一部として管理される。登録されたアノテーション情報は、XLink 表示できる環境を有するブラウザにおいて、図 5 のようにリソース元となるテキストをクリックすると、ポップアップ画面に表示される。なお、マウスフォーカスがこのポップアップ画面から外れたとき、この画面は消去するにようにした。

5　考察と今後の展望

本稿では JavaScript のみを用いて、一般のブラウザ（ここではインターネットエクスプローラを用いた）上で、XLink 登録処理を提案した。当初 XLink 登録機能に終始していたが、XLink データの生成が自由にできるようになったことによって、XLink がもたらすであろうところの大きな変化、新しい流れを実感することができた。そのポイントは次の３つである。１つ目は、XLink は RSS(RDF Site Summary)などの手段と組み合わせるとリソース並びにリンクの強力な管理が可能となること。２つ目は、第三者リンクの利用によってマイ・コンテンツの作成が可能となること。３つ目として、ページに対してメモやコメントなどを付けたり、ページを客観的に評価できるアノテーション[Annotea]による新たなコンテンツ管理が可能なことが挙げられる。これらの内で特に 2 つ目の機能は、我々に新しい情報の整理方法を提供するもので、今後、サーバーサイトのプログラムと組み合わせ本格的なシステム構築を行いたいと考えている。

参考文献

[Annotea]　“Annotea: AN Open RDF Infrastructure for Shared Web Annotations ”、
http://www.w3.org/2001/Annotea/Papers/www10/annotea-www10.html、2001

[DuCharme] “XLink: Who Cares? ”、http://www.xml.com/pub/a/2002/03/13/xlink.html、2002

[LL2000]　Laurent Denoue & Laurence Vignollet “An annotation tool for Web Browsers and its applications
to information retrieval ”、http://www.univ-savoie.fr/labos/syscom/Laurent-Denoue/riao2000.pdf、2000

[MSDN1] “ondragstart Event ”、 http://msdn.microsoft.com/workshop/author/dhtml/reference/events/ondragstart.asp

[MSDN2] “About DHTML Data Transfer ”、http://msdn.microsoft.com/workshop/author/datatransfer/overview.asp

[MSDN3] “How To Create a Mouse Capture Context Menu ”、
http://msdn.microsoft.com/workshop/author/dhtml/howto/mousecapturecm.asp

[X2X] “empolis x2x™ v2.0 ”、http://www.stepuk.com/x2xdocs2/index.html

[XLink] “XML Pointer, XML Base and XML Linking ”、http://www.w3.org/XML/Linking、2001

[大坂、野村] ”SVG-DOM によるアニメーションと XHTML 中心複合文書の可能性 ”、
情報処理学会研究会報告、FI-66、DD-32、3/2002

[大坂] ”JavaScript を用いた XLink の実装方法について ”、情報処理学会研究会報告、DD-35、9/2002

[神崎] “RSS -- サイト情報の要約と公開 ”、http://www.kanzaki.com/docs/sw/rss.html、2001