Keio WIX システム（2）

サーバーサイド実装

森　良介† 藪　達也† 朱　成敏† 遠山　元道†

† 慶應義塾大学大学院理工学研究科　〒 223-0061 横浜市港北区日吉 3-14-1
E-mail: †{morisi,yabu,joo}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

あらまし　近年，Wikification システム等，Web ドキュメント中に存在する単語に対してリンクを生成するシステムが登場している．しかし，これらのシステムにおいて生成するリンクは同一ドメインのもののみであり，ユーザが得る情報が限定されてしまうといった問題点がある．これに対し本研究では，ユーザ主体の Web 資源結合を実現する Web Index システムの方式設計，実装を行った．これによって先行研究の課題であった，Web ドキュメントに対するドメインにとらわれない Web 資源結合サービスが実現された．更に，本研究の実装中において採用した辞書式文字列であるマッチング手法 Aho-Corasick 法に対する動的追加更新手法により，既存手法のおよそ 35 倍の速度で辞書語追加更新処理を行うことを示した．

キーワード　Web，ハイパーリンク，セマンティックアノテーション，辞書式キーワードマッチング, Aho-Corasick 法

Ryosuke MORI†, Tatsuya YABU†, Sungmin JOO†, and Motomichi TOYAMA†

† Graduate School of Science and Technology, Keio University
3-14-1 , Hiyoshi, Kohoku, Yokohama 223-0061 Japan
E-mail: †{morisi,yabu,joo}@db.ics.keio.ac.jp, ††toyama@ics.keio.ac.jp

1. 背　　景

検索エンジンの普及によって人々は日常的に Web を利用し情報検索を行うようになった．その上でユーザは検索エンジンによって必要な情報資源を検索し，結果の Web 文書中から必要な情報を得る。Web 上では関連する情報はハイパーリンクで結合され，リンク元の文書とリンク先の文書間の関連はホームページ作成者の意図した関係のみで提供されている．このため、ユーザが Web 文書中で見つけた単語や名称を元に新たな情報を得たいという要求が生じた場合，ユーザはさらにその単語を検索エンジンなどにかけなければならない．例えば，閲覧中の文書に含まれる全てのスポーツ選手の詳細情報を知りたいという要求が生じた場合，その選手の名前を検索エンジンで検索し，欲しい情報を得る作業を繰り返し行わなければならない．

近年，Web2.0 の広がりによって多くのユーザが「集合知」を形成し，利用するといった Web の利用形態が一般化してきた．その代表例である Wikipedia やはてな，ESPN においては文書中の単語がそれに対応する記事へリンクされており，ユーザの検索負担が軽減されている．また、Wikify! [1] のように Web ドキュメントに Wikipedia へのリンクを自動生成する Wikification システムも登場している．

しかし，これらのシステムにおいて生成するリンクは同一ドメインのリンクのみであり，ユーザが得る情報が限定されている．例えばスポーツ選手の情報を欲するユーザであっても，時と場合によって選手成績や履歴など，異なる情報を必要とする場合があると推測される．このため，これらのシステムは必ずしも閲覧者のニーズに応じた情報を結合し，提供できているとは言えない．

その一方で，本 Web Index（WIX）システム [4] [5] [6] においては関係データベースの考え方を Web の世界に持ち込むことによって，ドメインにとらわれない，ユーザ主体の Web 情報源の結合を図っている．関係モデルでは，それ以前のデータモデルがアドレス／ポインタによって行っていた関連付けを，値に基づいて主キー，外部キーで実現した．現在の Web におけるアンカー／ URL は関係モデル以前の DB モデルに酷似している．そこで WIX ではアンカーテキストとリンクを Web ドキュメントから独立した「キーワードとリンク先の集合」（WIX ファイル）という情報源として保存し，それを適宜ドキュメントに対して結合（アタッチ）を行うことによってリンクの生成を行う．WIX は先行研究によってクライアントサイドアプリケーションとして，また，サイト作成者支援システムとして実現されてきた．

2. 目　　的

本研究においてはサーバーサイドにおける WIX のシステム方式設計とその実装を行う．これによって先行研究の課題であった，一般ユーザが利用可能な Web ドキュメントに対するドメインにとらわれない大規模な Web 資源結合サービスを実現する．

更に，本研究の実装中において採用した辞書式文字列であるマッチング手法 Aho-Corasick 法 [2] に対する動的追加更新手法の提案を行う．

3. アタッチ処理

まず，システムの Web 資源結合動作であるアタッチ処理について述べる．

3.1 FSDR 処理

アタッチ処理は以下の 3 フェーズに分けられる．この一連の流れを *FSDR* 処理とする．

- *Find*
- *Select*
- *Decide*
- *Rewrite*

まず Find では Web 文書と WIX ファイル集合を入力として，全 WIX ファイル中の全てのエントリのうち Web 文書中に存在するキーワードをもつエントリを文書中の出現位置とセットにして返す．このセットの集合のことを Find 結果と呼ぶ．続いて Select に Find 結果を入力する．Select では WIX ファイルやユーザ情報を管理するデータベースに問い合わせを行い，Find 結果のうちユーザがブックマークしているエントリを選択する．次の Decide では Select 結果から更にキーワードの周辺文字列を利用してより Web 文書のトピックとマッチするエントリを抽出 [6] するなどして，最終的にハイパーリンクに書き換える箇所を決定する．この Decide 結果を用いて入力 Web 文書を新たなハイパーリンクのついた Web 文書に書き換える（Rewrite）ことによってアタッチが完了する．

4. アーキテクチャ

実際に構成したシステムのアーキテクチャを図 1 に示す．

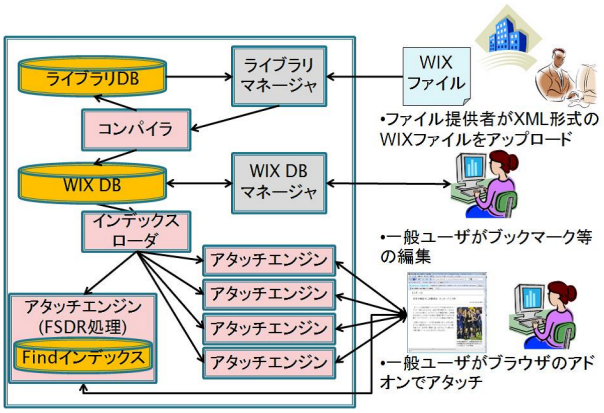

図 1　WIX アーキテクチャ

本システムでは WIX ファイルの情報をライブラリ，WIX DB，Find インデックスの 3 つの異なる形態で管理する．これら 3 形態の比較概念図を図 2 に示す．

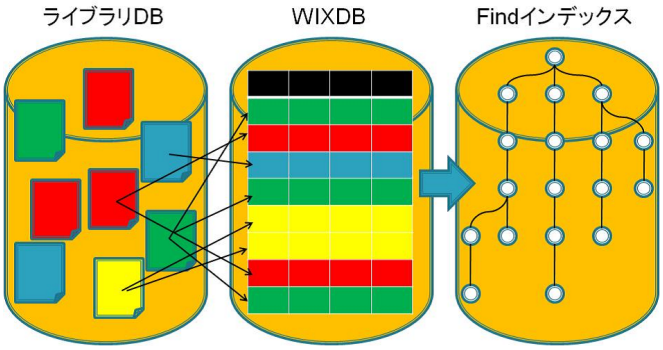

図 2　形態比較概念図

4.1 WIX ライブラリ

ライブラリにおいては，全ての WIX ファイルの全バージョンを WIX ファイルという XML 形式で保存する．（図 2 左）WIX ファイル例を図 3 に示す．WIX ファイルはキーワード（keyword）とそれに対応する文書（target）の組み合わせからなるエントリの集合である．

```
<wix>
 <header />
 <body>
  <entry>
   <keyword>細貝萌</keyword>
   <target>http://ja.wikipedia.org/…</target>
  </entry>
  <entry>
   <keyword>川島永嗣</keyword>
   <target>http://ja.wikipedia.org/…</target>
  </entry>
  <entry>
   <keyword>アウクスブルク</keyword>
   <target>http://ja.wikipedia.org/…</target>
  </entry>
  …
 </body>
</wix>
```

図 3　WIX ファイル例

このような XML 形式を用いることにより，ファイル製作者や外部プログラムとのデータの受け渡しが容易となる．ファイル作成者はライブラリマネージャを通じて GUI から WIX ファイルのアップロード，アップデートを行う．ライブラリで管理している過去バージョンの WIX ファイルは，ライブラリマネージャを通じてダウンロード可能である．ライブラリにおいては WIX ファイルを XML 形式で保存しているため，ファイル単

位での情報管理となっている．しかしながらアタッチにおいては全ての WIX のエントリに対して辞書式キーワードマッチングを行わなければならない．故に WIX ファイルの各エントリを一括管理するため，WIX ファイルをエントリ単位に分解し格納する WIX DB が必要となる．

4.2 WIX DB

WIX DB においては，ライブラリで管理している WIX ファイルのうち，最新バージョンの実際にアタッチに用いる WIX ファイルを 1 エントリ毎バラバラに、RDB のタプルとして管理する．（図 2 中央）即ち，ライブラリではファイル単位で格納していた WIX ファイルを，ここではエントリごとにバラバラにしてまとめて管理する．また，ユーザ情報，ブックマーク情報等，システムの機能に関する情報も WIX DB にて管理する．それ故，WIX DB にはユーザ管理テーブルやブックマークテーブルなど多くのテーブルが存在するが，ここでは WIX ファイルの情報を扱う wix テーブルと entry テーブルについて述べる．

wix テーブルにおいては WIX ファイル自体の情報管理を行う．（表 1）WIX ファイルの id である *wid*，一意名である *w_domain*，WIX ファイルのバージョン番号である *version* や登録時間 *regtime*，その他ヘッダに記載されたファイル情報等を属性として持つ．ユーザが作成したそれぞれのブックマークには WIX ファイルが対応しており，*wid* を用いて，Select 処理におけるブックマークの情報の絞り込みを行う．

表 1　wix テーブル

wid	w_domain	version	…
1	//toyamalab.alpha/wikipedia.wix	3	…
2	//mori.beta/baseball.wix	1	…
3	//mori.beta/samuraiblue.wix	1	…
4	//toyamalab.alpha/progressive.wix	1	…
5	//toyamalab.alpha/longman.wix	1	…
…			

entry テーブルにおいては WIX ファイルのもつエントリ情報の管理を行う．（表 2）そのエントリが所属する WIX ファイルの *wid*，エントリの id である *eid*，辞書語となるキーワードの *keyword* とそれに対応する URL である *target* 等を属性として持つ．

表 2　entry テーブル

wid	eid	keyword	target	…
1	1	細貝萌	http://ja.wikipedia…	…
1	2	川島永嗣	http://ja.wikipedia…	…
1	3	アウクスブルク	http://ja.wikipedia…	…
2	1	イチロー	http://web.51channel.tv…	…
3	1	細貝萌	http://samuraiblue.jp/…	…
3	2	川島永嗣	http://samuraiblue.jp…	…
…				

4.3 WIX コンパイラ

XML 形式で WIX ファイルを保存したライブラリから，エントリ単位にバラバラにして情報を取り扱う WIX DB の間には形式を変換するコンパイラが必要となる．本 WIX コンパイラは XML 形式の WIX ファイルに対しスキーマ検証を行い，合致したものに対してその情報を WIX DB に挿入するという処理を行う．

今回構築したコンパイラは，ユーザがアップロードした WIX ファイルが新規ファイルであった場合，WIX ファイルをパースして CSV を作成し，この CSV から WIX DB に一括挿入を行う．これに対し，ユーザがアップロードした WIX ファイルが更新ファイルであった場合，WIX ファイルをパースして CSV を作成した後旧バージョンの CSV と TextDiff を取り，差分のみの更新処理を行う．これによって，Web サービスの課題となる DB I/O を削減することが可能となる．

4.4 Find インデックス

Find インデックスにおいては，WIX DB の entry テーブルからエントリ情報をメモリ上に展開する．（図 2 右）本研究においては Aho-Corasick 法における辞書式マッチング処理のオーダーが，その辞書の大きさに関わらず入力文字列長のみに依存することを最大限に活用すべく，WIXDB 内の全エントリ情報から 1 つの大きな Find インデックスを構築している．Find インデックスについての詳細を次章にて詳述する．

5. Find インデックスと動的構成手法

Find 処理では高速化のために WIX の見出し語（keyword）を辞書とする辞書式文字列マッチングを行う．現在辞書式文字列マッチングアルゴリズムとして有用な手法がいくつか提案されているが，本研究では Aho-Corasick 法（以下 AC 法）を用いた辞書式文字列マッチングを行うことにする．

5.1 AC 法とその問題点

AC 法は任意のテキストから辞書に含まれる語とその出現位置をすべて抽出する辞書式文字列マッチングアルゴリズムの一種であり，辞書からパターンマッチングを行うオートマトンを構築し，入力テキストのサイズに対して線形な計算時間を実現する手法である．以下この AC 法に基づくオートマトンを AC マシンと呼ぶ．辞書 $D = abcc, abcde, bcd, ca$ からなる AC マシンを図 4 に示す．

AC マシンは GOTO 関数，Failure 関数，Output 関数の 3 つの関数から構成されている．Goto 関数は節点 s から遷移種 a による遷移先節点 t を返し，$Goto(s; a) = t$ と表す．Failure 関数は Goto 関数が fail を返した際に呼び出され，遷移失敗時における節点 s からの遷移先節点 f を返し，$Failure(s) = f$ と表す．ここで Failure 遷移先の節点 f は節点 s が表す文字列の最長接尾辞を表す節点とする．以下 Goto 関数，Failure 関数による遷移をそれぞれ Goto 遷移，Failure 遷移と呼ぶ．OUTPUT 関数は GOTO 遷移により到達した節点 s において検出する語集合 O_s を出力し，$Output(s) = O\_s$ と表す．図 4 においては実線が Goto 遷移，破線が Failure 遷移を表す．また，二重丸の節点が Output 集合をもつ節点であり， 括弧内の数値は

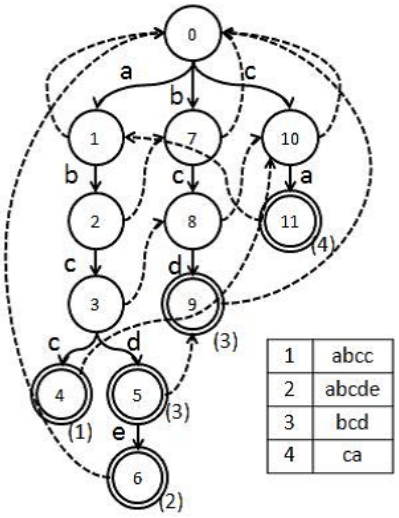

図 4　AC マシン

Output 集合のインデクスを表す．図 4 内の表はインデクスに対応する Output 集合を表す．AC マシンは Goto 遷移に失敗した際に Faillure 関数を呼び出すことによって入力テキストをバックトラックせずに照合を行うことができるため，代表的な辞書式文字列マッチングの手法であるトライ法に比べ高速な文字列マッチングが可能である．

ここで WIX が主にキーワードとして想定しているものは人名や地名，組織名，商品名などの固有名詞である．固有名詞の辞書は一般名詞のような，一度辞書オートマトンを構築すれば長期間更新なしで使用可能なものとは違い，日々新しい語が増え続けるため頻繁な更新が余儀なくされることが想定される，以上から辞書オートマトンのインクリメンタルな更新ができることが WIX における辞書式文字列マッチングマシンの前提条件となる．

しかしながら AC マシンは元々辞書の更新に対応しておらず，辞書に語が追加される際にはオートマトンを一から再構成する必要があった．そこで津田らは動的に AC マシンを構成する手法 [3] を提案している．

5.2　AC マシンの動的構成法とその問題点

AC マシンに新たな語が追加された場合，追加文字列をテキストと考え一文字ずつ読み進めながら既存のオートマトンを Goto 遷移していく．もし節点 s からの遷移先節点が存在しなければその時点で，Goto 関数と節点 s における Failure 関数の生成を行う．ここまではインクリメンタルな追加が可能であるが，問題は既存の節点（以下既存節点とする）から，新しく追加された節点 (以下新規節点とする) に対する Failure 関数の更新である．ここで辞書語の追加処理の様子を図 5 に示す．

図 5(a) は辞書 $D = abcc, abcde, bcd, ca$ を基に構築した AC マシンである．ここに辞書語”e”を追加することを考える．まず, 初期節点 0 から遷移種’e’ で Goto 遷移を行おうとするが，遷移失敗となるため新たに $Goto(0;'e') = 12$ を定義する．ここで節点 12 が表す文字列”e”は辞書語であることから，節点 12 に Output 関数を定義する． また節点 12 は深さ 1 に存在して

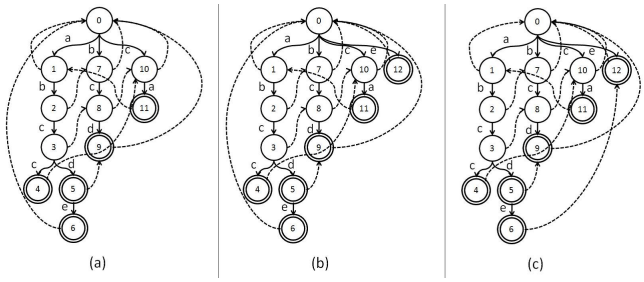

図 5　辞書語の追加処理

いるため，Failure 関数を初期節点に定義する．ここまでは逐次的な追加更新が可能である．この状況が図 4(b) である．しかし図 5(b) は完全な AC マシンとは言えない．なぜならば新たに生成された節点 12 は，節点 6 が表す文字列 ”abcde ”における接尾辞の中で，オートマトン中に存在している最長の接尾辞を表す節点となったため，節点 6 の Failure 遷移先を節点 12 に定義しなおす必要があるからである．さらに言えば節点 6 は，Failure 遷移先である節点 12 の Output 集合を含む必要があるため，Output 関数の更新処理も必要となる．これらの処理を行った結果が図 5(c) である．即ち AC マシンにおける辞書語の追加更新問題は，節点 12 の生成に伴い，どのようにして更新が必要となる節点 6 を見つけるか，という問題に帰着する．

ここで津田らの提案する AC マシンの辞書語の追加更新手法について述べる．以下この手法を既存手法と表記する．既存手法が AC マシンの辞書語追加更新処理に必要とするのは Failure 関数の逆関数のみである．これを逆 Failure 関数と呼び, 節点 s の Failure 遷移先節点が f, 即ち $Failure(s) = f$ である時の逆 Failure 関数を $R\_Failure(f) = s$ と表わす. ここで新規節点 X, 既存節点 A，B に対して，$Goto(A; a) = X$, $R_Failure(A) = B$, が定義されていたとすると，$Goto(B; a) = t$ による遷移先節点 t がもし存在するならば，節点 t は Failure 関数の更新が発生し，$Failure(t) = X$ が一意に決定する．こうして新規節点を生成しながら Failure 関数の更新をインクリメンタルに行うことができる．しかしこの方法では新規節点の親が非常に多くの逆 Failure 関数を持っていると，その全ての節点が更新の候補となってしまうという問題点がある．そこで本研究では予めオートマトン更新のため新たに Expect 関数を持たせた更新手法を提案する．

5.3　提 案 手 法

本手法では予め Failure 関数の更新を期待する節点一覧を転置インデックスとしてもたせておく．例えば 5(a) のように現在「abcde」を表す節点 6 の Failure 遷移先が初期節点 0 に定義されていたとする．この時もし「e」や「de」,「cde」,「bcde」を表す節点が生成されればこれらの節点は節点 6 にとって最長の接尾辞となる．そこで節点 6 はこれらの節点の生成を期待していると考え，$Expect(”e”) = 6$, $Expect(”de”) = 6$, $Expect(”cde”) = 6$, $Expect(”bcde”) = 6$ で表される Expect 関数を定義しておく．また節点 4 の場合では，節点 4 の表す文字列”abcc”と節点 4 の Failure 遷移先である節点 10 の表す文字列”c”との間には”cc”，”bcc”という接尾辞の差分が存在する．そこでこの差分

を $Expect(\text{"}cc\text{"}) = 4$, $Expect(\text{"}bcc\text{"}) = 4$ というように定義する．ここで図 5(a) における節点 1,2,3,5,7,8,10,11 らは Failrue 遷移先節点がそれぞれ最長の接尾辞を表す節点となっている．このような Failure 遷移先をもつ節点にとってはそれ以上長い接尾辞は無いので Expect 関数の定義はしなくて良い．以上のように Expect 関数を定義しておくことで，もし新たに「e」を表す節点が生成されたら，$Expect(\text{"}e\text{"})$ を参照することで，即座に Failure 関数更新対象となる節点 6 が見つかることになる．

6. 評価実験

6.1 実験条件

本研究の評価として，Find インデックスに採用した提案手法を用いた AC マシン (以下マシン A) と，既存手法による更新機能を持つ AC マシン (マシン X) の比較を行う．これらの AC マシンの Goto 関数は連想配列により実装されている．また辞書式文字列マッチングの処理速度の評価に関しては，Goto 関数をダブル配列とほぼ同様の操作で節点間遷移を実現する AC マシン (マシン B) と，茶筅や Mecab などの日本語形態素解析エンジンに採用されているダブル配列データ構造を用いたトライである Darts との比較を行う．

データセットとしては日本語版 wikipedia の記事タイトル集合を用いた．wikipedia は URL「http://ja.wikipedia.org/wiki/」の末尾に記事タイトルを挿入することによって, その記事自身を表す URL を生成することができるため記事タイトル集合から容易に WIX ファイルを作成することができる．また，wikipedia は一般名詞の他にも我々が主な WIX エントリとして想定している固有名詞を網羅的に扱っているため，本研究の評価実験に非常に適していると考えられる．そこで日本語版 Wikipedia の記事タイトル集合から 30 万件を抽出し，記事タイトルをキーワード，記事 URL をターゲットとする WIX ファイルを作成した．これに対し，追加更新用に記事タイトルに重複が起こらないよう新たに 2 万件の記事タイトルを抽出した WIX ファイルを作成した．また入力テキストとしてはおよそ 32000 文字 (74KB) の WEB 文書を用いた．これらのデータセットを用いて Find 時間の比較，メモリ使用量の比較，オートマトン構築時間の比較，追加更新時間の比較を行った．

6.2 実験結果

実験結果をまとめたものを表 3 に示す．

表 3　実験結果のまとめ

	Find 時間 (msec)	メモリ量 (GB)	構築時間 (sec)	更新時間 (sec)
マシン A	32	2.5	60	1.3
マシン B	18	3.6	130	3.9
マシン X	32	2.0	35	45.1
Darts	22	1.0	93	105

Find 時間はマシン B が最も高速であるが，マシンの構築時間は最も長い．ただし Find 処理を行う頻度が AC マシン構築の頻度を大きく上回ることが想定されるため, 構築時間の長さは大きな問題ではないと考える．またアルゴリズムとしてはトライ法よりも高速なはずのマシン A だが，Darts のほうが Find 時間は短い．これは Darts における GOTO 遷移もマシン B 同様配列の値を参照するだけで実現しており，連想配列を用いたマシン A との実装の差が表れていると考えられる．構築時間に関しては Darts が最も長いが，これはトライ構築前に辞書語を辞書順にソートする必要があるためである．更新時間に関しては既存手法に比べ提案手法はおよそ 35 分の 1 の時間で済み，提案手法の有用性を示した結果となった．なお Darts には更新機能が無いため更新時間は全件構築時間となっている．

7. まとめ

本研究ではサーバーサイド Web Index システムの方式設計と実装を行った．更に，各節点における逆 FAILURE 関数の数に依存して更新対象となる節点の発見に手間がかかる AC 法における既存の辞書語追加更新手法に対し，新たに EXPECT 関数を用いた更新手法を提案した．

これによって Web ドキュメントに対するドメインにとらわれない Web 資源結合サービスが実現すると共に，実験の結果から，30 万語の辞書語からなるオートマトンに 2 万語を追加した際に提案手法は既存手法のおよそ 35 倍の速度で辞書語追加更新処理を行うことを示した．また，Find 処理時間も茶筅や Mecab などの日本語形態素解析エンジンに採用されている Darts よりも短い結果となることを示した．

文　献

[1] Rada Mihalcea, Andras Csomai. Wikify! Linking Documents to Encyclopedic Knowledge. In *Proceedings of ACM CIKM '07 International Conference on Information and Knowledge Management*, pp.233-242, 2007.

[2] Alfred V. Aho, Margaret J. Corasick. Efficient String Matching:An Aid to Bibliographic Search In *Commun.ACM*, *18*, *6*, pp.333-340, 1975.

[3] 津田　和彦, 入口　浩一, 青江　順一. ストリングパターンマッチングマシンの動的構成法. In 電子情報通信学会論文誌, *VOL. J77*, *NO.4*, pp.282-289, 1994.

[4] 佐藤　裕紀, 遠山　元道. A-doc ファイルのアタッチの機能を持つ専用ブラウザの試作. データ工学ワークショップ, *DEWS2007*, 2007.

[5] 朱　成敏, 遠山　元道. A-doc に基づく閲覧者主導のハイパーリンク支援. データ工学ワークショップ, *DEWS2008*, 2008.

[6] 渋谷　雄一, 遠山　元道. Web 文書の特徴語を利用した Web Index(WIX) アタッチ支援. データ工学ワークショップ, *DEIM2009*, 2009.