

## ラッピング言語を用いた Web サイトの再構築手法の提案

石井 悠太<sup>†1</sup> 森嶋 厚行<sup>†1,†2</sup>  
杉本 重雄<sup>†1,†2</sup> 北川 博之<sup>†3</sup>

現在, Web サイトを通じた情報発信が普及しているが, それらの維持管理にかかるコストを大きくかけられる組織はそれほど多くない. 維持管理を効率化するためにはバックエンドに DB を持つシステムを構築することが一般的である. しかし, その構築コストは大きいため, 多くの Web サイトは, バックエンドに DB を持たない静的な HTML ページだけの Web サイトであることが多い. 本論文では, そのような静的な Web サイトを, バックエンドに DB を持つシステムに再構築するための手法を提案する. 本手法のアイデアは, 直接そのようなシステムを作成するのではなく, HTML ページから XML データへの写像を与えるラッピング式を与え, その逆写像を自動で計算することである. 我々は, 本方式を用いることにより, 直接システムを構築する場合に比べ, 大幅に少ないコストで Web サイトの再構築が可能であると予想している.

### A Method to Reconstruct Web Sites using a Wrapping Language

YUUTA ISHII,<sup>†1</sup> ATSUYUKI MORISHIMA,<sup>†1,†2</sup>  
SHIGEO SUGIMOTO<sup>†1,†2</sup> and HIROYUKI KITAGAWA<sup>†3</sup>

Today, publishing information from Web sites is common, but not so many organizations have enough resources to maintain the Web contents. Usually, we construct Web sites with a backend DBs in order to reduce the maintenance cost of the Web contents. However, the reconstruction process itself requires a lot of cost. So many Web sites remain to have only static HTML pages. This paper proposes a low cost method to reconstruct static Web sites to make them backend DBs. The idea is to use a wrapping language for giving descriptions of mappings from HTML to XML data and to compute its inverse mappings. We expect that the proposed method can achieve the reconstruction at much lower cost than the ordinary Web site construction from scratch.

#### 1. はじめに

現在, Web サイトを通じた情報発信が普及している. しかし, Web サイトの維持管理に大きなコストをかけることのできる組織はそれほど多くない. Web サイトのコンテンツの維持管理を効率化するためにはバックエンドに DB を持ち, DB 中のデータに基づき動的に HTML ページを生成するシステム (以下では動的 Web サイトと呼ぶ) を構築することが一般的である. しかし, 動的 Web サイトの構築コストは大きいため, 多くの Web サイトは, バックエンドに DB を持たない静的な HTML ページだけの Web サイト (以下では静的 Web サイトと呼ぶ) であることが多い<sup>1)</sup>. 例えば, 大学の Web サイトにおける入試情報や教員情報, 大学の研究室の Web サイトにおける論文リストや構成員情報など, バックエンドに DB を置いて管理した方が望ましい情報であっても, これらは静的な HTML データとして管理されている.

静的 Web サイトが多数存在している主な原因としては, 次の 2 つが考えられる. (1) Web サイトにかかるコストが直接の利益につながるような組織 (例えばショッピングサイトを持つような組織) などを除いて, Web サイトの構築には高い構築コストはかけられない場合が多い. (2) 静的 Web サイトを動的 Web サイトに再構築するには, 多くの人手を要するために非常に高いコストがかかる.

そこで, 本論文では, 静的 Web サイトを, 動的 Web サイトに低コストで再構築するためのシステムを提案する. 提案システムのアイデアは, HTML ページから XML データへの写像を与えるラッピング式 (ラッピング言語を用いてラッピング規則を記述したもの) を与え, その逆写像を計算する手法を用いることである. 本論文では, この提案システムと, そこで利用される逆写像の計算手法について説明する.

提案システム概要. 本システムの入出力を図 1 に示す.

入力は, 次の 2 つである. ① 静的 Web サイト内にある HTML ページの集合  $H = \{h_1, h_2, \dots\}$  (図 1(a)), および② HTML ページから XML データへのマッピングを示す

<sup>†1</sup> 筑波大学大学院 図書館情報メディア研究科

Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba

<sup>†2</sup> 筑波大学大学院 図書館情報メディア研究科/知的コミュニティ基盤研究センター

Grad. Sch. of Library, Information and Media Studies/Research Center for Knowledge Communities, Univ. of Tsukuba.

<sup>†3</sup> 筑波大学大学院 システム情報工学研究科

Grad. Sch. of Sys. and Info. Eng., Univ. of Tsukuba

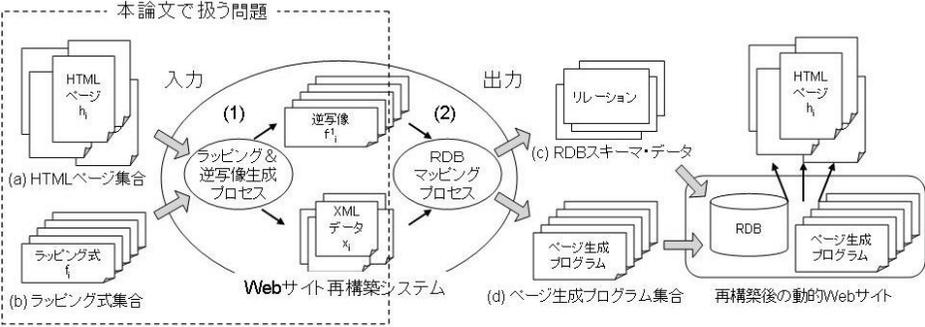


図 1 提案手法の概要

ラッピング式の集合  $F = \{f_1, f_2, \dots\}$  (図 1(b)). ここで, 各  $f_i$  は HTML ページ  $h_i$  を対応する XML データ  $x_i \in X$  にマッピングする式  $f_i: H \rightarrow X$  である.  $f_i$  の定義域  $H$  は, ラッピング式  $f_i$  によって XML データにラッピング可能な HTML ページの集合である.

出力は次の 2 つである. ① RDB スキーマおよび RDB に格納するデータ (図 1(c)). ② ページ生成プログラムの集合 (図 1(d)). ここで, ページ生成プログラムは, DB 中のデータから動的に HTML ページ ( $h_1, h_2, \dots$ ) を生成するためのプログラム (PHP や Perl プログラム) である.

最終的に, 上記に説明した本システムの出力結果を Web サーバと RDB 上に配置することによって, Web サイトの再構築が完了する.

本システムは大きく分けて次の 2 つのプロセスを実行する.

- (1) ラッピング&逆写像生成プロセス. 入力された HTML ページ  $h_i$  および XML へのラッピング式  $f_i$  から, 変換後の XML データ  $x_i$ , およびそのラッピング式の逆写像  $f_i^{-1}$  (XML データから HTML への変換) を求めるプロセス.
- (2) RDB マッピングプロセス. ラッピング&逆写像生成プロセスで生成された XML データ  $x_i$  および逆写像  $f_i^{-1}$  から, XML データの RDB へのマッピング, およびページ生成プログラムを求めるプロセス.

本論文の扱う問題と貢献. 本論文では上記のうち (1) ラッピング&逆写像生成プロセスの問題を扱う. 形式的には, このプロセスは次の様に記述できる. 入力として HTML ページ集合  $H$  およびラッピング式集合  $F$  を受け取り, 次の 2 つを求める. ①  $h_i$  の  $f_i$  による XML データ  $x_i$  へのマッピング結果  $X = \{x_1, x_2, \dots\}$ , および ②  $f_i$  の逆写像の集合

$F^{-1} = \{f_1^{-1}, f_2^{-1}, \dots\}$ . ここで, 逆写像  $f_i^{-1}$  は, XML データ  $x_i \in X$  から, HTML ページ  $h_i \in H$  へのマッピングを示すプログラム  $f_i^{-1}: X \rightarrow H$  (本システムでは XQuery として出力される) である.

本論文の第一の貢献は, 「Web サイトの再構築を行う際には,  $f_i^{-1}$  を直接構築するよりも,  $f_i$  を与え, 逆写像を計算する方が低コストで再構築可能」であることが期待できることを示すことである. 後の章ではこれについて例や実験結果を使って説明する. 本論文の第二の貢献は, 我々の研究グループで開発したラッピング言語である Wraplet<sup>(2)(3)</sup> を逆写像の自動生成が可能となるように制限した言語である iWraplet (invertible Wraplet) を提案し, この iWraplet を用いた場合の, 逆写像を求めるアルゴリズムを示すことである. これにより, この言語の表現力の範囲で, 低コストでの Web サイト再構築が可能であることを示す.

本論文の構成は次の通りである. 2 章で, 関連研究・ソフトウェアについて説明する. 3 章で, Web サイト再構築のためのラッピング言語 iWraplet について説明する. 4 章で, iWraplet 式の逆写像の計算について説明する. 5 章では, iWraplet を用いた Web サイト再構築の容易性に関する考察を示す. 6 章はまとめと今後の課題である.

2. 関連研究・ソフトウェア

Web コンテンツのラッピング技術については, これまで多くの研究が行われてきた<sup>(4)(5)</sup>. 具体的には, HTML データを入力として, それらを XML データや他の半構造データ等にマッピングするための言語やツールの研究が行われてきた. これら既存の研究に対する本研究の新規性は, Web サイト再構築の問題をラッピング式の逆写像を求める問題としてモデル化し, それにより低コストの Web サイト再構築が可能であることを示すことである. 本論文で, HTML データのラッピング言語として XWRAP<sup>(4)</sup> 等の他の言語を用いずに, iWraplet を提案したのは, iWraplet が他の言語と比べて単純であり, この問題にアプローチする第一歩として適切だと考えたためである.

Web サイトの作成・管理を支援するための仕組みとしては, Zope<sup>(6)</sup> や XOOPS<sup>(7)</sup> などに代表される CMS がある. その目的はあくまで Web コンテンツの容易な作成の支援であり, Web コンテンツを一から作成することを支援するものである. それに対し, 本研究は, 既存の静的 Web サイトを, バックエンドに DB を持つ Web サイトに再構築することを支援するものである.

ソフトウェア工学の分野では, 既存のソフトウェアの再設計や再構築のための仕組みとして, リバースエンジニアリングやリエンジニアリング<sup>(8)(9)</sup> などの技術についての研究が盛ん

に行われてきた。本研究は、これらの考え方を、Web サイトのドメインに適用しようとするものである。ドメインが異なるため、本手法はこれらの既存研究とは異なるものになる。

Vu-X<sup>10)</sup> は双方向のデータ変換用言語 Bi-X<sup>11)</sup> を利用した Web サイト構築・更新手法である。この手法を用いてバックエンドを XML-DB とする Web サイトを構築すると、Web サイトの HTML を直接更新する事により、自動的にバックエンドの XML-DB の内容も更新される。Vu-X は Web サイトの再構成を行うものではないが、要素技術のいくつかは本研究と関連が深く、我々の研究に活用できる可能性がある。

### 3. Web サイト再構築のためのラッピング言語 iWraplet

本章では、Web サイト再構築のためのラッピング言語 iWraplet について説明する。3.1 節では、iWraplet の概要について説明する。3.2 節では iWraplet と Wraplet の違いについて説明する。3.3 節では iWraplet を用いた Web サイト再構築の例を示す。

#### 3.1 iWraplet の概要

iWraplet は、HTML ページから XML データを抽出するために我々の研究グループで開発したラッピング言語である Wraplet を、逆写像が必ず存在 (invertible) するように制限した言語である。制限については 3.2 節で述べる。逆写像は、XQuery を用いて表現する。説明のために、以降では XML データを木の用語を用いて表現する。XML データの要素のことをノードと呼び、要素の入れ子関係を親子関係で表現し、親ノード/子ノードと呼ぶ。また、ノードの名前をラベルと呼び、ノードが直接含むテキストデータをそのノードの値と呼ぶ。

まず、簡単な iWraplet 式の例を説明する。図 2(a) のような果物在庫を表現した HTML ページがあり、ラッピングした結果の XML データとして図 2(b) のような XML データを

<pre>&lt;html&gt; ... &lt;ul&gt; &lt;li&gt;りんご@ 10&lt;/li&gt; &lt;li&gt;みかん@ 20&lt;/li&gt; &lt;li&gt;桃@ 30&lt;/li&gt; &lt;/ul&gt; ... &lt;/html&gt;</pre>	<pre>&lt;在庫&gt; &lt;果物&gt;&lt;名前&gt;りんご&lt;/名前&gt;&lt;数量&gt;10&lt;/数量&gt;&lt;/果物&gt; &lt;果物&gt;&lt;名前&gt;みかん&lt;/名前&gt;&lt;数量&gt;20&lt;/数量&gt;&lt;/果物&gt; &lt;果物&gt;&lt;名前&gt;桃&lt;/名前&gt;&lt;数量&gt;30&lt;/数量&gt;&lt;/果物&gt; &lt;/在庫&gt;</pre>
---	--

(a) 果物在庫を表現した HTML ページ                      (b) iWraplet を使って抽出したい XML データ

図 2 ラッピング前の HTML ページの例 (a) およびラッピング後の XML データの例 (b)

抽出したいとする。この時、このラッピングを行うための iWraplet 式を図 3(a) に示す。また、図 3(a) の iWraplet 式の逆写像は図 3(b) のようになる。逆写像を求める方法については 4 章で述べる

iWraplet の構文。次に、iWraplet の構文の概要について説明する。詳細に関しては付録に示す。

iWraplet 式は、“ラベル:パターン/子ノードのための (一般には複数の)iWraplet 式 ” という入れ子構造で表現される。

ラベルは出力する XML データのノードのラベルを指定する。今回の例の場合では、<在庫>、<果物>、<名前>、<数量>がラベルである。

パターンはそのノードに対応する HTML ページの部分指定する。これは、文字の正規表現に、あらかじめライブラリとして用意されているパターン部品、特殊指示子などを組み合わせて指定する。パターン部品および特殊指示子については後述する。今回の例の場合では、#(1)ul と #li と \_val(#not(@)) @ と \_val(#num) がパターンである。パターンはマッチング対象の文字列の先頭の文字からマッチングされる。先頭の文字列を含まない任意の部分とマッチするパターンを書くためには、パターン部品や特殊指示子を用いたパターンを記述する必要がある。iWraplet 式のパターンが省略されていた場合は、.\* というパターンと解釈する。

<pre>&lt;在庫&gt;: #(1)ul/{   &lt;果物&gt;: #li/[     &lt;名前&gt;: _val(#not(@)) @,     &lt;数量&gt;: _val(#num)   ] }</pre> <p>(a) iWraplet 式</p>	<pre>&lt;html&gt; ... {   let \$data1 := doc('在庫.xml')/在庫   return   &lt;ul&gt;   {     for \$data11 in \$data1/果物     return     &lt;li&gt;       { \$data11/名前/text() } @       { \$data11/数量/text() }     &lt;/li&gt;   }   &lt;/ul&gt; } ... &lt;/html&gt;</pre> <p>(b) iWraplet 式の逆写像</p>
---	--

図 3 iWraplet 式の例 (a) およびその逆関数の例 (b)

パターン部品名	機能
#tag	<tag> ... </tag>内の文字列... にマッチ 例えば#li や#h1 など
#name	氏名にマッチ (名前辞書を利用)
#not (パターン)	引数に指定されたパターンにマッチするまでの文字列にマッチ 例えば#not(z) は文字列: abcdefg の abcd にマッチ
#num	数字にマッチ

図 4 パターン部品の一部

iWraplet 式と HTML ページとのパターンマッチングの手順は次の通りである。iWraplet 式とマッチング対象の HTML ページが与えられると、iWraplet 式の入れ子構造の外側から順に HTML ページのテキストにパターンマッチを行い、パターンがマッチする毎に XML 木のノードを生成する。このノードの子ノードのための iWraplet 式の評価は、親ノードのための iWraplet 式のパターンとマッチした部分をマッチングの対象として行う。

パターン部品は、よく使うパターンや複雑なパターンに名前を付けたものであり、“#(<出現位置>)<パターン部品名>”と表す。パターン部品の一部を図 3.1 に示す。出現位置は正の整数で指定する。出現位置を指定した場合は、マッチング結果は必ずしも先頭の文字を含まない。例えば、#(1)ul はマッチング対象の先頭から見て 1 番目に現れる ul 要素にマッチする。つまり #(1)ul は ab<ul> </ul><ul>...</ul>の 部分にマッチする。出現位置は省略可能であり、省略した場合は通常通り、マッチング対象の先頭の文字を含んだ文字列とのマッチングが行われる。つまり #ul は <ul> </ul>... の とはマッチするが ab<ul> </ul>... とはマッチしない。

特殊指示子は、マッチしたパターンに対する何らかの処理を指示する。上の例では\_val( ) が特殊指示子である。\_val(パターン) はパラメータで指定されたパターンにマッチした文字列を対応する XML ノードの値とする特殊指示子である。例えば、文字列: abcdefg に対して iWraplet 式: <test>:\_val(#not(z)) を使ってラッピングすると <test>abcd</test> という XML データが得られる。

他にも様々なパターン部品や特殊指示子があるが、それらについての説明は論文<sup>2)</sup> で述

```

<在庫>: #(1)ul/[
  <果物>:_val(#not(@)).*?<li>_val(#not(@))
]

```

(a) Wraplet 式

```

<在庫>
  <果物>りんごみかん</果物>
</在庫>

```

(b) 出力となる XML データ

図 5 (1) の条件を満たさない XML データの抽出の例

```

<在庫>: #(1)ul/{
  <果物>:_val(#li)/[
    <名前>:_val(#not(@)) @,
    <数量>:_val(#num)
  ]
}

```

(a) Wraplet 式

```

<在庫>
  <果物>りんご@ 10<名前>りんご</名前><数量>10</数量></果物>
  <果物>みかん@ 20<名前>みかん</名前><数量>20</数量></果物>
  <果物>桃@ 30<名前>桃</名前><数量>30</数量></果物>
</在庫>

```

(b) 出力となる XML データ

図 6 (2) の条件を満たさない XML データの抽出の例

べられているため、ここでは省略する。

子ノードのための iWraplet 式は必ず{...}または[...]でくくる必要がある。{...}は繰り返し構造を表す。図 3(a) の在庫の子ノードのための iWraplet 式は {...} でくくられているため図 2(b) では、在庫の子ノードの果物ノードが繰り返し構造になっている。[...]は列構造を表す。図 3(a) の果物の子ノードのための iWraplet 式は [...] でくくられているため図 2(b) では、[...] でくくられた果物の子ノードが同階層になっている。

3.2 iWraplet と Wraplet の違い

iWraplet のための Wraplet の制限について説明する。iWraplet のための Wraplet の制限は大きく分けて次の 2 つがある。

- (1) XML データの要素の値は元の HTML ページのいずれかの要素の連続した部分文字列でなくてはならない
- (2) 元の HTML ページに現れる同一の文字列が XML データに複数現れてはならない それぞれについて次で説明する。

まず制限 (1) について説明する。(1) の条件を満たさない XML データの抽出の例を図 5 に示す。図 5(a) の Wraplet 式を用いて図 2(a) の HTML ページから XML データの抽出を行うと、図 5(b) の XML データが得られる。図 5(b) の XML データでは、元の HTML データでは別々の<li>タグで区切られていた果物の情報が結合されて<果物>の値となっており、いずれかの要素の部分文字列になっていない。このような XML データは、逆写像を利用した復元が一般に不可能になるため、iWraplet では、元の HTML ページのタグに含まれる内容の結合となる XML データの抽出を許していない。

次に制限 (2) について説明する。元の HTML ページの文字列が複数回現れる XML データの抽出の例を図 6 に示す。図 6(a) の Wraplet 式を用いて図 2(a) の HTML ページから XML データの抽出を行うと、図 6(b) の XML データが得られる。図 6(a) の Wraplet 式では<果物>のパターンで\_val(#li)を利用して値を出力した後に、<果物>の子ノードのため

の Wraplet 式でも\_val() を利用しているため、同じ文字列を持った XML データが抽出される。このように、元の HTML データの内容が複数回現れる XML データが抽出された場合、その XML データにおける複数の文字列の一つだけを更新した場合、逆写像での扱いを一意に決定できない。そのため iWraplet では、元の HTML ページの内容が複数回現れるような XML データへの写像を許さない。

3.3 iWraplet を用いた Web サイト再構築の例

本節では、iWraplet を用いた Web サイト再構築の例について説明する。例として、ある大学の研究室に所属する学生の Web サイトの場合を取り上げ、図 1 で示した「本論文で扱う問題」の部分について説明する。

まず、この学生の Web サイトに図 7(a) に示すような HTML ページがあったとする。そして、この HTML ページの中から「論文」に関するデータを DB で管理したいとする。この時、この Web サイトの再構築を行うためには、図 7(b) に示す iWraplet 式を記述する。図 7(b) の iWraplet 式は図 7(a) の HTML ページから論文に関するデータを XML データとして抽出するための式である。HTML ページに対する iWraplet 式の記述が完了したら、HTML ページおよび iWraplet 式を入力としてシステムに与える(図 1 の入力部分)。

システムは入力された HTML ページおよび iWraplet 式に対して、ラッピング&逆写像生成プロセスで、XML データの抽出および逆写像の生成を行う(図 1 のラッピング&逆写像生成プロセス)。今回の例の場合は、図 8(a) の XML データおよび図 8(b) の逆写像が生

```
<html>
...
<h3>プロフィール</h3>
<ul>
  <li>名前: 筑波太郎</li>
  <li>出身地: 茨城県つくば市</li>
</ul>
...
<h3>論文一覧</h3>
<ul>
  <li>筑波太郎「x x x の提案」2009 年 3 月.</li>
  <li>筑波太郎「      の提案」2008 年 12 月.</li>
  <li>筑波太郎「      の提案」2007 年 9 月.</li>
</ul>
...
</html>
```

(a) 再構築の対象となる HTML ページ

```
<論文一覧>:#ul(2)/{
  <論文>:#li/{
    <著者>:_val(#not(' '))',
    <タイトル>:_val(#not(' ')),
    <日付>:_val(#not(\.))
  }
}
```

(b) 再構築のための iWraplet 式

図 7 再構築の対象となる HTML ページ (a) および再構築のための iWraplet 式 (b)

```
<論文一覧>
<論文>
  <著者>筑波太郎</著者>
  <タイトル>x x x の提案</タイトル>
  <日付>2009 年 3 月</日付>
</論文>
<論文>
  <著者>筑波太郎</著者>
  <タイトル>      の提案</タイトル>
  <日付>2008 年 12 月</日付>
</論文>
<論文>
  <著者>筑波太郎</著者>
  <タイトル>      の提案</タイトル>
  <日付>2007 年 9 月</日付>
</論文>
</論文一覧>
```

(a) 抽出された XML データ

```
<html>
...
<h3>論文一覧</h3>
{
  let $data1 := doc('論文一覧.xml')/論文一覧
  return
  <ul>
    {
      for $data11 in $data1/論文
      return
      <li>
        { $data11/著者/text() } 「
        { $data11/タイトル/text() }
        { $data11/日付/text() }.
      </li>
    }
  </ul>
  ...
}</html>
```

(b) 生成された逆写像

図 8 ラッピング式処理プロセスの処理結果

成される。

これらの出力は、続く RDB マッピングプロセスの入力として利用される。ここでは、RDB に格納するための処理、および、HTML への逆写像から、RDB から HTML データを生成するプログラムへの逆写像の変換が行われる。(図 1 の RDB マッピングプロセス)。

以上のように、iWraplet を用いた Web サイト再構築において人手を要するのは、iWraplet 式を記述する部分のみである。3 章で説明したように、iWraplet は記述が容易であり、提案手法を用いれば、静的 Web サイトから動的 Web サイトへの低コストでの再構築が可能であると思われる。

4. iWraplet 式の逆写像の計算

本章では、iWraplet 式 f の逆写像 f<sup>-1</sup> を求める手法について説明する。図 9 に f<sup>-1</sup> による HTML ページ復元のイメージを示す。f<sup>-1</sup> はテンプレート部および問合せ部から構成されている。テンプレート部は元の HTML から XML データの値として抽出した内容を除いた、HTML ページ内の全ての要素である。問合せ部は f で得られた XML データを読み込んで、元の HTML ページのタグを付与する処理を行う。

逆写像を求める基本的なアイデアを図 10 に示す。逆写像の生成は、iWraplet 式の入れ子

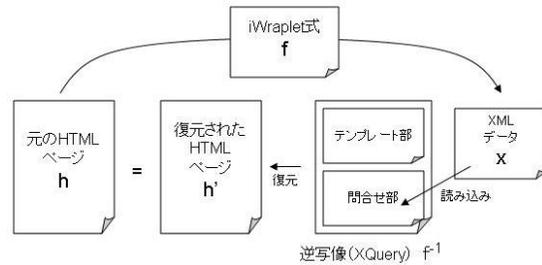


図 9 逆写像による HTML ページ復元のイメージ

構造の外側から順に HTML ページとのパターンマッチを行い、マッチしたパターンに `_val()` が含まれていた場合、`_val()` の引数に与えられたパターンの部分とマッチした文字列を問合せ部と置換する事を再帰的に繰り返すことによって行われる。問合せ部の内容は、iWraplet 式の繰り返し構造の有無や、パターンに含まれる特殊指示子やパターン部品などによって、変化する。

ここでは、入れ子構造を持たない (`label : pattern` の形の) iWraplet 式の逆写像の計算について説明する。図 11 に入れ子構造を持たない単一の iWraplet 式に対する逆写像の計算アルゴリズムを示す。入れ子構造を持つ iWraplet 式に対する逆写像の計算アルゴリズムは付録に示す。

図 11 で示した、逆写像を求める `Rev` 関数について説明する。まず、`Rev` 関数の入力および出力を説明する。`Rev` 関数は入力として次の 3 つを受け取る。(1) 入れ子構造を持たない単

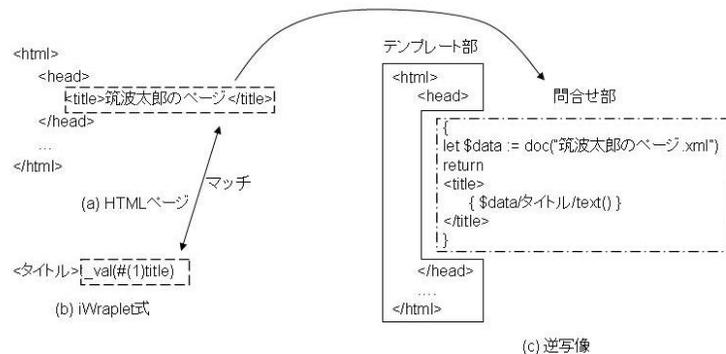


図 10 逆写像の生成

```

1: Rev(label:pattern, text, name) =
2:   if(matchString(pattern,text)!=null){ //pattern が text にマッチした場合
3:     if(pattern.contains("_val")){ //pattern が _val() を含んでいたら
4:       repelem = "{ "
5:       repelem += "let $data := doc(\" + name + ".xml")"
6:       repelem += "return"
7:       repelem += "{ $data/" + label + "/text()}"
8:       repelem += "}"
9:       val = pattern.valPattern // _val() 内のパターンを返す
10:      text = text.replaceFirst(val, repelem) // _val() で出力対象となる部分を置換
11:      output(text)
12:     }else{
13:       output(text)
14:     }
15:   }else{
16:     error; //パターンにマッチしなかったならばエラーを表示して終了
17:   }

```

図 11 入れ子構造を持たない iWraplet 式に対する逆写像を求めるアルゴリズム

一の iWraplet 式 (`label:pattern`) (2) HTML ページ (`text`) (3) HTML ページから iWraplet 式を用いて抽出した XML データのファイル名 (`name`)。 `Rev` 関数の出力は XQuery 形式の逆関数である。

図 11 のアルゴリズムが、入れ子構造を持たない (`label : pattern` の形の) iWraplet 式の逆写像を生成するプロセスを、図 10 を用いて説明する。図 10(a) の HTML ページおよび図 10(b) の iWraplet 式に対する逆写像を計算するとき、`Rev` 関数への入力は `label=<タイトル>`、`pattern=_val(#{1}title)`、`text="<html><head><title>筑波太郎のページ..."`、`name=出力する XML データのファイル名 (ここでは「筑波太郎のページ.xml」)` となる。

`Rev` 関数の処理は次のようになる。

- (1) iWraplet 式の `pattern` が HTML データ `text` にマッチするか調べる (2 行目)  
マッチしなかった場合はエラーを表示して終了する (15~17 行目)
- (2) iWraplet 式の `pattern` が `_val()` を含んでいるか調べる (3 行目)  
`_val()` を含んでいなかった場合は HTML データ `text` をそのまま出力する (12~14 行目)
- (3) ファイル名 `name` のファイルから XML データを読み込み元の HTML データに復元する処理を FLOWR 式で記述する (4~8 行目)
- (4) `_val()` の引数のパターンにマッチする部分を FLOWR 式に置換する (9~10 行目)
- (5) 逆写像を出力する (11 行目)

## 5. iWraplet を用いた Web サイト再構築の容易性に関する考察

本章では、提案手法を用いて Web サイトの再構築を行う際の容易性に関する考察を述べる。予備実験として、我々の研究グループに所属する 3 人の学生の HP を対象として、「どの程度のサイズの iWraplet 式が必要か」に関する調査を行った。例えばタイトルと著者名のみが記載されている場合は、iWraplet 式に現れうる項目 (式中の label) の最大数は 2 となり、タイトル、著者名、雑誌名、ページ数が記載されている場合は、項目数は 4 となる。このような調査を行った理由は、3.3 節の Web サイト再構築の例の図 7(この例では項目数は 3) を見れば分かるように、iWraplet 式の記述コストは、抽出する XML データに対してどの程度詳細なタグ付けを行うかに依存するためである。

調査を行った結果、項目数の最も少ない論文は 2 項目で、項目数の最も多い論文は 9 項目となった。実験結果から、少なくとも今回調査した範囲においては、最悪でも図 7 の iWraplet 式の 3 倍程度の iWraplet 式を記述することで可能であった。

このことから、Web サイトの再構築を行う際に、提案手法を用いた場合とそうでない場合を比較すると、前者の場合は数行程度の iWraplet 式を記述するのみであるのに対して、後者の場合は、XML データベースの作成や XQuery の記述などを一から行うことが必要となる。したがって提案手法はその再構築コストを大幅に削減できる可能性があると推測できる。

## 6. まとめと今後の課題

本論文では静的な HTML データで構成された Web サイトを、バックエンドにデータベースを持つ動的な Web サイトへと低コストで再構築するための手法を提案した。本手法では、直接動的な Web サイトを構築するのではなく、HTML データから XML データへのマッピングを作成し、その逆写像を求めることによって Web サイトの再構成を行うものである。本論文では、本手法の全体像についての説明、および提案手法の重要な要素技術である、ラッピング言語 iWraplet とラッピング式の逆写像の計算に関する説明を行った。

今後の課題としては、RDB マッピングプロセスの開発と提案手法に基づいたシステムの実装が挙げられる。また、手法の評価に関しては、iWraplet 式の表現力に関する評価や、様々なドメインの Web サイトを対象とした実験などを行う必要があると考えられる。

## 謝 辞

ゼミなどでコメントいただきました、筑波大学大学院図書館情報メディア研究科 阪口哲男准教授、永森光晴講師に感謝いたします。本研究の一部は科学研究費補助金特定領域研究 (#21013004)、科学研究費補助金基盤研究 (B)(#19300081)、科学研究費補助金若手研究 (#20700076) による。

## 参 考 文 献

- 1) 澤菜津美, 森嶋厚行, 飯田敏成, 杉本重雄「北川博之・コンテンツ一貫性制約を用いた Web サイト管理手法の提案」DEWS2007, 7 pages, 2007 年 3 月.
- 2) 澤菜津美, 森嶋厚行, 杉本重雄, 北川博之「バックエンド DB を持たない Web コンテンツ管理のためのラッピング言語」日本データベース学会 Letters, Vol. 6, No. 2, pp. 69-72, 2007 年 9 月, 日本データベース学会.
- 3) 澤菜津美, 森嶋厚行, 杉本重雄, 北川博之「情報統合利用を目的とした HTML ページのラッピング支援」電子情報通信学会第 19 回データ工学ワークショップ (DEWS2008), 7 pages, 宮崎, 2008 年 3 月.
- 4) L. Liu, C. Pu, and W. Han, "XWRAP: An XML-enabled wrapper construction system for web information sources." International Conference on Data Engineering (ICDE), pp. 611-621, 2000.
- 5) Arnaud Sahuguet, Fabien Azavant, "Building intelligent Web applications using lightweight wrappers." Data Knowl. Eng. 36(3): 283-316 (2001).
- 6) Zope, "http://www.zope.org/", (参照 2009-07-30)
- 7) XOOOPS, "http://xoopscube.org/", (参照 2009-07-30)
- 8) E. J. Chikofsky and J. H. Cross, II, "Reverse Engineering and Design Recovery: A Taxonomy." IEEE Software, vol. 7, no. 1, pp. 13-17, January 1990.
- 9) Kathi Hogshead Davis, Peter H. Aiken, "Data Reverse Engineering: A Historical Survey." wcre, pp.70, Seventh Working Conference on Reverse Engineering (WCRE 2000), 2000.
- 10) Keisuke Nakano, Zhenjiang Hu, Masato Takeichi, "Consistent Web Site Updating based on Bidirectional Transformation." 10th IEEE International Symposium on Web Site Evolution (WSE 2008), Beijing, China, October 3-4, 2008.
- 11) Dongxi Liu, Zhenjiang Hu and Masato Takeichi, "Bidirectional Interpretation of XQuery." ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation (PEPM 2007), Nice, France, January 15-16, 2007. pp.21-30.

付 録

A.1 iWraplet の構文

iWraplet の構文を示す .

iWraplet の基本構文の定義

```
<expression> ::= [<label>]: [<pattern>][ / '<children> ]
<children> ::= [ '(' <condition> ')' ] { '<expression>' |
    | '<expression>' } '<expression>' }
```

pattern に関する定義

```
<pattern> ::= <regexp> | '_stay('<regexp>')' | '_skip('<regexp>')'
<regexp> ::= <regexp>+
<regexp> ::= '#'[ '(' <digit>+ ') ] ( 'name' | 'num' | 'date' | 'not('<regexp>')' | <tag> )
<regexp> ::= '_match('<regexp>')'
<regexp> ::= '_val('<regexp>')'
<regexp> ::= <char> [ <quantity> ]
<regexp> ::= <regexp> '|' <regexp>
<regexp> ::= '(' <regexp> ')' [ <quantity> ]
<tag> ::= HTML タグ
<char> ::= <basechar> | '[' <basechar>+ ']' | '[' ^ <basechar>+ ']'
<quantity> ::= '+' | '?' | '*' | '?' | '?'
<basechar> ::= 任意の 1 文字 (メタ文字除く) | \メタ文字 | '.'
```

condition に関する定義

```
<condition> ::= '_skip('<regexp>')' | '_stay('<regexp>')'
    | '_skip('<regexp>')' | '_stay('<regexp>')'
    | <regexp>
```

- <pattern> に \_stay(<patexp>) を含むのは許さない
- <pattern> において \_val() と \_match() の入れ子は許さない
- <pattern> において \_val() は 0 回または 1 回現れる
- <pattern> において \_match() は 0 回または 1 回現れる
- 親ノードのための Wraplet 式の <pattern> に \_match() を含まずに \_val() を利用するのは許さない

A.2 入れ子構造を持った iWraplet 式に対する逆写像の計算アルゴリズム

入れ子構造を持った iWraplet 式に対する逆写像の計算アルゴリズムを示す . A.2.1 ~ A.2.3

を再帰的に適用することにより, 入れ子構造を持った iWraplet 式の逆写像が求められる .

A.2.1 列構造の子要素を持った iWraplet 式に対する逆写像の計算

```
Rev(label:pat/[e1, e2, e3, ...], text, name, depth) =
    if (matchString(pat,text)!=null) { //pat が text にマッチした場合
        repelem = ""
        repelem += "let $data" + depth + " := "
        if (depth.equals("1")) {
            "doc(" + name + ") /" + label
        } else {
            "$data" + depth.substring(0, length-1) + "/" + label
        }
        repelem += "return"
        pattern = Pattern.compile(pat) //正規表現をコンパイル
```

```
matcher = pattern.matcher(text) //パターンマッチ
n = exps.count //並列に並んだ UWraplet 式の数
child = matcher.group(1)
for (i=1; i<=n; i++) {
    repelem += Rev(ei, matchString(ei.pat, child), name, depth+Integer.toString(i))
    child = remove(condition, child)
}
repelem += "}"
text = text.replaceFirst(pat, repelem)
return text
} else {
    error; //パターンにマッチしなかったならばエラーを表示して終了
}
```

A.2.2 繰り返し構造の子要素を持った iWraplet 式に対する逆写像の計算

```
Rev(label:pat/(condition){e}, text, name, depth) =
    if (matchString(pat,text)!=null) { //pat が text にマッチした場合
        repelem = ""
        repelem += "let $data" + depth + " := "
        if (depth.equals("1")) {
            "doc(" + name + ") /" + label
        } else {
            "$data" + depth.substring(0, length-1) + "/" + label
        }
        repelem += "return"
        pattern = Pattern.compile(pat) //正規表現をコンパイル
        matcher = pattern.matcher(text) //パターンマッチ
        child = matcher.group(1)
        repelem += "for data" + depth + 1 + " indata" + depth + '/' + e.label
        repelem += "return"
        repelem += Rev(e, child, name, depth + 1)
        repelem += "}"
        text = text.replaceFirst(pat, repelem)
        return text
    } else {
        error; //パターンにマッチしなかったならばエラーを表示して終了
    }
```

A.2.3 単一の子要素のための iWraplet 式に対する逆写像の計算

```
Rev(label:pat, text, name, depth) =
    if (matchString(pat,text)!=null) { //pat が text にマッチした場合
        if (pat.contains("_val")) { //pat が _val() を含んでいたら
            repelem = "{ $data" + depth.substring(0, length-1) + "/" + label + "/text() }"
            val = pat.valPattern // _val() 内のパターンを返す
            text = text.replaceFirst(val, repelem) // _val() で出力対象となる部分を置換
            return text
        } else {
            return text
        }
    } else {
        error; //パターンにマッチしなかったならばエラーを表示して終了
    }
```