ソフトウェア工学　136－17
（2002．3．8）

LispへのXML文書構造変換言語の埋め込みとそれのシャッフル表現への拡張

紙名 哲生　　玉井 哲雄

東京大学

本研究ではXML文書の構造変換を行う方式として，パターンマッチを使ったXML文書構造変換用の構文を，マクロを使ってLispの拡張として実装する方法を提案する．これにより，LispからXML文書構造変換言語をシームレスに呼び出すことが可能になる．またこのパターンマッチの構文をシャッフル表現が記述できるように拡張した．これにより，RELAX NGのようにインターリーブ型のあるスキーマを持つXML文書に対してもパターンマッチが可能になる．変換先のXML文書の妥当性は動的に検証する方法を採用し，スキーマを変更してもシステムを再コンパイルする必要がない．

Embedding XML transformation language in Lisp and its extension to shuffle expression

Tetsuo Kamina　　Tetsuo Tamai

University of Tokyo

In this paper, we propose a new approach to achieve XML transformations: embedding XML transformation language in Lisp. By this approach, we can seamlessly use the functions of XML transformations from Lisp. The novel feature of this language is shuffle expression pattern matching. It supports shuffle operator, as RELAX NG schema language does, and shuffle closure operator. The validity of transformed XML documents can be dynamically checked. This makes it possible to change schemas of XML documents without recompiling the systems.

1　はじめに

XML [11] はDTD，XML Schema [10] 等のスキーマ言語によって文書に型をつけることができる．さらにそのスキーマとXML文書とを照らし合わせて，XML文書がそのスキーマに合っているかどうかを自動検証することができる（妥当性の検証）．そこでXMLはデータ交換やデータ処理などにおける安全性を高めることができると期待されており，現在ではインターネットなどにおけるデータ交換の標準的なデータ形式として発展してきている．

データ交換等でよく用いられるXMLにとって，構造変換は重要な技術である．それはXML文書を異なるアプリケーション間で用いる場合，そのXML文書をそれぞれのアプリケーションで用いられるデータ形式に変換する必要があるからである．例えば，XML文書をWebブラウザで表示する場合は，それをHTML文書に変換する必要がある．

XML文書の構造変換を行うのには，2種類のアプローチが考えられる．一つはXML文書を通常の汎用言語のデータ型にマップして，その言語を用いて構造変換プログラムを書く方法である．この場合，その言語の機能をそのまま使うことができるので，柔軟で複雑な処理ができるといった利点があるが，XML文書構造変換で頻出する処理 (例えばタグ名の付け換えなど) を簡単に書くためには，XML文書構造変換用の言語を使った方が便利である．そこでもう一つのアプローチとして，XSLTなどのXML文書処理用の言語が考えられている．しかしXML用の言語では複雑な計算を行うのには工夫が必要であ

る．現状では，これら別々の言語を組み合わせてシステムを構築するのが一般的であり，こうした別々の言語を意識せずに，同一の言語でこれらを扱えるような仕組みを作ることが望ましい．

本研究では，両者のアプローチを統合し，汎用言語からシームレスに呼び出すことができる XML 文書構造変換言語を提案する．これは Lisp のマクロ機能を使って，Lisp の構文として XML 文書構造変換言語を拡張することにより実現できる．これにより Lisp 以外の言語を特に意識しなくとも XML の処理系が使えるようになる．本言語設計上の留意点は次の 2 点である．

- 変換先の XML 文書の妥当性を検証できる必要がある．特にサービスとして動かすことの多い XML アプリケーションでは，XML のスキーマを変更してもシステムを再コンパイルすることなく変更されたスキーマを反映させることが望ましい．
- 2001 年 11 月に，国際的な標準化団体 OASIS によって XML スキーマ言語 RELAX NG が制定された [18]．これはシャッフル (インターリーブ) という型を扱うことのできる XML スキーマ言語であるが，プログラミング言語レベルでこれを扱うものは存在しない．そこでこれを扱うことのできるプログラミング言語が必要である．

この論文ではまず，関連研究について述べた後，本ツールの全体構成について述べる．次に Lisp 上での XML 文書の内部表現について述べる．次に Lisp に埋め込まれた XML 文書構造変換言語を紹介し，それのシャッフル表現への拡張について述べる．

2 関連研究

XML の処理系の中で，広く一般的に使われている API として DOM（Document Object Model） [12] があげられる．これは XML 文書を構文解析してオブジェクトの木に変換し，各オブジェクトを操作するインターフェイスが各種用意されてあるものである．DOM 構文解析器には XML 文書の妥当性検証を行うものも存在する．しかし，DOM 木から別の DOM 木への変換を行う等の処理をした後，その木が妥当であるかどうかを検証するシステマティックな方法は用意されていない．

Franz Inc. から発表された XML 構文解析器 [2] は，XML 文書をオブジェクトの木ではなくて，本研究同様 Lisp の S 式に変換するものである．これも妥当性の検証を行う仕組みは提供されていない．

他に XML 文書処理をプログラミング言語に組み込む手法として，Wallace と Runciman によって提案された，XML 文書処理に Haskell を使う方法がある [15]．ここでは，DTD から Haskell 上のデータ型へのマッピングを定義することによって，XML 文書は Haskell の型を持ったインスタンスとして表現される．型システムを用いて，XML 文書の妥当性が静的に保証される．しかし本研究のような動的な妥当性の検証方法についての仕組みは提供されていない．

これらはいずれも汎用言語から XML 文書を扱うものである．よってその言語の機能をそのまま使うことができるので，柔軟で複雑な処理が行える．しかし XML 文書をその言語のもつデータ構造に合わせる必要があり，例えば DOM のようなオブジェクトでは，実装の詳細は隠蔽しつつ多様な要求に応えるためどうしてもインターフェイスが複雑になる点，Haskell の場合は XML 文書で扱える「型」を Haskell 上での型システムで表現できるものに制限しなければならない点，Franz Inc. の場合，`car` や `cdr` などのプリミティブではそのプログラムが何をやっているかが理解しづらい点など，不便な点も多い．

XML 文書処理用のドメインスペシフィック言語としては，XSLT [13]，XQuery[14]，YTS [16]，XDuce [6]，XMLambda [9] 等があげられる．XSLT，YTS はいずれもスタイルシートと呼ばれる XML 文書構造変換用の言語である．XSLT は高い表現能力を持ち，YTS は簡単な処理が簡潔に書ける．いずれも変換先の XML 文書の妥当性は検証しない．XQuery は XML データベースへの問い合わせを行う言語である．

XDuce [6] は静的に型付けされた，XML 文書処理用の関数型言語である．正規表現型と正規表現パターンマッチを持ち合わせているという大きな特徴がある．主に XML 文書の構造変換などに用いられ，型システムを用いることにより，変換先の XML 文書の妥当性が保証されているという利点がある．同様に，XMLambda [9] も静的な XML 文書処理用の関数型言語である．両者とも，本研究のような動的な妥当性検証の仕組みは用意されていない．

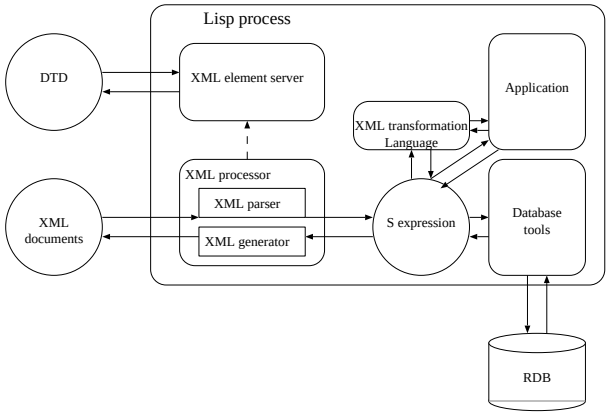

図 1: 全体構成図

XML 用言語を開発することにより，汎用言語に合わせることなく XML 文書処理に適した言語設計を行うことができるが，複雑な計算を行うなどの柔軟性には欠ける[1]．これらに対し本研究では，XML 文書処理用の構文を Lisp に新たに埋め込むというアプローチをとっており，Lisp から XML 文書構造変換プログラムを呼び出したり，XML 文書構造変換言語から Lisp の機能を使って複雑な処理を行うことがシームレスに実現される．

これらの関係を表 1 にまとめる．

3 全体構成

本ツールは，XML プロセサ (XML Processor)，XML 要素サーバ (XML element server)，XML 文書構造変換言語 (XML transformation language) 及び関係データベース関連ツール (Database tools) から構成される．XML プロセサは XML 構文解析系 (XML parser) 及び XML 文書自動生成系 (XML generator) より構成される (図 1)．XML 要素サーバは DTD の内容を保持した辞書である．XML 構文解析系は XML 文書を構文解析して S 式に変換し，XML 文書自動生成系は S 式から XML 文書を動的に自動生成する．関係データベース関連ツールは，データベースのスキーマから文書型へのマッピングや，問い合わせの結果を XML 文書に変換することを行う．Web サーバ等のアプリケーションと本ツールは，同じ Lisp プロセス上で実行される．

これらのうち，XML 要素サーバを使った動的な妥当性の検証方式とデータベース関連ツールについては文献 [17] で報告してある．また各ツールのインターフェイスは文献 [8] で報告している．

[1]XSLT では XSP の機能を使えば XSLT プログラムに Java プログラムを埋め込むことはできる．しかし変換先の XML 文書が妥当であるかを検証する仕組みは用意されていない

```
<html xmlns=
      "http://www.w3.org/1999/xhtml">
 <head>
  <title>Hello</title>
 </head>
 <body>
  <h1>Hello, World!</h1>
  I'm in Tokyo.
 </body>
</html>
```

図 2: XML 文書の例

4 Lisp 上での XML 文書の表現

この手法で，XML 文書を Lisp 上でどのように表現するかを述べる．ここではまず例を挙げる．この手法では，図 2 に書かれている XML 文書は，S 式を用いて図 3 のように表現される．ここでは，<head> タグは:head のように表現されており，

```
<html xmlns=
      "http://www.w3.org/1999/xhtml">
```

のように属性のあるタグは，タグ名の後に属性名と属性値を続けて，以下のように表現される．

```
(:html :xmlns
       "http://www.w3.org/1999/xhtml")
```

このように，S 式で XML 文書を表現すれば，Lisp 言語の単純なリスト処理を用いた XML アプリケーションプログラミングが可能になる．

また，これらの S 式は構文解析系等を使って XML 文書から機械的に構築することも可能であるが，プログラマが直接 S 式を記述することも簡便な方法となりうる．さらに，バッククォート（‘）を用いることによって，XML 文書の S 式表現の中に評価対象となる Lisp 式を自由に入れることにより，表現能力を広げることができる [5]．例えば，図 4 に書かれ

表 1: 関連研究

	汎用言語	ドメインスペシフィック言語
動的な妥当性の検証	**本研究**	
静的な妥当性の検証	Haskell	XDuce, XMLambda
妥当性の検証なし	Franz Inc. DOM	XSLT, XQuery, YTS

```
((:html :xmlns
        "http://www.w3.org/1999/xhtml")
 (:head (:title "Hello"))
 (:body (:h1 "Hello, World!")
        "I'm in Tokyo."))
```

図 3: S 式で表された XML 文書の例

```
`((:html :xmlns
         "http://www.w3.org/1999/xhtml")
  (:head (:title "Hello"))
  (:body (:h1 "Hello, World!")
         "You are "
         ,(increment-access-counter)
         " visitor."))
```

図 4: バッククォートとカンマを使った例

ている例で，式 (increment-access-counter) は，先頭にカンマ（,）が付いているので評価されるが，S 式全体としては，バッククォートが先頭についていることで評価されない．このように直接 S 式を書き，XML 文書自動生成系に渡せば，XML 文書の動的自動生成が容易に行われる．

XML 文書を表現している S 式の文法を形式的に書くと，次のようになる．

```
sexpr   ::= (element content*)
element ::= name
            (name attlist*)
attlist ::= name string
content ::= string
            lispCommand
            sexpr
```

ここで，name は Common Lisp でのキーワード，string は文字列，lispCommand はバッククォートされた S 式の中で評価したい Lisp の式である．

```
(:html
 (:head (:title $title))
 (:body (:h1 $title)
        $content))
```

図 5: XML 文書のパターンの例

5 XML 文書構造変換言語

この Lisp 上での XML 文書の表現方法を基に，XML 文書構造変換用の構文を Lisp に追加する．これはマクロを使って実現される[2]．

XML 文書の構造変換には，パターンマッチの技術が有効である．この言語では，XML 文書のパターンを，図 5 のようにして表す．ここで，$記号が先頭に現われる文字列はパターン変数を表している．図 5 の例だと，

```
(:html
 (:head (:title "Hello, World!"))
 (:body (:h1 "Hello, World!")
        "I'm in Tokyo."))
```

等がこのパターンにマッチし，パターン変数に対応する部分木が束縛される（この例では $title に "Hello, World!" が，$content に "I'm in Tokyo." がそれぞれ束縛される）．

このパターンを基に，以下の 3 つの文法を Common Lisp に組み込んだ．

- ルールの定義:

```
(defrule <name> <input pattern>
```

[2]構造変換を積極的に利用するアプリケーションの一つに，Web 出版がある．Lisp でも AllegroServe [3] や CL-HTTP[1] 等により，Java Servlet 同等の機能を使うことができる

```
<!ELEMENT document (header,content)>
<!ELEMENT header (title)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT content (paragraph|itemize)*>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT itemize (item)*>
<!ELEMENT item (#PCDATA)>
```

図 6: 変換プログラムで扱う DTD

```
                <output pattern>)
```

このマクロは <name> と名前をつけられた Common Lisp の関数を返す．<name> 関数は S 式で表現された XML 文書を受け取って，それが<input pattern> にマッチすれば <output pattern> に変換する．<output pattern> には任意の Lisp 式を書いて，複雑な計算をすることができる．

- ルールの選択:

```
(rule-set <pattern>
          <rule 1> <rule 2> ...)
```

このマクロは，<pattern> に対し，適切なルール<rule n> を適用する．それぞれの <rule n> は，defrule で定義されたルールでなければならない．どのルールを適用するかは，<rule n> の入力パターンにマッチするか否かで判断される．

- パターンへの繰り返し適用:

```
(for-each (<var> <list>)
          <iteration>)
```

このマクロは，Common Lisp の dolist に似ているが，<list> はパターンのリストである必要がある．任意の Lisp 式<iteration> を <list> の各要素 <var> にそれぞれ適用した結果をリストにして返す．

これらを用いて，図 6 で示した DTD を持つ XML 文書を HTML に変換するプログラムは，図 7 のようにして書くことができる．<output pattern> 中に直接 Lisp 式を書いて複雑な計算ができる点，また defrule は他の Lisp アプリケーションが直接呼び

```
(defrule document ((:document :|xml:lang|
                              $lang)
                   (:header (:title $title))
                   (:content $content))
  ((:html :|xml:lang| $lang)
   (:head (:title $title))
   (:body (:h1 $title)
    (for-each (i $content)
      (rule-set i
                (paragraph i)
                (itemize i))))))

(defrule paragraph (:paragraph $para)
  (:p $para))

(defrule itemize (:itemize $items)
  (:ul (for-each (i $items)
         (item i))))

(defrule item (:item $content)
  (:li $content))
```

図 7: 本ツールでの XML 構造変換プログラムの例

出すことのできる通常の Lisp 関数を返すため，Lisp との連携がシームレスである点が，他の XML 用言語には見られない特徴である．例えば，図 7 で定義した関数 document は次のようにして他の Lisp プログラムから呼び出すことが可能である．

```
(publish :path "/foo/bar"
  :content-type "text/html"
  :function
  #'(lambda (req ent)
      ...
      (generate-xml "html"
        (document (parse-xml p)))))
```

この例では，ストリーム p に送られた XML 文書を構文解析して S 式に変換した後，document 関数によって構文解析を行い，XML 文書 (テキスト) を自動生成して /foo/bar という URL パスで Web 出版している (ここでは AllegroServe[3] の関数 publish を使っている．:path で指定した URL を使って，MIME タイプ text/html で:function 以下の関数の結果を出力するという意味である)．

変換先の XML 文書の妥当性は，3 章で述べたスキーマ辞書を使うことにより，動的に保証することができる．XML のスキーマに変更点があった場合でも，スキーマ辞書に登録されてある情報を実行時に更新できるので，再コンパイルの必要がない．

6 XML文書構造変換言語のシャッフル表現への拡張

2001 年 11 月に，国際的な標準化団体 OASIS によって RELAX NG が制定された．これにより XML のスキーマとしてインターリーブ型が扱えるようになった．インターリーブ型の応用例として，順序のないレコード型が考えられる．例えば，文献データベースに，次のような書籍が登録されてあったとする．

```
<book>
 <author>Paul Graham</author>
 <title>ANSI Common Lisp</title>
 <publisher>Prentice Hall</publisher>
 <year>1996</year>
</book>
<book>
 <title>プログラミング言語 C</title>
 <author>カーニハン，リッチー</author>
 <translator>石田晴久</translator>
 <publisher>共立出版</publisher>
 <year>1989</year>
</book>
```

ここで，author，title，publisher，year は書籍の属性として必要だが，出現する順番は重要ではない．また，translator は場合によって必要になることがある．このような順序のないレコードは従来のスキーマ言語では記述が難しい．

本研究ではインターリーブ型を持った XML 文書に対するパターンマッチを実現させるため，上に述べた XML 文書構造変換言語を，正規表現の拡張であるシャッフル表現パターンが記述できるように拡張した．

6.1 シャッフル表現

まず，シャッフル表現を以下に定義する．

$\sum$ をアルファベットの集合，λ を空語とする．シャッフル（インターリーブ）オペレータ $\odot$ は次のように定義される．

- 全ての $u \in \sum^*$ について，$u \odot \lambda = \lambda \odot u = \{u\}$.
- 全ての $u, v \in \sum^*$，$a, b \in \sum$ について，$au \odot bv = a(u \odot bv) \cup b(au \odot v)$.

全ての言語 $L_1, L_2 \subset \sum^*$ について次が成り立つ．

$$L_1 \odot L_2 = \bigcup_{u \in L_1, v \in L_2} u \odot v$$

正規表現にシャッフルオペレータを加えて拡張したものも，言語のクラスとしては正規であることが証明されている（並列な状態遷移機械を一つの状態遷移機械でシミュレートできることから明らか）[4].

全ての言語 $L \in \sum^*$ について，シャッフル閉包オペレータ $\otimes$ は以下のように定義される．

$$L^{\otimes} = \bigcup_{i=0}^{\infty} L^{\odot i}, \text{ ただし } L^{\odot 0} = \{\lambda\}, L^{\odot i} = L^{\odot i-1}$$

シャッフル閉包を含むシャッフル言語のクラスは文脈依存に含まれ，文脈自由よりも真に大きいことが知られている [7]．以下に，シャッフル表現とシャッフル言語の定義をまとめる．

定義（シャッフル表現）： $a \in \sum$，λ，$\emptyset$ はそれぞれシャッフル表現である．S_1, S_2 がシャッフル表現のとき，$(S_1 \cdot S_2)$，S_1^*，$(S_1 + S_2)$，$(S_1 \odot S_2)$，$S_1^{\otimes}$ はシャッフル表現であり，それ以外はシャッフル表現でない．

定義（シャッフル言語）： シャッフル表現 S から生成される言語 $L(S)$ は次のように定義される．まず，$L(a) = \{a\}$，$L(\lambda) = \{\lambda\}$，$L(\emptyset) = \emptyset$．$L(S_1) = L_1$ で $L(S_2) = L_2$ のとき，$L(S_1 \cdot S_2) = L_1 \cdot L_2$，$L(S_1 + S_2) = L_1 \cup L_2$，$L(S_1^*) = L_1^*$，$L(S_1 \odot S_2) = S_1 \odot S_2$，$L(S_1^{\otimes}) = L_1^{\otimes}$.

なお，シャッフル閉包は RELAX NG でも扱えないが，言語の表現能力をさらに拡張する目的で，本研究ではこれについても考えることにする．

6.2 記法

以後，シャッフル表現パターンを S 式で書く．左側に書かれてあるものは文書型定義に現われる正規表現，及びシャッフル表現である．

```
a,b,c --> (seq (:a $a) (:b $b) (:c $c))
a|b --> (or (:a $a) (:b $b))
a?  --> (?  (:a $a))
a* --> (* (:a $a))
a+ --> (+ (:a $a))
a⊙b --> (% (:a $a) (:b $b))
a⊗ --> (& (seq (:a $a) (:b $b)))
```

シャッフル表現のアルファベットはS式（木）である．つまり，木として同じものであればそれらは同じアルファベットとして解釈される．なお，パターン変数には任意の（部分）木が代入されるので，(:a (:b "foo")) は(:a $a) で表されるアルファベットの集合の一要素であると解釈される．

この章の冒頭で述べたXML文書にマッチするシャッフル表現パターンは，次のように書ける．

```
(:book (% (:author $author)
          (:title $title)
          (:publisher $publisher)
          (:year $year)
          (? (:translator $translator))))
```

6.3 実装

シャッフル表現を受理するオートマトンとして，シャッフルオートマトンが提案されている [7]．本研究ではこのシャッフルオートマトンをベースにした，より簡素化したアルゴリズムを用いてシャッフル表現を受理しており [19]，状態数，計算時間はシャッフルオートマトンと比べてある程度抑えられているのではないかと期待している．ただしシャッフルオペレータ及びシャッフル閉包オペレータの入れ子が書けないという制限がある．

また，閉包 *，+，& のセマンティクスに注意が必要である．例えば，

```
(:a (* (:e $e)))
```

というパターンには

```
(:a (:e "e1") (:e "e2") (:e "e3"))
```

などがマッチし，変数束縛は次のようになる．

```
$e = ("e1" "e2" "e3")
```

その他の実装上の留意点として，パターンマッチの曖昧性の問題がある．例えば次のようなパターン

```
(seq (* (:a $foo)) (* (:a $bar)))
```

に ((:a "a1") (:a "a2") (:a "a3")) をマッチさせようとすると，マッチの仕方に複数通り存在する．ここでは，ほとんどの関数型言語がそうであるように，常に左側のパターンが優先されるという方針をとる．この例では変数 $foo には ("a1" "a2" "a3") が，$bar には nil が束縛される．

7 結論及び今後の研究課題

本研究では，パターンマッチを使ったXML文書構造変換言語をLispのマクロを使ってLisp言語の拡張として実現したことにより，XML文書構造変換言語とLispとのシームレスな結合が実現された．これによりLisp以外の言語を特に意識しなくともXMLアプリケーションの開発が可能になる．またこのパターンにはシャッフル表現を記述することができ，高い表現能力を持つ．これらのことから，本研究は今後のXMLアプリケーション開発の生産性向上に大きく貢献するものであると期待できる．

また本研究では妥当性の検証を動的に行い，スキーマが変更された場合でもシステムを再コンパイルすることなく新しいスキーマを使って妥当性の検証ができる．このことによりシステムの運用コストを下げることも可能になると期待できる．

今後はこのツールを使った大規模なアプリケーションの構築を行い，ツールの有用性を検証する必要がある．

8 謝辞

情報処理振興事業協会からは，平成12年度未踏ソフトウェア創造事業において，本研究への資金を提供していただきました．そのときにプロジェクトマネージャをして頂いた湯淺太一氏には，大変お世話になりました．また，増原英彦氏，五十嵐淳氏及び田中哲朗氏からは，本研究における初期段階から，たくさんの助言やコメントを頂きました．それぞれに心より感謝します．

参考文献

[1] CL-HTTP. http://wilson.ai.mit.edu/cl-http/cl-http.html.

[2] Franz Inc. A Lisp Based XML Parser. http://www.franz.com.

[3] Franz Inc. AllegroServe. http://allegroserve.sourceforge.net.

[4] Vijay K. Garg and M. T. Raghunath. Concurrent regular expressions and their relationship

to petri nets. *Theoretical Computer Science*, 96:285–304, 1992.

[5] Paul Graham. *ANSI Common Lisp*. Prentice Hall, 1996.

[6] Haruo Hosoya and Benjamin Pierce. XDuce: A Typed XML Processing Language. In *Proceedings of Third International Workshop on the Web and Databases (WebDB2000)*, pages 226–244, May 2000.

[7] Joanna Jedrzejowicz and Andrzej Szepietowski. Shuffle languages are in P. *Theoretical Computer Science*, 250:31–53, 2001.

[8] Tetsuo Kamina, Taiichi Yuasa, and Tetsuo Tamai. A Light-Weight Programming Interface for XML. In **第 4 回インターネットテクノロジーワークショップ** *(WIT2001)*, 2001 年 9 月.

[9] Erik Meijer and Mark Shields. XMlambda: A Functional Language for Constructing and Manipulating XML Documents. Submitted to USENIX 2000 Technical Conference, 1999.

[10] W3C Web Site. XML Schema. http://www.w3.org/XML/Schema.

[11] W3C Web Site. Extensible Markup Language (XML) 1.0. http://www.w3.org, 1998.

[12] W3C Web Site. Document Object Model (DOM) Specification. http://www.w3.org, 2000.

[13] W3C Web Site. XSL Transformations (XSLT) Version 1.0. http://www.w3.org, 2000.

[14] W3C Web Site. XQuery 1.0 and XPath 2.0 Data Model. http://www.w3.org/TR/query-datamodel/, 2001.

[15] Malcolm Wallace and Colin Runciman. Haskell and XML: Generic Combinators or Type-Based Translation? In *Proceedings of the Fourth ACM SIGPLAN International Conference on Functional Programming (ICFP '99)*, pages 148–159, September 1999.

[16] YTS – Yet another Transformation Sheet. http://www.flexfirm.co.jp/tech-field/yts/.

[17] 紙名哲生, 玉井哲雄. Lisp を基にした新しい XML プログラミングツール実現手法. In *2002* **年情報学シンポジウム講演論文集**, pages 39–46, 2002 年 1 月.

[18] 村田 真. XML スキーマ言語 RELAX NG. In *2002* **年情報学シンポジウム講演論文集**, pages 33–38, 2002 年 1 月.

[19] 紙名 哲生. 新しい XML プログラミングツールの実現法に関する研究. 東京大学大学院 総合文化研究科 広域科学専攻 広域システム科学系修士学位論文, 2002 年.