

Lisp を基にした新しい XML プログラミングツール実現手法

紙名 哲生

玉井 哲雄

東京大学大学院 総合文化研究科 広域システム科学系

〒 153-8902 東京都目黒区駒場 3-8-1 15 号館

{kamina,tamai}@graco.c.u-tokyo.ac.jp

概要

XML アプリケーション開発は DOM (Document Object Model) に基づいて行われることが一般的であるが、DOM は開発者にとって必ずしも扱いやすいものであるとは言えない。そこで、DOM の代案として、Lisp を基にした XML プログラミングツールの実現手法を提案し、それに基づくツールを実装した。この手法によって、XML アプリケーション開発は、Lisp の経験者であれば容易なりスト処理で行うことが可能になる。インターフェイスを介さず、直接 Lisp で扱えることによる柔軟性も得られる。また、S 式をプログラム中に直接書くことによって、プログラムによる XML 文書の動的自動生成も容易になる。さらにこの手法を基に、関係データベースから XML 文書へのマッピング機能と XML 文書構造変換言語を開発した。

1 はじめに

XML [13] は DTD (Document Type Definition) によって文書に型をつけることができるので、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。

XML 文書はテキスト形式なので、プログラムが直接操作するデータ形式としては適していない。そこで、XML 文書を構文解析して、プログラムが直接扱えるデータ表現に変換するようなツールが必要になってくる。現在ではそのようなツールの大半は、DOM (Document Object Model) [14] に基づいて作られている。DOM では文書をオブジェクトの木としてモデル化し

ており、開発者には DOM オブジェクトへのインターフェイスに対する知識が要求される。このことは、開発者にとっての、XML アプリケーションプログラミングに対する敷居を高くしている。

一方で、XML 文書をインターフェイスを通してしか操作できないオブジェクトに変換するのではなく、通常のプログラミング言語で直接扱えるようなデータ表現に変換することによって、その言語上に「組み込む」というアプローチが考えられる。このアプローチにより、複雑なインターフェイスを覚えなくても XML アプリケーションの開発が可能になる。またインターフェイスは、必要であれば目的に応じて簡単に構築することができ、それにより高い柔軟性も得られる。そこで本研究では、XML 文書が組み込まれるホスト言語に Lisp を採用し、Lisp に基づく XML プログラミングツールを実装した¹。Lisp の S 式は XML と構造が非常によく似ており、S 式を使って XML 文書を簡単に表現することができる。これにより、XML アプリケーション開発は Lisp の経験者であれば容易なりスト処理で行うことが可能になる。また、Lisp にはデータとプログラムを同じ構造 (S 式) で表すという特徴があり、この特徴を活かして、プログラムによる XML 文書の動的自動生成も容易に行えるようになる。

この論文では、まず最初に S 式を用いてどのようにして XML 文書を表現するかを示す。しかし S 式だけでは、XML の大きな特徴である妥当性の検証ができない。そこで次に、この手法でどのようにして妥当性の検証をしているか、またその妥当性の検証方法にどのような利点があるかを示す。次に、この手法を基にした XML 関連技術として、関係データベースの内容

¹この XML プログラミングツールは、<http://www.graco.c.u-tokyo.ac.jp/~kamina/xmltools/>からダウンロードすることができる。

```
<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 1: XML 文書の例

を XML 文書に自動変換するツールと、XML 文書を構造変換する言語を紹介する。

2 S 式による XML 文書の表現

2.1 XML 文書の表現方法

この手法で、XML 文書を S 式によってどのように表現するかを述べる。S 式による正確な定義は、付録 A に載せてある。また、XML 構文解析系や XML 文書自動生成系などのインターフェイスは文献 [16] で報告している。ここではまず例を挙げる。この手法では、図 1 に書かれている XML 文書は、S 式を用いて図 2 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 (car、cdr、cons など) 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

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

図 2: 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 3: バッククォートとカンマを使った例

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 [4]。例えば、図 3 に書かれている例で、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。

2.2 DOM との比較

ここで、DOM と本手法との相違点を見ていきたい。まず、DOM においてはインターフェイスを通じて XML 文書进行操作するが、本手法ではインターフェイスを介さなくとも、Lisp で直接 S 式を扱うことができる点があげられる。すなわち、DOM を用いる場合、DOM オブジェクトがどのような木構造を構成することによ

って XML 文書表現しているかや、`getTagName()`、`getValue()`、`getElementByTagName()` などのインターフェイスがどのように振舞うかなどの正確な知識が必要であり、開発者が正確にそれらについて覚えていない場合は、適宜マニュアル等を参照していかねばならないのに対し、本手法においては、XML 文書が Lisp 上での基本的なデータ表現形式である S 式によって表されていることにより、Lisp の経験のある開発者は XML 構文解析系などのインターフェイスを覚えるだけでよい。もちろん、インターフェイスを介して操作したほうがプログラムの理解には役立つことが多いが、そのようなインターフェイスを構築することは容易に行うことができ、むしろ目的に応じたインターフェイスを開発者自らカスタマイズできるという柔軟性が本手法にはある。

また、XML 文書を S 式で表現すると、Lisp の特性をいかしたプログラミングが可能になる。図 3 のようなプログラムによる XML 文書の自動生成が容易に行われるのは、データとプログラムを区別せずに、S 式で表された XML 文書を直接プログラム中に書けるという特色が Lisp にあるからである。一方 DOM オブジェクト木をプログラミングによって一から構築していくのは、大変な作業になるであろう。

3 妥当性の検証

実際には、S 式を用いて XML 文書表現するだけでは解決できない問題がある。それは、XML の重要な特徴である妥当性の検証が、S 式だけでは不可能であるということである。DOM の場合、各オブジェクトが型を持っており、それぞれのオブジェクトが表す要素の名前は何かとか、子供のノードとして持つことのできるオブジェクトの型は何かなどの情報を持たせることができる。しかし、S 式は単純なセルとポイントだけの構造である(図 4)。そこでこの節では、本手法を用いる中でいかにして妥当性の検証を可能にしているかを述べる。

妥当性の検証を可能にするために、本手法では XML 要素サーバというものを導入した。これは、Lisp プロセスの実行メモリ空間上に常駐している、DTD の内容を保持した辞書のようなものである。具体的には、文書型名をキーに、文書型を表す CLOS (Common Lisp Object System) のインスタンスを値にした連想リスト

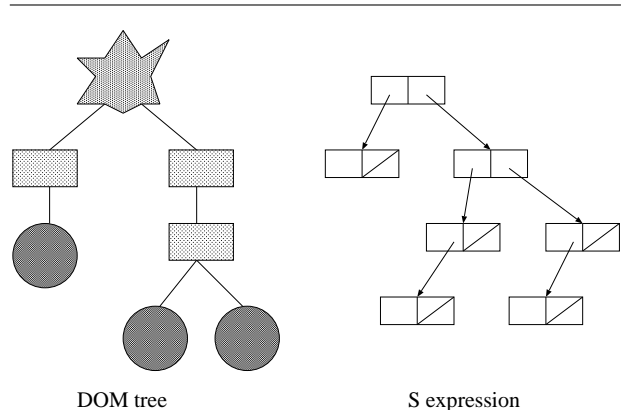


図 4: DOM 木と S 式

```
<!DOCTYPE xdoc [
...
<!ELEMENT foo (a,(b|c)*)+>
...
]>
```

図 5: XML 要素宣言の例

である。文書型クラスのインスタンスには、XML 要素や実体、その他文書型の属性を表すのに必要な情報が入っている。XML 要素等もまた CLOS のインスタンスとして表されている。例えば、図 5 に書かれている文書型定義中に宣言されている `foo` XML 要素は、図 6 のような CLOS インスタンスで表される。

図 6 中の `children` スロット (メンバ) は、妥当性検証で重要な役割を果たす。これは、図 5 の XML 要素宣言に書かれている正規表現を、決定性有限オートマトンに変換したものである²。 `begin-state` スロットと `final-states` スロットは、それぞれオートマトンの初期状態と終了状態である。XML プロセッサは、実行時にこのオートマトンを実行することにより、妥当性を検証することができる。

例えば、XML プロセッサがある XML 文書の妥当性を検証しようとしているとする。XML プロセッサはま

²図 6 のオートマトンは `(a,(a|b|c)*)*` を変換したものにみえるが、`(a,(b|c)*)+` を変換したものと等価である。

```

name:      :foo
attlist:   nil
children:  (((6 :a) 7)
            ((7 :a) 7)
            ((7 :b) 7)
            ((7 :c) 7))
begin-state: 6
final-states: (7)
doctype:   "xdoc"

```

図 6: CLOS インスタンスの例

ず、XML 文書の文書型宣言の名前を使って、XML 要素サーバに問い合わせる。もし対応する文書型 CLOS インスタンスが見つければ、XML プロセサはそれを使う。しかしもしそれが見つからなかった場合、XML プロセサは文書型宣言のシステム識別子や公開識別子を用いて、その DTD ファイルを探しに行き、それを XML 要素サーバに登録する。一度登録すれば、次からは DTD を構文解析する必要はない。

これは、XML プロセサが実行時に妥当性の検証を行うので、動的な妥当性の検証である。5 章で述べるように、妥当性の検証方法には、コンパイル時に妥当かどうかを検出できる静的な方法もある。しかし、Web サービスなどの運用コストを下げる目的には、動的な方法に利点がある。例えば、システムで使用している DTD に変更が加えられたり、新しい文書型に関する処理をシステムに追加したりする時に、システムを再コンパイルしたり、停止したりすることなしに変更や追加を行うことが可能になる。また、XML 文書は必ずしも妥当である必要はなく、妥当でなくとも使用することができるということが XML1.0 勧告 [13] でも定められている。つまり XML プログラミングツールには、妥当性の検証を行うか否かをユーザのオプション指定で行えるようにする必要があるが、その指定の変更も、システムを停止することなく行うことができる。

4 本手法に基づく XML 関連技術

4.1 RDB から XML 文書への変換

XML を使って関係データベースを表現する試みにはすでに数多くのものである [11, 3]。便利なツールも既にいくつか存在するが、ここでは、本手法を基に、そのようなツールを非常に簡単に構築できることを示す。なお、ここでは Lisp の処理系からデータベースへの接続を行うのに、Allegro ODBC インターフェイスの使用 [7] を想定している。

Allegro ODBC では、例えば図 7 のようなデータベースへの問い合わせを行うと、以下のようなリストを返してくる。

```

(("Paul Graham" "ANSI Common Lisp")
 ("Bob DuCharme"
  "XML: The Annotated Specification"))
("author" "title")

```

これを、図 8 のような S 式に変換するのは、単純なリスト処理によって実現できる。ここで、関係データベースから XML 文書へのマッピング規則がどのようになっているかが問題になる。本ツールでは以下に示した規則で行っているが、可能なマッピング規則はこの限りではない。

- 文書型名は ODBC データソース名から得られる。
- ルート要素 data-source は、DTD 中で以下のよう宣言された XML 要素として得られる:
`<!ELEMENT data-source (table_1| ... |table_n)+>`
- 各テーブル table_i は以下のように宣言された XML 要素として得られる:
`<!ELEMENT table_i EMPTY>`
- 各テーブルのカラムは、以下のように宣言された属性として得られる:
`<!ATTLIST table_i column_j CDATA #IMPLIED>`

このような単純なリスト処理による実現方法は、データベースからオブジェクトの木を構築する場合と比較すると、非常に簡潔であることがわかる。

データソース名: booklist

テーブル名: table

author	title
Paul Graham	ANSI Common Lisp
Bob DuCharme	XML: The Annotated Specification

図 7: booklist データベース

```
(:booklist
  ((:book
    :author "Paul Graham"
    :title "ANSI Common Lisp"))
  ((:book
    :author "Bob DuCharme"
    :title "XML: The Annotated
           Specification"))))
```

図 8: S 式で表現されたデータベース

4.2 XML 構造変換言語

データ交換等でよく用いられる XML にとって、構造変換は非常に重要な技術である。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。その変換ルールは XML 文書を構造変換するための言語を用いて記述することができ、現在では XSLT [15] などが広く用いられている。ここでは、本手法をベースに、パターンマッチの技術を使って、XML 文書の構造変換言語を、マクロを使って簡単に Common Lisp に組み込むことができることを示す。

この言語では、XML 文書のパターンを、図 9 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 9 の例だと、

```
(:html
  (:head (:title "Hello, World!"))
```

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

図 9: XML 文書のパターンの例

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

等がこのパターンにマッチする。このパターン中には、(? ...) や(or ...) という記法も許しており、それぞれ正規表現における?、|に対応している。

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>
  <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。

- 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。

これらを用いると、例えば図 10 で書かれた XSLT プログラムと同等のプログラムは図 11 のようにして書くことができる。関数型言語に特徴的なパターン照合の技術を使っているため、変換元の XML 文書と変換後の文書の対応関係が直観的に理解でき、プログラムのソースも簡潔になっている。この他に、Common Lisp に組み込まれているため、マクロを使って新しい文法を追加することができるなどの拡張性があり、また XSLT では難しい複雑な計算なども容易に行うことができるという利点がある。

5 関連研究

Wallace と Runciman は、XML 文書処理に Haskell を使う方法を提案している [17]。本研究同様、これは XML 文書処理を Haskell に埋め込ませるというものである。本研究と違うところは、Haskell の型システムを用いて、DTD から Haskell 上のデータ型へのマッピングによって XML 文書の妥当性を静的に保証している点である。

XDuce [5] 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 [10] も静的な XML 文書処理用の関数型言語である。

これら静的なアプローチに対し、本研究では動的に妥当性の検証を行う。静的な場合、例えば変換先の XML 文書が妥当であるか否か等の検査がコンパイル時に行われるため、テストにかかるコストを抑えることができる等の利点がある。しかし、DTD に変更が加えられたり、新しい文書型に関する処理をシステムに追加するときなどは、一度システムを停止して、再コンパイルしたものを起動しなおす作業が必要である。妥当性の検証を行うか否かのポリシーを変更する場合も同様である。一方動的な場合、システムを停止することなく DTD の変更、新しい文書型の変更などを行うことができる。このことは、Web サービスなどのサービスの運用コストを抑えることに役立つ。Common Lisp は Web サーバプログラミングに適した言語であると

```
<xsl:stylesheet version="1.0"
  xmlns:xsl=
    "http://www.w3.org/1999/XSL/Transform"
  xmlns=
    "http://www.w3.org/1999/xhtml">
<xsl:template match="/document">
<html>
  <xsl:for-each select="@xml:lang">
    <xsl:copy/>
  </xsl:for-each>
<head>
  <title>
    <xsl:value-of
      select="header/title"/>
  </title>
</head>
<body>
  <xsl:apply-templates
    select="header/title"/>
  <xsl:apply-templates
    select="content"/>
</body>
</html>
</xsl:template>

<xsl:template
  match="/document/header/title">
<h1>
  <xsl:value-of select="."/>
</h1>
</xsl:template>

<xsl:template
  match="/document/content">
<xsl:apply-templates/>
</xsl:template>

<xsl:template match="paragraph">
<p>
  <xsl:value-of select="."/>
</p>
</xsl:template>

<xsl:template match="itemize">
<ul>
  <xsl:for-each select="./item">
    <li>
      <xsl:value-of select="."/>
    </li>
  </xsl:for-each>
</ul>
</xsl:template>
</xsl:stylesheet>
```

図 10: XSLT プログラムの例

```

(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 11: 本ツールでの XML 構造変換プログラムの例

考えられ [8, 2]、本研究も Web サービスなどのサーバアプリケーション開発を対象にしているため、動的なアプローチをとった。

本研究同様、Lisp を基にした XML プログラミングツールに、Franz Inc. の XML 構文解析系がある [6]。これは本研究同様、XML 文書を S 式に変換して扱うものであり、S 式の文法も本研究のものと非常によく似ている。このように、XML 文書を S 式で表現する方式は企業などでもサポートを始めており、この分野の今後の需要はより一層高まってくるものと思われる。一方で Franz Inc. の構文解析系は妥当性を検証しないものであり、S 式ベースで妥当性の検証を行ったところに本研究の特色がある。

6 結論

本研究では、DOM の代案として、XML 文書を通常のプログラミング言語で直接扱えるようなデータ構造に変換するという XML プログラミング手法を提案し、具体的な言語として Lisp を採用した。XML 文書は Lisp の S 式で自然に表現され、単純なリスト処理での XML アプリケーション開発が可能になった。ま

たデータとプログラムを区別しないという Lisp の特徴をいかし、直接プログラム中に S 式を書くことでプログラムによる XML 文書の自動生成が容易になることを示した。妥当性の検証方法についても議論し、S 式ベースでも動的に妥当性の検証が可能であるだけでなく、この妥当性検証方法による利点が存在することも示した。また本手法を用いて構築された XML の関連技術として、データベースから XML 文書へのマッピング機能と XML 文書構造変換言語が示された。以上のことから、Lisp は XML プログラミングツールを実現する際のベースとして適格であるということが言える。

7 今後の課題

XML のスキーマ言語は DTD の他に、より強力な XML Schema [12] や RELAX [1] がある。これらへの対応が望ましい。また、本手法における XML 文書構造変換言語はパターン照合を使っているが、記述できるパターンは限られている。パターン部分に正規表現やシャッフル表現 [9] などを記述できれば、この言語の表現能力はより広がるものと考えられる。シャッフル表現とは、正規表現にシャッフル (インターリーブ) 演算子などを導入して拡張したもので、並行モデルをシーケンシャルに表現するときに、便利となる記法である。

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浅太一氏には、大変お世話になりました。また、増原英彦氏及び五十嵐淳氏からは、本ソフトウェア開発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。また情報学シンポジウムの査読者の方々から、たくさんのコメントを頂きました。それぞれに感謝します。

参考文献

- [1] ISO/IEC DTR 22250-1. Document Description and Processing Language – Regular Language Description for XML (RELAX) – Part1: RELAX Core, 2000.
- [2] CL-HTTP. <http://wilson.ai.mit.edu/cl-http/cl-http.html>.
- [3] Oracle Corporation. XML SQL Utility (XSU). <http://www.oracle.com>.
- [4] P. Graham. *ANSI Common Lisp*. Prentice Hall, 1996.
- [5] H. Hosoya and B. Pierce. XDuce: A Typed XML Processing Language. In *Proceedings of Third International Workshop on the Web and Databases (WebDB2000)*, pages 226–244, May 2000.
- [6] Franz Inc. A Lisp Based XML Parser. <http://www.franz.com>.
- [7] Franz Inc. Allegro ODBC. <http://www.franz.com>.
- [8] Franz Inc. AllegroServe. <http://allegroserve.sourceforge.net>.
- [9] J. Jędrzejowicz and A. Szepietowski. Shuffle languages are in P. *Theoretical Computer Science*, 250:31–53, 2001.
- [10] E. Meijer and M. Shields. XMLambda: A Functional Language for Constructing and Manipulating XML Documents. Submitted to USENIX 2000 Technical Conference, 1999.
- [11] W3C Web Site. XML representation of a relational database. <http://www.w3.org/XML/RDB.html>.
- [12] W3C Web Site. XML Schema. <http://www.w3.org/XML/Schema>.
- [13] W3C Web Site. Extensible Markup Language (XML) 1.0. <http://www.w3.org>, 1998.
- [14] W3C Web Site. Document Object Model (DOM) Specification. <http://www.w3.org>, 2000.
- [15] W3C Web Site. XSL Transformations (XSLT) Version 1.0. <http://www.w3.org>, 2000.
- [16] T. Kamina, T. Yuasa and T. Tamai. A Lightweight Programming Interface for XML. In *Proceedings of The Fourth Workshop on Internet Technology (WIT2001)*, September 2001.
- [17] M. Wallace and C. 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.

A XML 文書の S 式表現の文法

本研究の中で使われている XML 文書の S 式表現の文法は以下の通りである。

```
S_Expression ::= '( Element Space
                  Content )'
Element ::= ElementName |
           '( ElementName Space
             AttributeList )'
AttributeList ::= ( AttributeName
                  Space Value )*
Content ::= ( String | LispCommand |
            S_Expression )*
ElementName ::= Keyword
AttributeName ::= Keyword
```

ただし、Space は XML1.0 勧告 [13] の中で定義されている空白と同じである。ListCommand はバッククオートされた S 式の中で評価したい Lisp の式である。Keyword は Common Lisp のキーワードパッケージの中で'intern'されたシンボルである。