S式を用いたXMLサーバプログラミングツール

A Lisp Based XML Server Programming Tool

紙名 哲生
Tetsuo KAMINA

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 E-mail: kamina@graco.c.u-tokyo.ac.jp)

ABSTRACT. Current XML processors tend to be based on DOM (Document Object Model) specification. However, DOM objects require complicated interface to process them. We proposed and implemented a Lisp based XML programming toolkit as an alternative to DOM. It represents XML documents as S expressions internally; therefore, XML application programs can be simply coded as list processing, and we can make use of advantage of using Lisp such as treating data and programs uniformly. Based on this programming toolkit, we also embedded an XML transformation language in Common Lisp. Applying pattern matching to S expressions representing XML documents, the notations of this transformation language are concise, and the syntax of this language is simple and extensible.

1. 背景

現在では、電子商取引を始めとした、企業全体の業務をインターネット上で行う e-ビジネスの実現が求められている。これには、インターネット上で広く提供されるサービス同士を相互運用するための技術が必要であり、XML (eXtensible Markup Language) [14] が注目されている。XML 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 のフレームワークを用いて記述されたマークアップ言語として、例えば以下のようなものがある:

- XSLT [16]: データの構造変換用言語
- SVG [10]: グラフィックデータの記述
- SOAP [11]: 分散システムにおけるプロトコル
- WSDL [12]: Web サービスの定義

XML 文書そのものはテキスト形式なので、プログラムから直接扱うのには適していない。そこで、XML 文書を読み込んで、プログラムが直接扱えるデータ表現に変換するようなライブラリが必要になってくる。現在ではそのようなライブラリの大半は、DOM (Document Object Model) [15] 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というアプローチが考えられる。例えば、Lisp の S 式は構造が XML と非常によく似ている。また、Common Lisp は Web サービスの構築に適していると考えられ [8, 2]、Lisp 上で Web サービスを XML 対応に出来るような XML プログラミングツールの開発が望まれる。

2. 目的

以上の背景に基づき、本テーマでは、Lisp の S 式による XML 文書の表現方法を提案し、それに基づく XML プログラミングツールを実装することを目的とする [17, 19][1]。これにより、開発者は DOM を覚えることなく、単純なリスト処理によって XML アプリケーション開発を行えるようになる。また、このツールを Lisp から使用できるライブラリとして実装することにより、Lisp で扱えることによる高い柔軟性が得られる。さらに、このツールを基にした S 式と、パターンマッチの技術を組み合わせて用いることにより、XML 文書の構造変換などの基本的な XML 文書処理が簡単に実現可能になる。

[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。

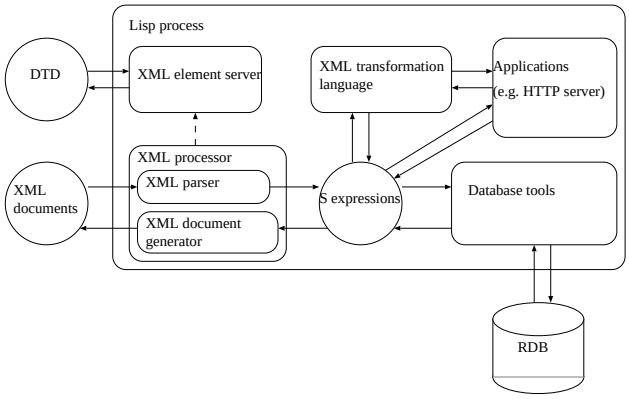

図 1: 全体構成図

この論文では、全体構成を示したあと、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、実は XML の大きな特徴である妥当性の検証ができない。そこで次の節で、本ツール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、本ツールを基にした XML 関連技術として、関係データベースの内容を 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。特に XML 文書を構造変換する言語は、S 式表現とパターンマッチの技術を用いることにより、従来技術の XSLT よりも簡潔で強力なものになっている。

3. 全体構成

このツールは、XML 文書と S 式との間の変換を行う XML プロセサ (XML 構文解析系及び XML 文書自動生成) と DTD の内容を Lisp の実行メモリ上に常駐させておく XML 要素サーバ、そしてその上で動作する、関係データベースから XML 文書へのマッピングツールと XML 文書構造変換言語から成る。XML 要素サーバは妥当性の検証に必要である。それぞれのツールを操作するインターフェイスは付録 C に示す。

Common Lisp は Web サービスの構築に適しており [8, 2]、本ツールを Common Lisp にロードすることにより、XML 対応した Web サービスの開発が容易に行えるようになる。HTTP サーバと本ツールのコードは、同一プロセス上で実行可能である (図 1)。

4. S 式による XML 文書の表現

まず最初に、このツールの中で、XML 文書が S 式によってどのようにして表現されているかを見ていく。正確な S 式による定義は、付録 A に載せてある。まず例を挙げれば、以下に示す操作により、図 2 で書かれている XML 文書 `hoge.xml` は、S 式を用いて図 3 のように表現される:

```
> (with-open-file (p "hoge.xml"
                     :direction :input)
    (parse-xml p :validate t))
```

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。

これらの S 式はプログラミングによって (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 [4]。例えば、図 4 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、XML 文書を表す 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 (`generate-xml-with-stream` のインターフェイスについては付録 C で述べる)。

図 4 から、例えば以下のような XML 文書が生成さ

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

図 2: XML 文書の例 (`hoge.xml`)

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

図 3: S 式で表された XML 文書の例

```
(generate-xml-with-stream
 t "html"
 `((:html :xmlns
          "http://www.w3.org/1999/xhtml")
   (:head (:title "Hello"))
   (:body (:h1 "Hello, World!")
          (:p "You are "
              ,(increment-access-counter)
              " visitor.")))
 :validate t
 :system
 "/home/users/kamina/xhtml1-strict.dtd")
```

図 4: バッククォートとカンマを使った例

れる:

```
<!DOCTYPE html SYSTEM
 "/home/users/kamina/xhtml1-strict.dtd">
<html xmlns=
      "http://www.w3.org/1999/xhtml">
 <head>
  <title>Hello</title>
 </head>
 <body>
  <h1>Hello, World!</h1>
  <p>You are 1234th visitor.</p>
 </body>
</html>
```

ここで、従来技術の DOM と比較して本ツールはどこが違うかを見ていきたい。まず、DOM を使いこなすには、DOM オブジェクトがどのようにして木構造を構成することによって XML 文書を表現しているかや、`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、DOM インターフェイスに対する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なければならない。一方、本ツールにおいては、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、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、後は他の分野で培われてきたプログラミング技術をそのまま XML アプリケーション開発に応用することができる。また、DOM オブジェクトの場合には、上のようなインターフェイスを介すことでしか XML 文書の内部表現にアクセスすることはできないが、本ツールにおいては、インターフェイスではなく Lisp 言語の基本的構成要素によって XML 文書の内部表現にアクセスするため、それらのようなインターフェイスを、目的・要求に応じて開発者自らカスタマイズできるという柔軟性がある。さらに、Lisp はデータとプログラムを区別しないという特色があるので、S 式で表された XML を直接プログラム中に書くことも可能である。よって図 4 のようなプログラミングも可能になるのである。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。

5. 妥当性の検証

今まで S 式を用いて XML 文書を表現することによって、いかに開発者の覚えるインターフェイスが軽量になり、柔軟にさまざまな要求に対応していくことが可能になるかを議論してきた。しかし実際には、これ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、このまま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、子供のノードに来ていい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タだけの構造である (図 5)。そこでこの節では、本ツール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、本ツールでは 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

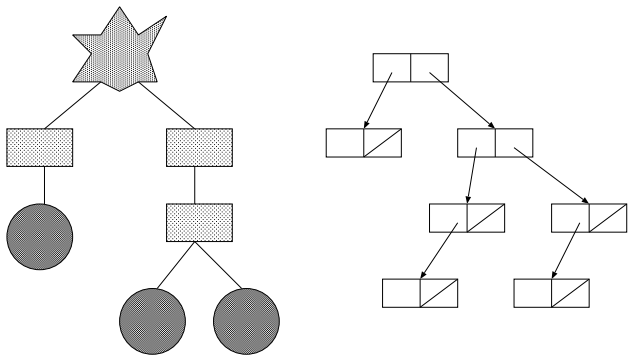

図 5: DOM 木と S 式

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

図 6: XML 要素宣言の例

を保持した辞書のようなものである。例えば、図 6 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 7 のような CLOS インスタンスで表される。

図 7 中の`children` スロット (メンバ) は、図 6 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。

ここで、XML プロセサがある XML 文書の妥当性を検証しようとしていると考える。XML プロセサはまず、

```
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 7: CLOS インスタンスの例

[2]図 7 のオートマトンは`(a,(a|b|c))*`を変換したものに見えるが、`(a,(b|c)*)+`を変換したものと等価である。

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かれば、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。7 章で述べるように、世の中には静的にこれを行うものもあるが、動的な方法には次の利点がある。まず、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 [14] 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。このことは、Web サービスなどの運用コストを抑えることに役立つ。

6. 本ツールを基にした XML 関連技術

本ツールでの XML プロセサを基にして、いかに XML 関連技術が簡単に構築されているかを示す。まず、関係データベースの内容を XML 文書に変換するツールを紹介し、次に XML 文書を構造変換する言語を紹介する。

(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がある [13, 3]。ここでは、本ツールを基に、この変換をリスト処理によって簡単に行うことができることを示す。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 [7] を使っている。

まず、本ツール上で、データベースのテーブルから XML 文書への変換が、どのような規則に基づいて行われているかを述べる。変換の規則は、以下のようになる:

- 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 中で以下のように宣言された 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>`

Allegro ODBC では、例えば図 8 のようなデータベースへの問い合わせを行うと、以下のような結果を返してくる:

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

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 8: 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 9: S 式で表現されたデータベース

これを、上記の変換規則に従って、図 9 のような S 式に変換するには、単純に以下のようなプログラムでよい:

```
(multiple-value-bind (rows cols)
    (dbi:sql query :db db)
  (let (retval)
    (push data-source-name retval)
    (dolist (r rows)
      (push (list (let (item)
                    (push table-name item)
                    (dolist (c cols)
                      (push c item))
                    (reverse item)))
            retval))
    (reverse retval)))
```

DOM を用いる場合には、データベースから DOM 木を構築していかなければならないことを考えると、このような単純なリスト処理による実現は非常に簡潔であることがわかる。

(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。というのは、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

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

図 10: XML 文書のパターンの例

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 [16] などが広く用いられている。ここでは、本ツールをベースに、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 10 のようにして表す。ここで、$記号が先頭に現れる記法はパターン変数を表している。図 10 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する。

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、nil を返す代わりにパターンのリスト<list>の各要素<var>に Lisp 式<iteration>を適用して得られたリストを返す。

これらを用いると、例えば図 11 で書かれた XSLT プログラムと同等なプログラムは図 12 のようにして書くことができる。変換前と変換後に同じようなパターンを使っているため、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。

7. 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 [18]。本ツール同様、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。本ツールと違うところは、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 [9] 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、本ツール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、再コンパイルが必要である。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、再コンパイルせずに、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考えられ [8, 2]、本ツールも 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。

本ツール同様、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]。これは本ツール同様、XML 文書を S 式に変換して扱うものであり、S 式の文法も本ツール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. のツールは妥当性を検証しない XML 構文解析系であり、S 式ベースで妥当性の検証を行ったところに本ツールの特色がある。

8. まとめ

本テーマでは、DOM の代案として Lisp ベースの XML プログラミングツールを提案し、実装した。これにより、複雑な DOM インターフェイスを覚えることなく、開発者が培ってきたプログラミング技術をそのまま XML アプリケーション開発に応用することが可能になった。またインターフェイスではなく、Lisp 言語の基本的構成要素によって XML 文書の内部表現にアクセスすることによる柔軟性も得られた。さらにデータを直

```
<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 11: 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 12: 本ツールでの XML 構造変換プログラムの例

接記述することができる等の Lisp の利点を用いることも可能になった。また、S 式ベースの XML プログラミングツールにおける妥当性の検証も可能になった。そして本ツールを基にした XML 関連技術の開発を通じて、リレーショナルデータベースから XML 文書へのマッピングが実際にリスト処理によって非常に簡単に実現できることが検証され、さらに本ツールを基にした XML 文書構造変換言語が、従来技術の XSLT と比べて簡潔で理解しやすいものであることが示された。

今後は、本ツールを例えば RELAX [1] 対応にするなどの機能拡張をしていく必要がある。

9. 謝辞

今回の事業において、湯淺太一氏にプロジェクトマネージャをしていただきました。プロジェクト管理に関しては、日本エンジェルス・インベストメント株式会社よりご協力をいただきました。また、私の指導教官である玉井哲雄氏をはじめ、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いただきました。それぞれに感謝します。

10. 参加企業及び機関

本開発は、紙名哲生個人で行ったものである。

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

[5] Haruo Hosoya and Benjamin Pierce. XDuce: A Typed XML Processing Language. In *Proceedings of Third International Workshop on the Web and Databases (WebDB2000)*, 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] Erik Meijer and Mark Shields. XMLambda: A Functional Language for Constructing and Manipulating XML Documents. In *USENIX Annual Technical Conference*, 2000.

[10] W3C Web Site. Scalable Vector Graphics (SVG) 1.0 Specification. http://www.w3.org/TR/SVG/.

[11] W3C Web Site. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/SOAP/.

[12] W3C Web Site. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl.

[13] W3C Web Site. XML representation of a relatinal database. http://www.w3.org/XML/RDB.html.

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

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

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

[17] Tetsuo Kamina, Taiichi Yuasa and Tetsuo Tamai. A Light-Weight Programming Interface for XML. In *Proceedings of The Fourth Workshop on Internet Technology (WIT2001)*, 2001.

[18] Malcolm Wallace and Colin Runciman. Haskell and XML: Generic Combinators or Type-Based Translation? In *Proceedings of the International Conference on Functional Programming*, 1999.

[19] 紙名哲生, 玉井哲雄. Lisp を基にした軽量で柔軟な XML プログラミングツール. In **情報学シンポジウム** *2002*, to appear.

A. XML 文書の S 式表現のための文法

本ツール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 [14] の中で定義されている空白と同じである。ListCommand 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。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。

B. 本ツールの動作環境

本ツールは、以下のプラットフォーム上で動作する:

- Common Lisp 処理系
  - Allegro Common Lisp 6.0 以上
  - CMU Common Lisp 18b 以上

ただし、CMU Common Lisp 上では、処理系自体が Unicode に対応していない等の制約がある。

C. 本ツールの API

(1) XML 構文解析系

XML 構文解析系のインターフェイスは以下の通りである:

```
(parse-xml stream &key validate)
```

stream は XML 文書が送られる入力ストリームである。validate が t にセットされたとき、parse-xml は妥当性を検証する。

(2) XML 文書自動生成

XML 文書自動生成のプログラミングインターフェイスは以下の通りである:

```
(generate-xml-with-stream dest doctype
 s-expr &key validate public system)

(generate-xml doctype s-expr
 &key validate public system)
```

これら二つの関数は、s-expr から受け取った S 式から XML 文書を生成する。generate-xml-with-stream 関数は、dest 引数で出力ストリームを指定することができる。validate が t にセットされたとき、妥当性を検証する。public、system で、それぞれ出力される XML 文書の公開識別子、システム識別子を指定することができる。

generate-xml 関数は、生成された XML 文書を直接 Web サーバ [8] に渡す。

(3) XML 要素サーバへの登録・削除

XML 要素サーバへの登録・削除を行うプログラミングインターフェイスは、以下の通りである:

```
(parse-dtd-only stream doctype)

(remove-doctype doctype)
```

parse-dtd-only 関数は、stream 入力ストリームに送られた DTD ファイルを構文解析して、XML 要素サーバに登録する。doctype で文書型名を指定する。remove-doctype 関数は、doctype で指定した文書型名を持つ文書型を、XML 要素サーバから削除する。

(4) RDB から XML 文書への変換

これには、以下の関数が存在する:

```
(register-dtd-from-db data-source &key db)

(sqltoxml query data-source &key db)

(publish-dtd data-source)
```

register-dtd-from-db 関数は、RDB のスキーマ情報から文書型を自動生成し、XML 要素サーバに登録する。db でデータベースへの接続インスタンスを指定し[3]、data-source でデータベースのデータソース名を指定する。

sqltoxml 関数は、query で受け取った SQL 分をデータベースに問い合わせ、結果を S 式に変換して返す。

publish-dtd 関数は、データソース名にdata-source を持つデータベースから自動生成され、XML 要素サーバに登録されてある文書型を、文字列にして返す。

(5) XML 構造変換言語

以下の 3 つの文法が、マクロで定義されている:

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

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

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

defrule 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。

rule-set 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。

for-each マクロは、Common Lisp のdolist に似ているが、nil を返す代わりにパターンのリスト<list>の各要素に<iteration>を適用して得られたリストを返す。

[3]Allegro ODBC [7] 参照