SGMLインスタンスの変換方式の検討

4W−2

今郷 詔, 西村美苗

(株)リコー 情報通信研究所

1 はじめに

紙に印刷するためだけにコンピュータで文書を作る場合、コンピュータ上での文書の表現は重要ではない。しかし、文書のデータベース化やオンラインでの交換を行なう場合には、様々なアプリケーションで利用可能な表現にする必要がある。

この目的に最適なのが、SGML (ISO 8879) である。SGMLは文書を属性を伴ったelementの木構造として表現でき、文書の論理構造などを表現することができる。

SGML文書は幅広いアプリケーションで利用可能であるが、アプリケーション依存形式にどのようにして変換するかが問題となる。

2 必要な変換機能

変換先のデータ構造が変換元のSGML文書に似ている場合には、タグの写像のような単純な変換だけで済むことも多い。しかし一般的な変換を行なうには少なくとも次のような変換が可能でなければならない。

- 属性値とelementとを変換する。
- elementを削除する。
- 属性値あるいはelement名によって処理を変える。
- 同一レベルでelementの順序を変更する。
- elementを任意の位置に移動する。
- 省略されたelementに対してデータを生成する。

3 これまでの変換方法

SGML文書は、文書の構造定義に当たるDTD (Document Type Definition) と、実際の文書内容であるインスタンスとからなる。変換指定を行なう対象がDTDかインスタンスかによってこれまでの変換方法を二種類に分けることができる。

インスタンス対象 それぞれのelementに対して、先頭および末尾で実行すべきコードや出力すべき文字列を指定する。例えばMARKUP[1]というシステムでは、図1のように指定する。この例では、インスタンスの中で'TITLE'という名前のelementが出現すると、まず"\title{"という文字列を出力し、次にelementの内容をすべて出力した後、"}"という文字列を出力することを意味する。

```
<ELEMENT TITLE>
  <START-STRING>\title{</START-STRING>
  <END-STRING>}
</END-STRING>
```

図1: インスタンス対象変換指定の例

```
<!element chapter 0 0
  (heading
   , { /* processing before pp-sequence */ }
   pp( /* parameters */ )
     { /* processing for each pp */ }
   * { /* processing after pp-sequence */ }
   , { /* processing before appendix */ }
   appendix)>
```

図2: DTD対象の変換指定の例

DTD対象 DTDに含まれているelement宣言のcontent modelの中に変換処理を埋め込む。Amsterdam SGML Parser[2]では、それぞれのelement宣言を一つの関数定義のようにみなし、図2のようにCで変換処理を記述する。LL(1)パーサがインスタンスを解析をする時に、適用した文法に対応するC関数を実行することになる。

インスタンス対象方式は記述が単純なために、変換先のデータの構造が変換元の文書と似ている場合には適するが、次のような欠点がある。

- 省略された要素に対してdefault処理ができない。
- 二つのelementの順序の入れ替えのように構造を変更する場合の変換指定が簡単ではない。
- 同名であっても出現環境(親子関係、前後関係など)によって意味が異なるelementに対する処理を記述することが難しい。

したがって、複雑な変換を行なう場合には、DTDを対象として変換規則を記述するべきである。

Data conversion from SGML instances.
IMAGO Satosi and NISHIMURA Mina
RICOH Co., Ltd.

しかし Amsterdam SGML Parser 方式では、生成規則に C 関数を結び付けているだけなので、次の欠点がある。

- element の出現環境を明示的に渡す必要がある。
- 戻り値が一つしか利用できないため、二つ以上の情報を戻す場合は、別の仕掛けを使う必要がある。
- DTD そのものに変換規則を書き込むので記述したデータが理解しづらい。

4 属性文法による変換指定

"Chameleon" というデータ変換系では、element の順序の置換に合成属性だけを用いた属性文法を利用している [3]。この考え方を発展させて、属性文法を利用して SGML 文書の変換規則すべてを記述し、上で示した問題を解決することを検討している。

属性文法は構文規則 (CFG) と意味規則とからなる。CFG のそれぞれの記号には属性が付随している。属性は次のように合成属性と相続属性との二通りに分れ、生成規則における親の合成属性と子の相続属性の値を意味規則で定める [4]。

合成属性 木の下から上へ値が決まり、構文規則の左辺の非終端記号に対してのみ定義できる。

相続属性 木の上から下へ、あるいは兄弟間で値が決まり、構文規則の右辺の非終端記号に対してのみ定義できる。

SGML 文書は element を SGML 属性付きノードとする木構造と捉えることができる。変換結果とその他の中間データを合成属性として与えれば、根の element の合成属性として最終的な変換結果を得ることができる。

一方、個々の element の出現環境を相続属性として渡すことができる。この環境の多くは属性評価器側で用意することができ、明示的にユーザが指定する必要はない。出現環境には次のようなものがある。

- element 名、祖先 element 名
- 前方の兄弟 element 名
- SGML 属性値
- 繰り返しの回数

SGML の element 同士の関係は LL(1) 文法で記述できるので、1 パスで属性評価が可能な L 属性文法が適する。L 属性文法の制約内で実用的な変換規則を記述できると考えている。

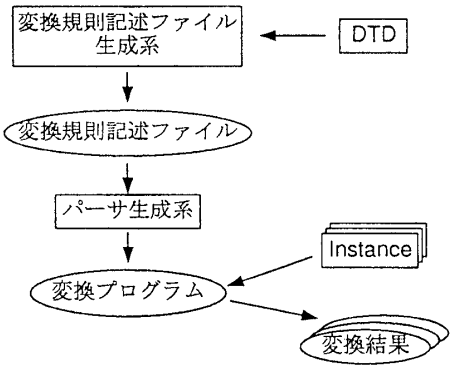

図 3: 変換手順

属性文法を用いた変換プログラムの作成とそれを使った変換の手順を図 3に示す。まず、DTD から変換規則を記述するためのファイルを自動的に生成する。このファイルに対して必要な意味規則を書き込んだ後、そのファイルを元に DTD 固有の変換プログラムを生成する。変換プログラムを一度作成しておけば、インスタンスを与えるだけで変換を行なうことができる。

5 おわりに

SGML 文書を任意のデータに変換する方法として、属性文法を利用した方法を検討している。この方法には次の利点がある。

- 省略された要素に対して default 処理が行なえる。
- 変換結果以外に、中間データを合成属性として渡すことで、大域変数が不要になる。
- 個々の要素の出現環境を相続属性として渡すことで、環境による処理の切り替えが容易になる。
- 属性評価器側で相続属性の設定やコピー規則を補うことにより、簡潔な変換仕様記述ができる。

今後は、理解しやすい変換規則記述方法を決定した上で、実装・評価を行なう予定である。

参考文献

[1] L. A. Price and J. Schneider. Evolution of an SGML application generator. In *ACM Conf. Document Processing Systems*, pages 51–60, 1988.

[2] J. Warmer and H. van Vliet. Processing SGML documents. *Electronic Publishing*, 4(1):3–26, 1991.

[3] S. A. Mamrak, M. J. Kaelbling, C. K. Nicholas, and M. Share. Chameleon: A system for solving the data-translation problem. *IEEE Trans. Software Eng.*, 15(9):1090–1108, 1989.

[4] 佐々政孝. 属性文法. コンピュータソフトウェア, 3(4):73–91, 1986.