TEI-P3 について

豊島正之 / mtoyo@lit.hokudai.ac.jp

これは、情報処理語学文学研究会 (JALLC) 会報 15 号 (1994 年 7 月) に掲載し、その後、情報処理語学文学研究会会報累積版 (1996 年 7 月) の pp.227–236 に整形掲載されたものを、新たに整形して掲げるものです。再整形以外は、(著者の所属を含め)「累積版」に一切変更を加えていません。特に、TEI-P3 の配付元等は、現在は有効ではないので、御注意下さい。　　2000 年 8 月

TEI-P3 がようやく[1]公刊された。本稿は、この TEI-P3 を簡単に概観し、問題点を指摘する。

以下 TEI-P3 は、「25.8.3,p.695」(chapter 25, section 8.3, 印刷公開版 p.695) の様に引用する。印刷版のページ数は、配付母体によって異なっているかも知れないので、注意。

1　TEI と TEI-P3

TEI(Text Encoding Initiative) は、言語データをテキストとして流通する為の共通交換仕様 (encoding format) の提唱である。

TEI は、ACH(The association for computers and the humanities)、ACL(The association for computational linguistics)、ALLC(The association for literary and linguistic computing) が運営委員会 (steering committee) を組織した大規模なプロジェクトで、その成果は、TEI-P1 (TEI Public draft 1, 1990)、TEI-P2(1992-、未完のまま放棄)、TEI-P3(1994) として公表された。

1.1　入手方法

TEI-P1(本篇 275 ページ) は、現在では入手困難。

TEI-P2 は、結局印刷物としては公表されていない。

TEI-P3 は、

〒 263 千葉市稲毛区弥生町 1-33 千葉大学文学部 土屋俊様方
TEI 日本委員会

から入手出来る。(上下 2 冊、1289 ページの大冊である)。

ファイル (機械可読形式) の入手先は、いずれも internet で

[1] **[累積版補注]** 本稿は、1994 年 7 月稿。

1. anonymous FTP `sgml1.ox.ac.uk`、`ftp.ifi.uio.no` (この他に未公開国内サーバあり)
2. 自動応答メール `Listserv@uicvm.uic.edu (filelist tei)`
3. (累積版追加)
   現在 (1996 年 5 月) では、下記の www サイトに情報がある。

- `http://www.ceth.rutgers.edu/projects/hercproj/front.htm`
- `http://info.ox.ac.uk/ archive/teij31/`

但し、全部で約 70 ファイル、3.7MB もあるので、それなりの覚悟が必要である。(圧縮すれば 2DD 一枚に何とか納まる)。

TEI-P3 は、このファイルを適宜 grep しながら、印刷本をランダムアクセスして読んで行くのが一番具合がいい様である[2] 。

1.2 適用範囲

TEI プロジェクトの開始当時は、文字文献データの流通を目指したものだったが、TEI-P1(1990)、TEI-P2(1993)、TEI-P3(1994) と進むにつれ、書誌記述、会話、グラフ等にも適用範囲 (scope) を広げ、一般に「言語データのテキストとしての encoding」形式に至った。(TEI-P1 → TEI-P3 で五倍以上にふくれ上がったのはこの為)。

2 TEI-P3 概観

2.1 モデル

TEI は、言語データを SGML(ISO 8879/JIS X4151) に基づいてテキストとして流通させる為の、文書諸要素 (element) の定義集である。

具体的には、あるデータをテキスト化 (encoding) するものは、そのジャンルに相応しい TEI の DTD(document type definition、即ち文書の構造のみに就ての抽象的記述) を選び取り、それを参照 (include) して、文書記述をして行く事になる。つまり、TEI は、テキスト化の為には、どういう element を、どういう意味・用法で用いて行くかにに就ての、合意の集合である。

2.2 オプションの選択機構

TEI は、あらゆる局面に備えるべく、なるべく多くの element を用意しようとした様であり、element の説明付き一覧だけで実に 400 頁にも達するのであるが、それでも実際には、ユーザ (テキスト作成者) がこれらを基礎として更に独自の DTD を立てる事が必要になるのではないかと思われ、その場合、後述の様な様々な問題が生ずる事になる。

用意された一般ジャンルには、著者・書名・底本、或は版式・版次等の書誌記述、校異、引証、韻文、押韻表示などが挙げられ、より特殊なジャンルとしては戯曲、自然会話、辞書、データベース、原本翻刻 (欠損、数次訂正等を含む)、相互参照、グラフ、表、図、等が用意されている。(今の所、楽譜・地図は無い)。これらの中には、排他的 (あるものを選んだら、他は選べない) なものもあるので、その選択がなるべく自動的に行なわれる事が望ましい。

そこで、こうした豊富な TEI の DTD からなるべく自動的に必要な定義 (DTD の一部) を選び出す為に、いくつかのマクロ (parameter ENTITY) がスイッチになっており、それらを ON/OFF する事で後は全て自動的に組み込まれるという、極めて巧妙な仕掛けが使われている。

[2] **[累積版補注]** 本稿後、*Dyna*Text 版が発売された。
Sperberg-McQueen,Michael & Burnard,Lou (1994) Guidelines for electronic text encoding and interchange (TEI P3) / Electronic Book Technologies,Inc(Providence)/ phone +1-401-421-9550, Fax +1-401-421-9551 / ISBN `1-886034-00-1`

しかしその巧妙な分、DTD を直に読んでも殆ど理解不能に近い代物になっているのも事実である。プログラム言語ならまだしも、TEI は人間が読む為の物である (何故なら、後述する様に 3.2 (p. 4) 少なくとも現状では、TEI 文書の本来の意味での機械処理は困難であるから) ので、これは困った事である。

3 SGML に起因する問題

3.1 attribute に構造が無い

これは、後述の 3.4 (p. 5) スコープと並んで、SGML 最大の問題である。

SGML は、意味論を全く持たない珍しい言語で、メタ言語とも呼ばれる。意味付与 (意味論記述) は全てユーザの責任で、具体的には、「この element は、これこれこういう意味で使おうね」という合意をした上で SGML 文書交換を行なう事になる。TEI は、正にそうした意味付与の一例である。(この他に `HyTime` 等も SGML に対する意味付与合意規格である)。

僅かにユーザに与えられた SGML 内での意味付与用のラベルが attribute で、

```
<!ATTLIST minutes status (draft | proposed | rectified) proposed>
<!ATTLIST 人物 属柄 (父 | 母 | 子)  性別 (男 | 女) ...>
```

の様に DTD で定義して置き、

```
<minuites status=proposed> The minuites of the 16th conference... </minuites>
<人物 属柄=母 性別=女 年齢=52 歳>鈴木花子... </人物>
```

の様に element にラベルを与えるのに使う。ラベルは、予め指定されたものしか使えないので、文書標準化に役立つ、という仕組である。(勿論ラベル自体が直接に意味を与える訳ではなく、意味付与に役立つ、というだけの事だが)。

問題は、attribute が単なるラベルで、その中に更に構造を作る事が不可能な点である。例えば、

```
<minuites status=proposed(version=4 superseds=2)> ... </minuites>
<人物 属柄=太郎の母>鈴木花子...</人物>
```

の様に、attribute 自体を更に構造化したい事はよくあるが、こうした事は一切出来ない。この為

```
<minuites status=proposed version=4 superseds=2> ... </minuites>
<minuites status=draft version=4> ... </minuites>
<人物 属柄=母 子=太郎>鈴木花子...</人物>
```

の様に、バラバラの attribute を立てる以外に無い。上の二つの例では、`version` が `minuites` という element 自体の version の様に見え、`proposed` / `draft` の内部 version であるという意味が全く伝わらないばかりか、`version` に関しては上と下の element は同一であるという誤った情報を伝える事になる。

この様に、SGML には attribute の構造を書く方法が無い為、attribute の相互依存関係 (hierarchy の一種) も書けない。例えば、

1. 「刷=初刷」という attribute を振る為には、「版本」attribute を振って置かねば成らないとか、
2. 「活用=ハ行四段」という Attribute があったら「品詞=動詞」Attribute が仮定されるとか
3. color=red という attribute を振りながら、display=monochrome であってはいけないとか

といった、attribute 相互の持つ依存関係 (構造) は、一切記述する事が出来ない。「属柄=父」とあれば当然「性別=男」である、といった推論を行なう為の機構は、SGML には存在しない。

この様に、attribute の構造を管理する方法が SGML に欠けている為に、TEI は、極力それを element として立てる方針を取ったが、この様な実質的に attribute である空の element (EMPTY elements) の乱立は、文書の構造を不明確にし、descriptive markup の哲学に矛盾する。

3.2 attribute のチェックが無い

上記の様に、attribute の構造管理の方法が無い為、矛盾した attribute もそのまま通って仕舞う。

```
<人物 属柄=父 性別=女 年齢=2 歳>
```

等というのも平気で通る。

従って、SGML 文書には、SGML パーザだけではなく、attribute のチェックの為の validator が必須であり、TEI-P3 中には、そこかしこにこの validator に頼った記述がある。(eg. 16.4/p.484、26.3/p.706)。しかし、汎用 validator の実装例は、未だに[3]一つも知られていない。汎用 validator の為には、attribute の意味仕様記述が可能な言語が必要な筈であり、SGML は明らかにそれを満たさないから、SGML だけに依存している TEI-P3 の範囲で validator が実現出来なくても不思議は無いとも言えるが[4]。

つまり、矛盾の有無が指摘できるのは人間だけであり、今の所

TEI 文書は、人間が読んでチェックするもの

である。

3.3 element の再帰

element の定義が自分自身を指すの事は勿論可能である。例えば、「文字列とは任意数の文字と任意数の文字列をつなげたもの」($string = character* + string*$) という定義は、

```
<!ELEMENT str - - (char*, str*) >
```

と書ける。

この手の「再帰」(recursive) で危険なのは止らなくなるケースで、

```
<!ELEMENT str - - (str*) >
```

等とすると、永遠に str の中身に行き着けない。

SGML としては、これらの永遠再帰定義は十分に文法的で、何の問題も無い。実用に耐えないだけである。(止らないプログラムと同じ)。という事は、「止らない」 DTD を書いても、何のエラーメッセージも出ない、つまり処理系に指摘して貰えないという事である。

これが自分で定義する element ならばまだ問題は少ないが、TEI の定義する element は極めて複雑で、どれがどれを呼んでいるか一見して分る等という甘い代物ではない。TEI-P3 の element 説明集の相互依存関係の記述は不十分で、相互依存関係一覧の為には、プログラムが必要な程である。せめて、element や attribute の名前の付け方 (naming convention) にもう少し工夫をして置けば、混乱は少なかっただろうに、実際には、`<language>`、`<usg>`、`<val>`が element で `lang`、`usage`、`value` が attribute、`<when>`は element、`who` は attribute、`<class>`と`<classes>`は別の element 等と、名前付けに方針など無いに等しく、混乱は著しい。このような状況では、TEI の DTD を逐一たどって熟知して置かない限り、うっかりすると自分自身を間接的に (永遠に) 呼出し続けるような文書定義を書き兼ねない。つまり、端的に言えば、

TEI の DTD はブラックボックスに成れない

のである。

3.4 スコープが無い

[3] **[累積版補注]** 本稿の時点 (1994 年 7 月) での記述であるが、現在 (1996 年 5 月) に至っても同じである。

[4] **[累積版補注]** SGML では、相異なる DTD に従う文書の結合が不可能である為、TEI に限らず、DTD の標準化が国際的に進められている。日本でも、日本工業規格 (JIS) 案として「分散形情報共有環境における文書記述用の基準文書型定義」(通称 G-DTD) が提案され、1995 年に公開レビューが行なわれた。G-DTD は、TEI 同様、パラメタ ENTITY の多用による条件付き inclusion によって共通 DTD をカスタマイズして用いる様になっている為、SGML 内部での検証は不可能で、検証は、やはり外部の validator 任せである。但し、適用分野が実工業界の為、商用 validator が続々登場するという見通しを持てる点が TEI と異なる (規格関係者からの通信) との事である。

信じられない様な事だが、SGML の element は全て global 定義であり、名前のスコープというものは存在しない。(かろうじて attribute だけは element をスコープに持つ)。

つまり、ある element を hierarchy の下の方で定義しても、その element は文書全体から参照可能である。逆にいうと、DTD/文書中の element 定義は、必ず別の名前にしなければ成らない。つまり、TEI の様にお仕着せ DTD を配付して、且つそこにユーザ定義の構造を加えて行こうという場合は、全ての定義済 element が、「システム予約語」扱いになると言う事である。

これが、TEI の利用に対する大きな障碍である事は容易に分る。巨大な TEI headers を参照したら、どこで定義してあるか分らない element とぶつからない様に、慎重に名前を選ばないと大変な事になる。一番怖いのは、うっかりタイプミスなどをして、自分で定義した element の代りに、偶然 TEI headers に定義済の element を参照して仕舞う事である。何のエラーも無しに通るから、全く誤った文書構造記述をしていても気付かない。

名前スコープがないと言う事は、こうしたシステム定義とユーザ定義名前の衝突だけではなく、同一名称を対象の実質に応じて適宜置き換えて使わせる overloading が不可能であるという事である。これは、文書のオブジェクト指向化には深刻な問題で、抽象的な element を順次詳細化して行って、上位の記述を受け継ぎながら (inheritance)、細部はそれぞれ独特の修正を加えて (overloading)、具体的なオブジェクトにたどり着く、といった自然な記述が不可能になる。

TEI は、こうしたオブジェクト指向化を element class という概念を立てて行なっているが、実はそこで行なっている inheritance は、上位の attribute 記述をマクロ化してそれを丸ごと include するか、或は手作業で (!) コピーする (3.7.1/p.59) というものである。これでは、上から下まで熟知しているエキスパート[5] 以外には、安全な TEI 文書を書くのは至難の技である。

3.5 文書単位がファイルである

SGML 文書が他の SGML 文書を含む (include) 事は困難である。(SGML 自体は SUBDOC という機構を用意しているので、全く不可能とは言えないが、SUBDOC の実装自体困難の様であるし、そもそも、TEI は SUBDOC は使っていない)。もっとも、上記のスコープの問題があるので、例え include の機構があったとしても、運用は極めて困難だろうが。

文書の include が出来ないという事は、小さな文書を集めて大きな文書を作る事自体が出来ないという事である。これが文献 corpus に取ってどれ程深刻な問題かは、議論する迄も無い。

4 文字コードの問題

4.1 Writing system declaration

TEI-P1 では、文字は実質的に ASCII しか使えなかったが、さすがにそれではあんまりだという事で、TEI-P2 から WSD(Writing system declaration) が文書中の文字コード注釈を行なう事になった。これは、文書中に使われる字の一覧表を文書の最初に提示しようという考えで、校本万葉集の「校異を出さざる異体字の例」の発想と似ている。

ところが、(ここが校本万葉集と違うのだが) TEI 文書はネットワークなどで転送されるので、当然その過程で文字コード変換が掛かり、結果的に文書内にあるコード注釈と実際の文書が矛盾するという事態が起こり得る。

4.2 WSD のクレタのパラドクス

[5] **[累積版補注]** TEI-P3 を上から下まで熟知しているのは、世界中で Lou Bernard と Michael Sperberg-McQueen (共に TEI-P3 の編者) の二人しかいない、という冗談がある。しかし、三人目の名前を挙げようと考えた途端に、これが全く冗談になっていないことに気付く。大抵のプログラム言語では、あり得ない事である。

WSD は、自分自身のコード系に就て、(そのコード系を用いて) 記述するという自己参照を行なっているので、一たびコード変換が行なわれると、クレタのパラドクス同然の事態になる。

具体的に言えば、ある特定コード系に基づく特定字を定義するには、WSD の<character> 中の <form> element で codedCharSet という attribute に値を与える。(25.4.2,p.686)。ここで

```
<!ATTLIST form %a.global;
    string          CDATA   #IMPLIED
    codedCharSet    IDREF   #IMPLIED
    ...
>
```

と定義されているので、具体的には

```
<form string='ナ'  codedCharSet=JISX0201>
<form string='Å'  codedCharSet=ISO8859-LATIN-1>
```

(いずれも□内には、実際の文書では 0xc5 相当の符号が入る)

等とするのであろうか。(TEI-P3 には、具体例無し)。

奇妙な事だが、WSD 自体は ISO2022 の code extension technique (cf. 豊島稿「コードのはなし」/情報処理語学文学研究会会報第 9 号/1991.7) によって読まれるべきであると規定されている (25.8.3,p.695)。従って、string='C' のところで codeset のスイッチが行われる筈だが、もしこれを予期するなら、そもそも codedCharSet の attribute は不要の筈である。一方、Shift-JIS → EUC の様な変換が行なわれてファイルが届くと、この辺りの記述は軒並ファイル実態と矛盾して仕舞う。

或は、この部分は TEI-P3 の誤植で、正しくは

```
<!ATTLIST form %a.global;
    string          RCDATA  #IMPLIED  <!-- not CDATA -->
    codedCharSet    IDREF   #IMPLIED
    ...
>
```

であるべきなのかも知れない。RCDATA(reference character data) は、実質的にコード値の dump なので、

```
<form string='&#197'  codedCharSet=JISX0201> <!-- 197 == 0xc5 -->
```

の様に書かれる筈である。しかし、TEI-P3 の関連箇所には、何回も「byte string」と出るので、誤植とも思えず、この辺り、TEI の意図が何なのか掴み切れない。

4.3 SGML のクレタのパラドクス

実は上記は (も) SGML そのものが抱える問題である。

SGML は、SGML declaration によって、やはり自分自身の記述コード系を記述するというクレタのパラドクスを犯している。[6]

この問題は、1992 夏以降 comp.text.sgml で盛んに議論されたが、結局、現行 SGML 規格内での解決が見いだされなかった。

5 文献

【本節は累積版に当たっての追加である】

現在、SGML を学ぶために最初に読む本として推奨できるのは Softquad(1990) で、僅か 36 ページながら、大変分かりやすいパンフレットである。現在は、http://www.sq.com/ から見られる様になった。

日本規格協会の「実践 SGML」の原本である Herwijnen の本は、改訂第 2 版 Herwijnen(1994) が出て、くどいのが難点だが、初版 (1990) に比するとかなりよい教科書に仕上がっている。

しかし、SGML に就て語ろうと言うのであれば、Goldfarb(1990) が必読・必携である事は変らない。

[6] **[累積版補注]** 豊島「TEI から見た SGML のはなし」(本累積版収録) 参照。

この他に吉岡誠 (1993)、麻布メディア研究会 (1994)、根岸・石塚 (1994)、日本ユニテック SGML サロン (1995) 等もあるが、Goldfarb の代りにはならない。尚、Bryan(1988)「SGML 入門」は絶対に避けるべき悪書である。

1. Bryan,Martin 1988 SGML 入門 (アスキー)
2. Goldfarb,Charles 1990 The SGML handbook(Oxford UP)
3. Herwijnen,Eric van 1994 Practical SGML (2nd ed.) (Kluwer Academic)
4. JIS X4151 1992 文書記述言語 SGML(ISO 8879 の翻訳規格)(日本規格協会)
5. Softquad 1990 The SGML primer(SoftQuad, FAX +1-416-239-7105, mail@sq.com)
6. 麻布メディア研究会 1994 SGML のかきかた (麻布プロデュース, FAX 03-5570-9774)
7. 日本ユニテック SGML サロン 1995 はじめての SGML(技術評論社)
8. 根岸正光・石塚英弘 1994 SGML の活用 (オーム社)
9. 吉岡誠, 他 1993 SGML のススメ (オーム社)

6 その他

TEI プロジェクトは、偉大な試みであり、とにかく生産物 (product) を出したというだけでも、賞讃さるべきである。本稿は批判の言辞に終始して仕舞ったが、これはあくまで今後の更なる発展を希っての事と理解して戴ければ幸いである。

【謝辞】
TEI-P3 を頒布して下さった Michael Sperberg-McQueen 氏とそのアシスタント、更に、日本での TEI 普及に尽力され種々御教示下さった土屋俊氏に御礼申上げます。