半構造化文書のための一貫性制約管理機構の設計と実装

小　嶋　卓　也
大　本　英　徹

第1章　序　　論

グローバルネットワーク（Web）全体が，一つの巨大な情報システムあるいはデータベースとなっていくとき，ここに参加するアプリケーションの間で共通な情報表現のフォーマットが必要となってくる．この情報表現フォーマットとしてXML（eXtensible Markup Lnagage）の適用が注目されている．

XMLはアプリケーションの取り扱うビジネスデータを格納する規格であり，あらゆる種類の情報に合わせたさまざまな設定が可能な革新的でオープンな標準として，情報処理業界の専門家から一般のエンドユーザまで浸透しつつある．XMLがここまで浸透するようになった理由として，以下に示すXMLの特徴が挙げられる．

・XMLは，様々な種類の情報をニーズに合わせて柔軟に表現して格納できる．
・オープン標準であり，特定の企業の利害関係や特定のアプリケーションに依存することはない．
・文字集合にUnicodeを採用していることで，漢字を含む多くの文字と記号をサポートしている．
・XMLは，それが表現する電子文書を記述するための，シンタックス，内部リンク検証，文書モデルとの比較，データ型といった手段を提供しており，かつ，それを自動処理することが容易となるように設計されている．
・明解で単純なシンタックスとあいまいさのない構造によって，人間も機械も容易にXMLを読み取り・解釈することができる．
・XMLとスタイルシートによって，どのような見栄えの文書にも整形できる．純粋な形で情報の構造を持つことで書式の変更のためだけの文書変換は必要でなくなる．

XMLの一般的な利用形態のひとつは，相異なるアプリケーション間のデータ交換手段としての利用である．その際には，データの解釈において一意に定まるようにXML文書を作成する必要がある．XMLでは解釈の曖昧さが生じないようにに記述する方法を提供している．これは文書のモデル化とよばれている．最も一般的な文書のモデル化の方法は，文書型定義（DTD: Document Type Definition）を使うやり方である．DTDは，ルール，あるいは宣言の集まりで，どのタグが利用可能でどのタグを包含可能かを規定する．
DTDは文書のモデル化，つまり文書の構造に関する制約であるが，本論文では，DTDによる制約だけでは不十分で上手く表現できない制約が存在することについて述べ，それを解決するためのシステムの試作・開発について述べる．

第2章ではXMLとXPathの概略について述べる．第3章では本論文で提案するXML文書の一貫性制約に関して述べ，第4章でシステムの設計と実装，第5章でシステムの実行例を示す．第6章はまとめである．

第2章　XMLとXPath

2.1　XML概略

2.1.1　XMLの生い立ち

SGML

異なる環境のコンピュータがネットワークで接続される中，ドキュメントの標準化と共通利用へのニーズが高まり，ドキュメントの汎用フォーマットとして，まずSGML（Standard Generalized Markup Language）が提案された．
SGMLは代表的なマークアップ言語の１つで，文書の構造を記述することに重点が置かれている．これにより，文書の処理や管理，コンピュータ間でのデータ交換などが容易に行なえるようになる．
しかし，SGMLは「仕様が複雑過ぎて専門家でないと扱えない」「インターネット上でそのまま使用できない」などの理由で十分普及していない．

HTML

World Wide Webで採用されているHTML（Hyper Text Markup Language）は，SGMLを参考にして作られている．そのため見た目は似ているが，SGMLに比べてシンプルな形で習得しやすいのが特徴である．
HTMLならすべてのブラウザで同じように表示でき，全世界で共通に利用できる巨大な情報ネットワークの基盤となった．

SGMLとHTMLの融合

HTMLはもともと，文書の見出しやタイトルなど，情報の表示方法を規定するだけで，その内容や構造までを明確に規定する文法ではない．したがって，Web上の情報をデータベースのように活用するには限界がある．

そこでXMLが誕生した．XMLの仕様の大部分はSGMLから受け継ぎ，HTMLからは文法のシンプルさと，今日のインターネット環境とをそのまま受け継いでいる．

2.1.2 XMLの設計目標

XMLの設計目標は以下の通りである．

・XMLはインターネット上でそのまま使用できること．
・XMLは広範囲のアプリケーションを支援すること．
・XMLはSGMLと互換性を持つこと．
・XML文書を処理するプログラムを書くことは容易であること．
・XML文書は人間が読めて，十分に理解できるものであること．
・XMLの設計は速やかに行われること．
・XMLデザインは厳密かつ簡潔であること．
・XML文書は容易に作成できること．
・XMLではマーク付けの数を減らすことは重要ではない．

2.1.3 XMLの文書構造

XML文書は，XML宣言，DTD，XMLデータ（XML文書本体）の3つの部分から成る．DTDと

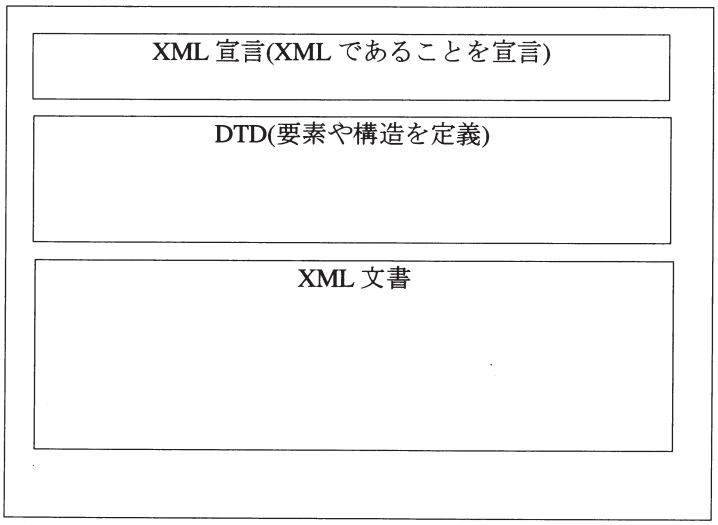


図2.1：XML文書の構成

いうのは要素や構造の定義をするもので，DTDがなくてもXML文書の生成や利用はできる．DTDをもたないXML文書を整形式XML文書（well-formed文書，DTDを持つXML文書は検証済みXML文書（valid XML）と呼ばれる．

XML宣言

XMLのバージョン，文字コードの宣言をする．

・バージョン

W3Cで決められたXMLのバージョンをversionという属性で指定する．現在は1.0を使用する．次のように記述する．

```
<?xml version=”1.0” ?>
```

・文字コード

encodingという属性で文字コードを指定する．

例えば，シフトJISの場合は，

```
<?xml version=”1.0” encoding=”Shift-JIS” ?>
```

・EUCの場合は

```
<?xml version=”1.0” encoding=”EUC-JP” ?>
```

と記述する．

ここでxml versionとencodingは小文字で記述しなければならない．

XML文書本体

XMLの基本単位は要素（エレメント）である．開始タグと対になる終了タグをつけたものである．開始タグと終了タグの名前は要素名という．

```
<要素名> 要素の内容 </要素名>
```

・ルート要素

XMLにはルートと呼ばれる要素があり，そのXML文書の全ての要素をまとめるタグ．ルートタグは文書に１つだけ必ず存在する必要がある．

・階層構造の記述

兄弟関係をあらわすときは，要素を並列に並べて記述する．

親子関係は要素の中に要素を持つこと．要素を入れ子に記述する．

```
<root>
  <oya>
    <AAA>AAA</AAA>
      <BBB>BBB</BBB>
  </oya>
</root>
```

AAA と BBB は兄弟。
oya は AAA と BBB の親。

・属性

属性とは要素に付随する情報のことで，要素名を変えずに種類（タイプ）だけを切り分けたいとき，同じ要素名でもアプリケーションでの対応を変えたり，外部のファイルをアクセスしたいときなどに利用する．属性の値を属性値という．属性値は（“ ）や（‘ ）で囲む．

ひとつのタグ内で同じ属性名を複数回使うことはできない．

例

```
<要素名　属性名 1=”属性値 1”　属性名 2=”属性名 2” ・・・>
```

・空要素

内容を持たない要素を空要素という．

```
<要素名/>
```

このように記述する．また

```
<要素名></要素名>
```

でも構わない．一般には，空要素はいずれかの属性と一緒に使う．

```
<student ID=”254054” />
```

2.1.4 DTD

DTD（Document Type Definition：文書型定義）はXML文書の設計図のようなもので，タグ名やデータ構造などを定義する．

主に以下の4つを宣言する．

・要素型宣言（要素の名前と階層構造を指定）

・属性リストの宣言（各要素に指定可能な属性を定義）

・エンティティ宣言（置き換えたい文字列や参照したい外部ファイルと，その参照名の指定）

・そのほか，記法宣言，処理命令，コメントなど

内部DTDと外部DTD

DTDには内部DTDと外部DTDがある．

内部DTDはXML宣言に続いて記述する．DTDの記述はDOCTYPEで開始する．

```
<!DOCTYPE ルート要素名 [...DTDの記述...]>
```

DTDの記述を外部のファイルに置くことができ，外部DTDという．複数のXML文書が同一のDTDを利用する際に重要な方式である．DTDのありかは以下のようにURIで指定する．

```
<!DOCTYPE ルート要素名 PUBLIC "公開識別子" "URI" [ DTDの記述 ]>
                        └──公開されているDTDの場合 (公開識別子を持つ場合)

<!DOCTYPE ルート要素名 SYSTEM "URI" [ DTDの記述 ]>
          公開されていない┘        └── 外部DTDの場所
            DTDの場合
```

［DTDの記述］の記述は省略可能．

公開識別子（Public Identifer）はDTDを認識するための単一名称．

外部DTDを使用するかどうかの指定はXML宣言部のstandalone属性で指定する．

```
<?xml version="1.0" encoding="EUC-JP" standalone="yes" ?>
```

外部DTDやパラメータ宣言を参照する場合はno．参照しない場合はyesを指定する．

文書型宣言

DTD内ではDOCTYPE,ELEMENT,ATTLISTの各宣言を使って定義を行う．

・DTDとルート要素の指定　DOCTYPE

DOCTYPE宣言はルート要素を指定する．DTDの記述はこのDOCTYPEの宣言から始めなくてはならない．

```
<!DOCTYPE root要素 [
        .....
        .....
]>
```

・要素を宣言　ELEMENT

ELEMENTを使用して要素の名前と要素間の親子関係（内容モデル）を指定する．

```
<!ELEMENT 要素名 (内容モデル)>
```

内容モデルは要素間の親子関係を指定するもの．その要素の下にくる複数の要素名を指定する．

```
<!DOCTYPE root [
                        親要素
<!ELEMENT root ( AAA, BBB ,CCC)>
                              ┐
<!ELEMENT AAA ( DDD, EEE )>   │
<!ELEMENT BBB (#PCDATA) >     │
.....                         │
.....                         子要素
.....                         │
]>                            ┘
```

要素BBBのように要素内に実際の文字データを置きたい場合，要素宣言で#PCDATAを使用する．#PCDATAは要素宣言の末端になければならない（EMPTY要素は除く）．すべての要素宣言が#PCDATAで終わっていなければ，DTDは構文的に誤りである．

要素型宣言では，要素名と内容モデルを指定するが，同じ要素が繰り返し出てくる場合には，それを何度も記述するのではなく，繰り返し指定の表現を使って記述する．

記号	意味	例
+	1回以上の繰り返し	aaa+
*	0回以上の繰り返し、省略も可	aaa*
?	省略または1回のみ	aaa?
,	連続(a,b,a要素の次にb要素)	a,b
|	|の左右の要素のいずれか一方	aaa | bbb

繰り返しの例をみてみる．

DTDの記述

```
<!DOCTYPE root[
…
<!ELEMENT AAA (BBB)+ >
<!ELEMENT BBB ( A,B,C )
>
…
]>
```

DTDの記述に対応するXML文書

```
<root>
……
……
    <AAA>
        <BBB>
            <A>内容 1</A>
            <B>内容 2</B>
            <C>内容 3</C>
        </BBB>
        <BBB>
            <A>内容 4</A>
            <B>内容 5</B>
            <C>内容 6</C>
        </BBB>
……
……
    </AAA>
</root>
```

繰り返し

空要素はタグだけで構成される要素で，実際のデータは持っていない．DTDではEMPTYを使ってあらわす．

```
<!ELEMENT 要素　EMPTY >
```

XMLデータの記述は次のようになる．

```
<要素></要素>　あるいは　　<要素/>
```

多くの場合，空要素には属性を定義する．空要素に指定した属性はグラフィックスなどの所在を示したり，検索や選択でメタ情報として利用する．
以下のように図の番号を指定したり，商品要素に対して属性名と値の組み合わせを条件に検索するなど，データベースのような応用が可能になる．

```
<図　図番号="fig1-2" />
<商品　タイプ="BOX" 色="RED" 重さ
="300" />
```

HTMLにXMLで記述したメタ情報を追加すると，HTML情報検索的な機能を付け加えることができる．HTMLで書かれたWebコンテンツを活用しつつ，高度な応用が可能になるのもXMLの大きな魅力のひとつである．
現在，HTMLとXMLの混在する場合のXHTML規約が，W3Cのワーキンググループで討論されている．HTMLが整形式（Well-Formed）に準拠されれば，パーサーなどのXML処理系で扱えるので，XML活用のひとつとしてXMLとHTMLの混在利用が広がるであろう．
任意の要素を子要素にしたい場合は，ANYを使う．この宣言が含まれているDTD内で定義されている任意の要素を子要素にできる．

```
<!ELEMENT AAA ANY >
```

属性リストの宣言
同じ要素でも個々に区別をつけたいときは，属性情報を設定する．属性を使うには，あらかじめDTDで属性を宣言しておく必要がある．DTDでは属性名，属性のデータ型，省略時の値の指定などを記述する．
属性はXML文書の内容記述として利用するより，管理や検索のために利用する．
属性の宣言にはATTLISTを使う．

```
<!ATTLIST 要素名　属性名　データ型　省略指定>
```

属性のデータ型は属性値のデータ型を指定するもので，以下のものが指定できる．

データ型の指定	属性の値として許されるデータ
CDATA	文字データ
NMTOKEN, NMTOKENS	数値(NMTOKENS は、複数の数値)
ENTITY, ENTITIES	ENTITY で宣言された参照名。実際の参照は ENTITY に宣言された参照名を経て指定内容が参照される。

属性の省略を許す指定や省略時の暗黙値の指定ができ，省略指定には以下の3種類がある．

指定	意味
#REQUIRED	属性値を必ず指定。
#IMPLIED	属性値は省略可能。省略すると既定の値と解釈される。
#FIXED	DTD 中で指定された値が指定値と解釈される。XML 文書中で他の値を設定するとエラーになる。

エンティティの利用

エンティティ（entity）はコンピュータ上に格納されている情報の記憶単位のことである．具体的には，ファイルや文字列，XMLでは直接扱えないようなグラフィックデータやマルチメディアデータなどのファイルを指す．エンティティには名前（エンティティ名）がつけられていて，この名前をもとに，実際のエンティティを参照する．

・一般エンティティ

長い文字列などを利用するときは，DTD中でその文字列とエンティティ名（短縮名）を宣言しておき，XML文書から対応するエンティティ名を参照することで，もとの文字列に展開することができる．

このようにユーザが文字列を定義してその文字列に名前をつける役割を果たすエンティティを一般エンティティと呼び，以下のように指定する．

```
<!ENTITY エンティティ名　“ 置き換える文字列 “>
```

エンティティを参照するときは以下のようにエンティティ名を指定する．&で参照が始まり，;で参照が終わる．

```
&エンティティ名;
```

XML文書の部分データを外部に置いてURIで引用することもできる．もとのXML文書から参照する外部ファイルを外部エンティティと呼び，以下のように指定する．

```
<!ENTITY エンティティ名　SYSTEM “参照するファイルの場所 “>
```

エンティティを参照するときは，以下のようにエンティティ名を指定する．

```
&エンティティ名;
```

・解析対象外エンティティ

グラフィックデータやバイナリデータなどXMLでは直接扱えないデータを解析対象外エンティティという．そのようなデータを扱う場合には，そのデータを実行するアプリケーションのありかを指定する必要があり，これを記法宣言（Notation Declaration）という．

記法は以下のようにNOTATIONを使って宣言する．

```
<!NOTATION 記法名　SYSTEM “記法データを処理できる実行ファイル
の場所　“>

<!NOTATION 記法名　PUBLIC “公開識別子　“　“ 記法データを処理
できる実行ファイルのありか　“>
```

次の例は，実行ファイルの所在を示す記法宣言．

```
<!NOTATION cgm SYSTEM “file:/draw/mycgm.exe” >
```

このNOTATIONに対応する属性宣言は以下のようにする．

```
<!ATLIST 図　形式 (cgm|dxf)>
```

またDTDなしで使用可能なエンティティがあり，以下に示す．

表現したい文字	エンティティ名
<	lt
>	gt
'	apos
“	quot
&	amp

2.1.5 XML文書の例

サンプルXML文書

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- DTD( 文書型定義 ) -->
<!DOCTYPE sample SYSTEM "sample.dtd">

<!-- XML 文書　-->
<sample>
    <title>XML 文書の例 </title>
    <contents>
        <subtitle>XML 文書の表示例 </subtitle>
        <topics> これは XML 文書のサンプルである。 </topics>
    </contens>
</sample>
```

サンプル XML 文書

1行目はXML宣言である．妥当なXML文書はXML宣言で始まるべきである．XML宣言には終了タグはない．XML宣言の基本構文は，

<?xml version=“バージョン情報” encoding=“文字コード” standalone “スタンドアロン宣言”?>

である．上記の例ではバージョン指定と文字コードのみである．

3行目は文書型定義である．文書型定義はDTDを実際の文書と結びつけたり，または文書構造

について独自の宣言を行うことができる．

サンプルDTD

```
<!ELEMENT sample (title,contents)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT contents (subtitle | topics*)>
<!ELEMENT subtitle (#PCDATA)>
<!ELEMENT topics (#PCDATA)>
```

サンプル DTD

各行は要素型宣言である．基本的な要素型宣言の構文は
<!ELEMENT 要素名 内容モデル>
である．
1行目はルート要素sampleがtitleとcontents要素を持ち，出現順序はtitle，contentsの順である．
2，4，5行目の#PCDATAは要素の内容モデルであり，子要素が存在しないテキストのみの要素であることを意味する．
３行目は要素contentsがsubtileとtopics要素を持ち，出現順位は任意であり，要素topicsの後の*は0回以上の出現を意味する．

2.2 XPath

XpathはXML文書の一部を指し示すための言語であり，W3C勧告である．また，XML文書中に登場する文字列や整数，真理値などのさまざまなデータ型を処理する機能も持っている．XPathが指し示す対象は，XML文書中の要素，属性やテキストを組み合わせたノードの集合である．このデザインのおかげで，XPathは使う人が直感的に理解しやすいようになっている．また，XML文書を木として扱うことで，数学的な見地からその性質を形式化できる可能性も秘めている．

2.2.1 XPathの文法と意味

UNIXやWindowsのようにXPathにも，ファイルシステムと同様に，木構造のXML文書におけるノード集合を指し示すためのパスの記述方法がある．XPathにおけるパスはロケーションパスと呼ばれる．親要素とその子要素はスラッシュ（ / ）を用いて区切られる．例えば，次のXPathによって，あるXML文書の「文書要素の子要素であるHeader要素の子要素であるSignature要素（複数の場合もある）」を指し示す．

/Evelope/Header/Signature

Xpathでは，スラッシュで区切られた各要素はロケーションステップと呼ばれる．上記の例では，Envelope，Header，Signatureがそれぞれロケーションステップである．

ファイルパスとXPathには１つ違いがある．それはXPathが１つのノードではなく，ノード集合を指し示す点である．ファイルパスの場合は，通常，１つのファイルパスで指し示されるファイルやディレクトリは１つだけである．XPathが１つのノードしか指し示さない場合であっても，その評価結果は要素を1つしか持たない場合と考えられる．上記の例では，XPathが必ずしも１つの要素をユニークに指し示すとは限らない．すなわち，１つのHeader要素の下には１つ以上のSignature要素が存在するかもしれない．その場合は，XPathはSignature要素の集合を選択することになる．

ファイルパスの指定方法には，絶対パスと相対パスの２種類がある．同様に，XPathにも絶対的な記述方法と相対的な記述方法がある．すなわち，絶対ロケーションパスと相対ロケーションパスである．絶対ロケーションパスで基準となるノードはXML文書の文書ルートである．相対ロケーションパスの基準となるノードを文脈ノードと呼ぶ．UNIXやWindowsでのカレントディレクトリの考え方に似ている．文脈ノードはXPathを評価するときに決定される．

Xpathでは，正確性に重きを置いた一般的な記述方法とは別に，簡潔性に重きを置いた省略記法がある．上記のXpathの例は一般的な記述方法の省略記法である．

/Evelope/Header/Signature ... 省略記法

/chiled::Evelope/child::Header/child::Signature ... 一般的な記法

Xpathの一般的な記述方法では，文脈ノードとロケーションステップで選択されるノードの関係を明示的に記述する．この関係のことを軸と呼ぶ．軸には，たとえば「**child::**要素名」や「**attribute::**属性名」などがある．

ある与えられた文脈ノードに対して，軸がどのような意味を持つかを図2.1に示す．ここでは真中のノードが文脈ノードである，他のノードを参照する矢印が軸を表している．たとえば，parent軸は文脈ノードの親ノードを参照する．

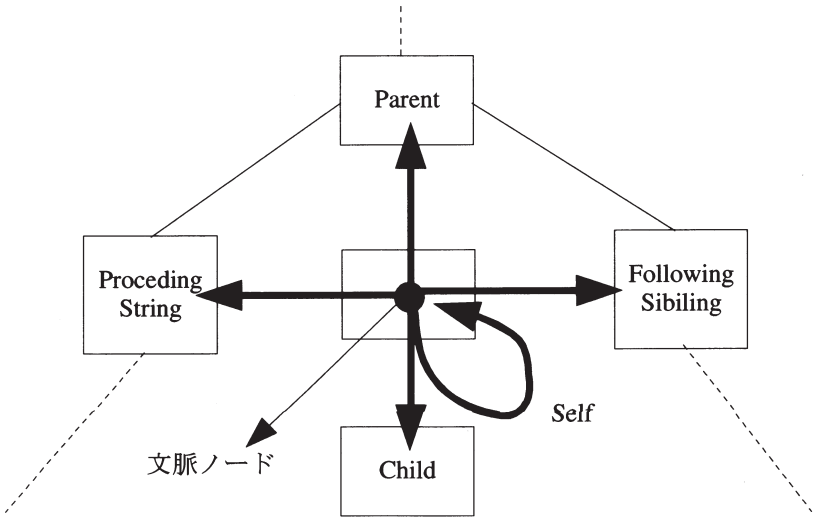


図2.2：XPathにおける軸と文脈ノード

図2.2で，軸をもう少し詳しく説明する．XPathには文書順序と呼ばれる順番が存在する．文書順序とは，文字列のままのXML文書において，それぞれのノードの開始タグが登場する順番のことである．たとえば図2.2では，太字で示された開始タグの例（A，B，C，D，.....）が文書順序を表す．軸は文書順序を基準に定義される．たとえば，preceding軸は文書順序で文脈ノードより前に現れるすべてのノードを選択する．

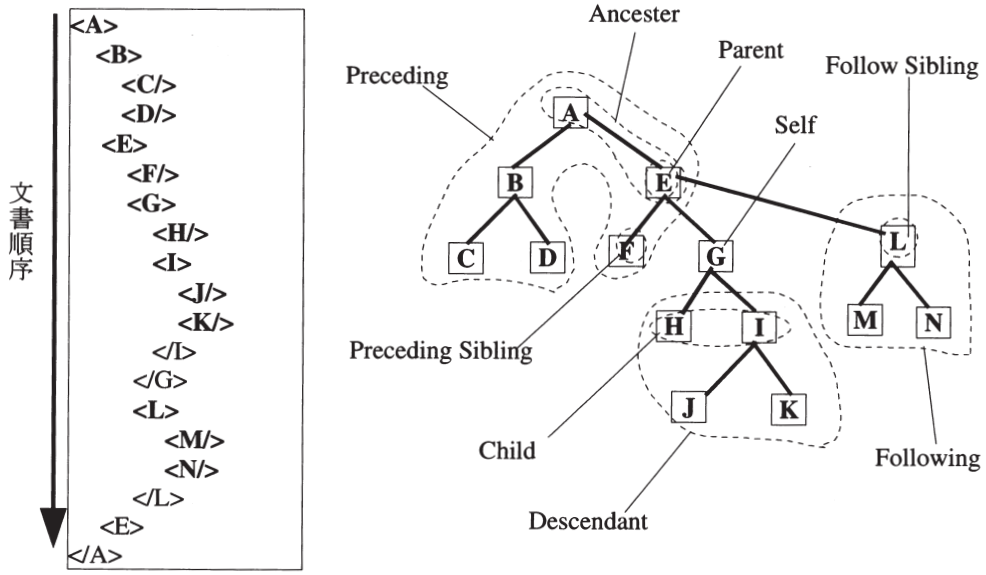


図2.3：XPathの典型的な軸の例

要素名，@属性名といった省略記法が，前述の一般的な記法に対応して用意されている．すなわち，この例では「child::要素名」，「attribute::属性名」が一般的な相当する．もちろん，一般的な記法とそれに対応する省略記法はまったく同じ意味を持つ．

次に，XPathプログラミングでしばしば使われる典型的な例を表2.1に紹介する．

表2.1：Xpathの例

Xpath の例	省略記法	説明
/	/	文書ルートを選択する。文書要素ではないことに注意。
child::person	person	文脈ノードの子要素の中から person 要素を選択する。
child::person | /	person | /	Person 要素と文脈ルートの両方を選択する。"|"は2つの XPath の和をとる。
/child::AddressBook	/AddressBook	文書ルートの子要素の中から AddressBook 要素を選択する。これは、文書要素が AddressBokk ならそれを選択し、そうでなければ何も選択されないことを意味している。
child::*	*	文脈ノードのすべての子要素を選択する。テキストや属性は選択されないことに注意。
child::text()	text()	文脈ノードの子ノードの中からすべてのテキストを選択する。要素の前後にテキストがある場合はそれらが結合されるわけではなく、テキストの集合として選択されることに注意。
attribute::type/descendant-or-self::node()/child::email	@type//email	文脈ノードの type 属性を選択する。文書に含まれるすべての email 要素を選択する。このように、//は/descendant-or-self::node()/の省略形である。node()関数はそのノード型にかかわらずすべてのノードを選択する。
child::person/child::email [position()=1]	person/email[1]	文脈ノードの子要素の中の person 要素の最初の email 要素を選択する。この例のように、" [ ] "の中には述部(Predicate)を記述できる。簡単に言うと、選択される要素に「ただし」の条件を付けられるのである。
child::person[child::email and attribute::id="12345"]	person[email and @id="12345"]	文脈ノードの子供の中の person 要素のうちで、少なくとも 1 つの email 要素を持ち、id 属性の値が 12345 であるものを選択する。

child::name [attribute::family="kozima"] [position()=last()]	name[@family="kozima"] [last()]	文脈ノードの子要素の中のname要素のうちで、family属性の値がkozimaである最後の要素を選択する。

2.2.2 オブジェクトと型

Xpathを評価した結果はオブジェクトと呼ばれる．オブジェクトは，以下４つの基本型のうちのいずれかの型を持つ.

１．ノード集合
　重複を許さず順序も持たないノードの集まり
２．文字列
　文字列の並び
３．真理値
　真（true）または偽（false）
４．数値
　浮動小数点

ここでノード集合について注意がある．それは，対象となるXML文書を表す木自身は順序を持った木として処理されるが，結果のオブジェクトであるノード集合はその定義により順序を持っていない，ということである.

Xpathでは，ロケーションステップの中で使用するさまざまな関係が用意されている．それらは４つの基本型に対応して4種類のグループに分類される．XPathの代表的な関数を表7.2にあげる．これらの関数は，基本的にそれぞれ対応する型の引数をとる．もし，それ以外の型の引数が渡された場合は，XPathで定義された変換規則に従って型変換を行う．たとえば，空でないノード集合は真理値の“真”に変換される．ただし，文字列，真理値，数値をノード集合に型変換することはできない.

表2.2：XPathの代表的な関数

関数グループ	型変換関数	代表的な関数	説明
ノード集合	なし	数値 last()	選択されたノードの数を返す（これを文脈サイズと呼ぶ）。
		数値 position()	選択されたノードインデックスを返す（これを文脈ポジションと呼ぶ）。

		数値 count(ノード集合)	引数のノード集合の中のノードの数を返す。
		文字列 local-name(ノード集合?)	ノードの局所部分を返す。その際、与えられたノード集合の中で文書順序での最初のノードが対象となる。
		文字列 namespace-uri (ノード集合?)	ノードの名前空間 URI を返す。その際、与えられたノード集合の中で文書順序での最初のノードが対象となる。
文字列	文字列 String(オブジェクト?)	文字列 concat(文字列, 文字列,文字列*)	引数の文字列を結合した文字列を返す。
		真理値 starts-with(文字列,文字列)	第 1 引数の文字列が第 2 引数の文字列から始まるかどうかをチェックする。
		真理値 contains(文字列, 文字列)	第 1 引数の文字列が第 2 引数の文字列を含むかどうかをチェックする。
		数値 string-length(文字列?)	引数の文字列の長さを返す。
		文字列 normalize-space (文字列?)	前後の空白文字を取り除き、連続する空白文字を 1 つの空白文字で置き子会えることで、与えられた文字列の空白文字を正規化する。
真理値	真理値 boolean(オブジェクト)	真理値 not(真理値)	引数の真理値の否定をとる。
		真理値 true()	常に真を返す。
		真理値 false()	常に偽を返す。
数値	数値 number(オブジェクト?)	数値 sum(ノード集合)	与えられたノード集合の和を返す。その際、ノード集合中の各ノードは一度文字列に変換され、さらに数値に変換される。

一般的に，関数グループに属する関数はそのグループのプリミティブ型に対応する型の引数をとる．たとえば，真理値の関数グループの関数は真理値の引数をとる．もしある関数がその関数に合わない型の引数を与えられて呼び出されたら，Xpath の処理系は暗黙のうちに型変換の関数を呼び出して引数を適切な型に変換する．たとえば，空でないノード集合は booklean()関数によって真理値型の真に変換される．その意味で，型変換の関数の振る舞いは XPath の型変換のルールを定めたものと考えられる．ただし，文字列，真理値，数値からノード集合への変換はできないことに注意する必要がある．

第3章 XML文書の一貫性制約

本章では，構造化文書における制約表現の必要性，DTDでの制約表現の問題について述べ，木構造に基づく一貫性制約をどのように表現するべきかを述べる．

3.1 構造化文書における制約表現の必要性

XML（eXtensible Markup Language）は自己拡張可能なマークアップ言語である．文書構造を決定するための規則を文書の作成者が決定できるのである．
これにより，XMLという標準仕様を用いながら，特定の構造にとらわれない自由度の高い文書を広く交換できるようになる．インターネットの普及に伴って，これを利用した電子商取引が活発化したことから，電子商取引においてネットワーク上でやり取りする標準文書仕様として注目されることとなった．

XMLは文書形式の自由度で優れているが，コンピュータでの管理・自動処理を考えると処理対象として受けとったデータが適正なものであるかどうかを調べる必要がある．電子商取引等におけるXML文書では，構造などの制約だけでなく，不正なデータを受け付けないための制約も必要となってくる．不正なデータを検出する機能が必要である．インターネット上では，相手から送信されてきたデータが悪意が無く，「正しいデータ」である保証はないのである．

例えば以下の図3.1のような発注書が送信されてきたとする．
この場合，「数量の値」と「単価の値」を掛けた値が「金額の値」と一致しなければいけない，などの一貫性制約が必要と考えられる．また企業によっては「数量の値」は100以上でなければならないなどの制約も考えられる．このようなXMLで表現されたビジネスデータの意味的制約を表現する能力はDTDには無い．

コード	商　品　名	数　量	単　価	金　額	備　考
				0	
				0	
				0	
				0	
				0	
				0	
				0	
				0	
			小　計	0	
			税　額	0	
			合　計	¥0	

図3.1：発注書の例

こういった一貫性制約表現を用いることによって，ショッピングサイト等で，0の数がひとつ足りずに桁が間違ったままの金額を表示してしまう等の誤りも未然に防ぐことができる．

XML文書の一貫性をチェックするプログラムは必要であるが，それを応用プログラムに実装すれば，プログラム開発の負担となる．
XML文書の一貫性管理を応用プログラムから切り離しておき，一ヶ所で集中してチェックする仕組みがあれば有用である．すなわち，納品書XML文書，発注書XML文書，御見積書XML文書など，一貫性制約の必要なXML文書の一貫性管理を一ヶ所で一括して行う．また，個々の応用プログラムから切り離して実現することで，専用のXMLエディタなどを用いる必要もない．つまりどのような環境で作成されたXML文書でも一貫性のチェックを行うことができる．

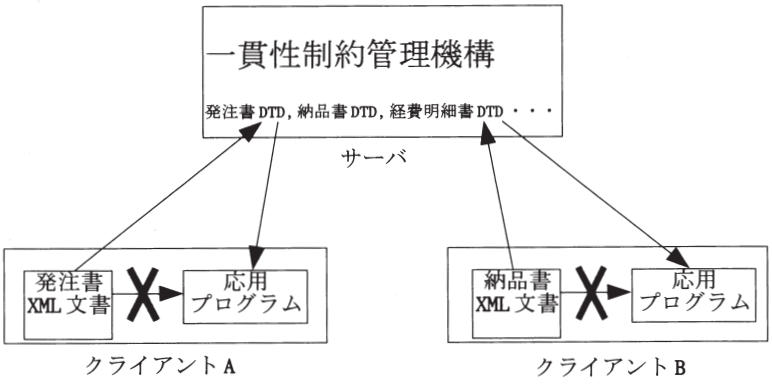


図3.2：一貫性の集中管理

次のような応用例が考えられる．

・自社内で作成したXML文書の一貫性検証
発注書や明細書などのXML文書のDTDを自社内の一ヶ所にまとめて保存しておき，XML文書の作成時にSDICに従っているかの検証を行う．

・他企業からの自社の指定したDTDに従ったXML文書の一貫性検証
他企業へあらかじめDTDを指定しておき，他企業から送られてきた納品書や発注書などのXML文書のSDICに従っているかの検証を行う．そのDTDは自社内の一ヶ所にまとめて管理しておく．

・Webサイトでの利用者からの入力の一貫性検証
ショッピングサイトなどで利用者からの入力が不正なものでないかどうかの検証を行う．購

入額や購入数などの桁間違い等，あきらかに間違ったデータの検出を行う．

3.2 DTDによる制約表現の問題点

DTDでは，XMLで文書を記述する際，その文書中でどのようなタグや属性が使われているかを定義し，文書の構造を定める．これもある意味での「制約」を表現しているが，それでは不十分で上手く表現できない「制約」がある．

例えば，DTDでは，属性に対していくつか用意されている以外，データ型がほとんど用意されていない．ある属性widthに整数の値が指定されてほしい場合でも，属性値の型に整数型などというものがないので， width=“abc” のような属性指定を認めないようにできない．
DTDをスキーマとするXMLプロセッサがデータ型を十分チェックできないので，プロセッサからデータを受け取るアプリケーション側でチェックしなければならない．

また，XML文書の内容の一貫性もDTDだけでは制約できない．
以下の図3.3のようにDTDで構造を定義されたXML文書に，ノードA,B,Cがあったとする．構造は定義できるが，「Aの値 > Bの値」でなければならない，「Aの値 + Bの値 = Cの値」でなければならない，などもDTDでは表現できない．
本研究では，XML文書に含まれる種々のデータについて，その意味的制約条件の表現方式に注目する．

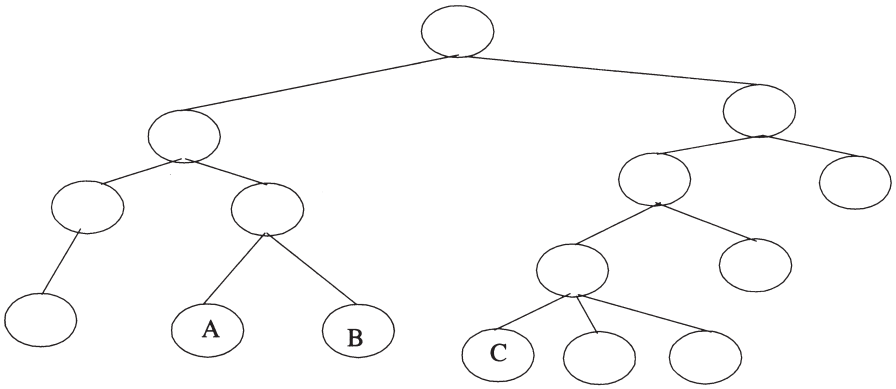


図3.3：DTDで構造を定義された木構造

3.3 木構造に基づく一貫性制約表現方式

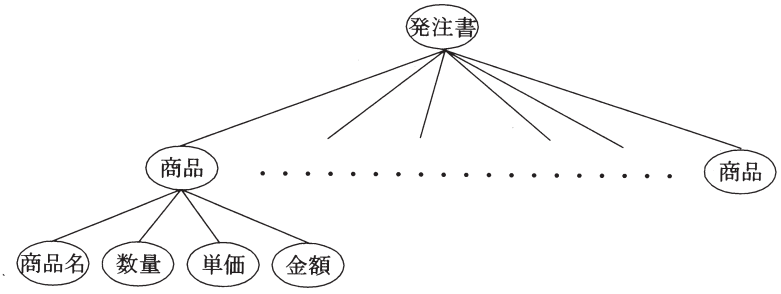
文書の一貫性を表現するためには，XML文書の木構造中の部分木を特定する必要がある．
Xpathで表現した部分木間の一貫性制約を「文書一貫性制約（Semi-structured Document Integrity Constraints)」と称し（以下SDIC)，それをシステムに格納しておくことで，XML文

書の一貫性管理を応用プログラムから切り離して実現する方法を本論文では提案する．

SDICを表現するのにはJavaで扱える式を用いることにする．XPathで表現された部分木にはあらかじめ，任意の名前を与えておく．

次の発注書を例にあげる．

コード	商　品　名	数　量	単　価	金　額	備　考
	○○○	0	100	100,00	
	△△△	150	200	300,00	
	□□□	200	300	60,000	
				0	
				0	
				0	
				0	
				0	
				0	



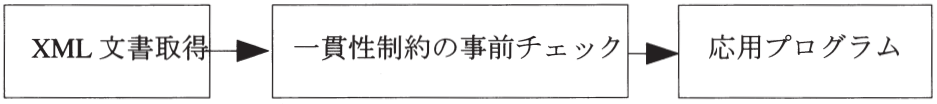
/child::order_sheet/child::contents_of_order/child::goods/child::unit_price/child::text()
というXPathで表現された部分木に「**unit_price**」という名前をつける．
これは上記の発注書の単価を示す．

/child::order_sheet/child::contents_of_order/child::goods/child::quantity/child::text()
このXPathで表現された部分木には「**quantity**」という名前をつける．
これは上記の発注書の数量を示す．

/child::order_sheet/child::contents_of_order/child::goods/child::total_price/child::text()
このXPathで表現された部分木には「**total_price**」という名前をつける．
これは上記の発注書の金額を示す．

この，**uniti_price,quantity,total_price**と名付けられた部分木を使ってSDICを表現すると
uniti_price * quantity == total_price
quantity >= 100
(unit_price * quantity)*1.05 == 100000
等のような式になる．

このようなSDICを一括して格納したシステムを用意できれば，XML文書の一貫性管理を応用プログラムから切り離して実現することができる．
ここで，取得したXML文書の処理の流れは次のようになる．



このような手順を踏むことになる．

これにより，例えば企業システム全体にわたって，XML文書の一貫性制約を集中管理でき，応用プログラムを作成するプログラマの負担を下げることができると考えられる．

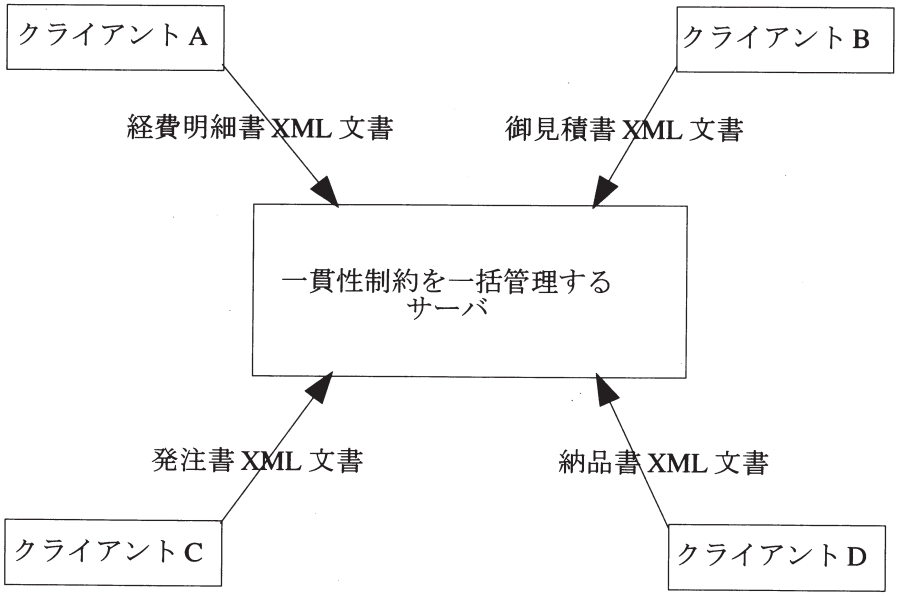


図3.4：企業システム等での一貫性制約の集中管理

第4章 システムの設計と実装

前章で述べた，XML文書のSDICの一括管理を行うシステムの概略，および設計と実装について述べる．SDICの管理を行うためのデータベースの実装にPostgreSQLを用いる．このシステムの実装には，プラットフォームに非依存なプログラム開発言語Javaを用いる．また，XML文書の妥当性を検証するための解析ツールであるXMLプロセッサとしてXercesパーサを用いる．

4.1 システムの概略

本節では，SDICチェックシステムの概略について述べ，システム利用の際の前提条件について述べる．

4.1.1 システムの概略

第3章で説明したとおり，SDICを格納しておき，XML文書の一貫性管理をできるシステムである．

システムはサーバとクライアントに分ける．クライアントから送られてきたXML文書からDTDを特定し，サーバ側で管理しているSDICに基づいてチェックを行う．

4.1.2 前提事項

前提条件として，XML文書に適用するDTDは外部参照とする．DTDは一括して管理しておき，XML文書内で使用するDTDのURIを指定するためである．

4.2 SDIC格納用データベースの設計

4.2.1 保存するデータ

データベースに登録する情報は以下に示すようなDTDのURI，XPath，XPathに与えた任意の名前，SDICの表現，である．

・DTDのURI

DTDの存在する場所を示すURI.

例

file:///home/ktakuya/dtd/sampleDTD.dtd

・XPath

XML文書の特定のノードを示すXPath.

一般的な記述方法でも省略記法でもどちらでもよい．

例

一般的な記述方法

/child::order_sheet/child::contents_of_order/child::goods/child::quantity/child::text()

省略記法

/order_sheet/contents_of_order/goods/quantity/text()

・**XPath**に与えた任意の名前

上記のXPathに与えた任意の名前．その名前を使ってSDICを表現する．ただし，ひとつのDTD内で同じ名前は使えない．また，数字は使用できない．

・**SDIC**の表現

上記で与えた名前を使い，Javaで使用できる式を使う．

例

Xpathに**a,b,c**という名前を与えたとする．

a == b	**a**の値と**b**の値は一致しなければならない
a >= 200	**a**の値は200以上の値でなければならない
a > 200	**a**の値は200より大きい値でなければならない
a * 3 == b	**a**の値に3掛けた値と**b**の値が一致しなければならない
a / 4 == b	**a**の値を4で割ったものと**c**の値が一致しなければならない
(a + b)*2 == c	(**a**の値 + **b**の値)の倍の値が**c**の値と一致しなければならない
c != 100	**c**の値は100であってはならない

以上のような表現である．

4.2.2 データのテーブル

これらを格納するテーブルを2つ用意する．

ひとつはDTDのURI，XPath名，XPathを格納する．これをxpathtableとする．

表4.1のようになる．

表4.1

xpathtable		
DTD の URI	Xpath Name	XPath
file:///home/ktakuya/dtd/sampleDTD.dtd	quantity	/order_sheet/contents_of_order/goods/quanti ty/text()
file:///home/ktakuya/dtd/sampleDTD.dtd	unit_price	/order_sheet/contents_of_order/goods/unit_p rice/text()
file:///home/ktakuya/dtd/sampleDTD.dtd	total_price	/order_sheet/contents_of_order/goods/total_ price/text()
file:///home/ktakuya/dtd/sampleDTD2.dtd	quantity	/order_sheet/contents_of_order/goods/quanti ty/text()
file:///home/ktakuya/dtd/sampleDTD2.dtd	unit_price	/order_sheet/contents_of_order/goods/unit_p rice/text()
file:///home/ktakuya/dtd/sampleDTD2.dtd	tax	/order_sheet/contents_of_order/goods/tax/te xt()

XPathに与えられた名前はDTDが異なれば同じ名前でもよい．表4.1ではunit_price，quantityという名前が2つずつあるが，DTDが異なるので構わない．ただしXPath名には数字は使用できない．

もうひとつはDTDのURIとSDICを格納する．これを***exptable***とする．
表4.2のようになる．

表4.2

exptable	
DTD の URI	SDIC
file:///home/ktakuya/dtd/sampleDTD.dtd	quantity * unit_price == total_price
file:///home/ktakuya/dtd/sampleDTD.dtd	quantity >= 1000

xpathtableに格納されているXPath名を使ってSDICの表現したものを格納する．

4.3 実際の動作

システムはXML文書を受け取ると，そのXML文書のDOCTYPE宣言からDTDのURIを抜き出す．

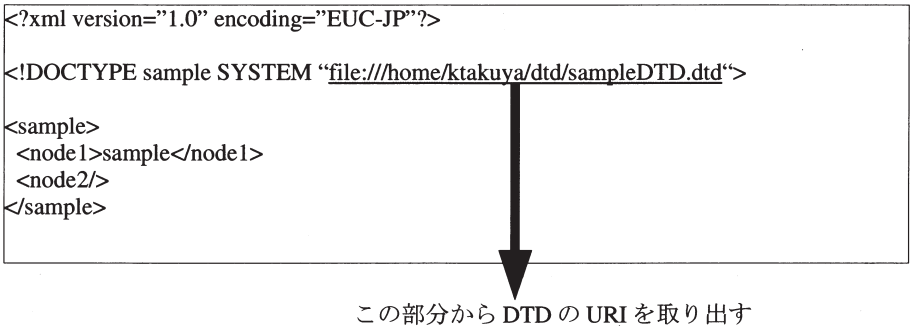


図4.1：DTDのURIの抜き出し

次に抜き出したDTDのURIをもとに***exptable***からそのDTDで使用されるSDICの表現を取り出す．そしてSDICに使用されている文字列のうち「()+-*/><=!」の文字列と数字以外の文字列をXPath名とみなし，XPath名を抜き出す．

例えば，

SDICが「**(a + b)*2 == c**」の場合

a，b，c　の3つのXPath名が得られる．

SDICが「**quantity >= 1000**」の場合

quantityというXPath名が得られる．

次に得られたXPath名をもとに***xpathtable***から使用DTDとXpath名の一致するXPathを取り出す．そしてXPathからXML文書のノードを参照し，値を取り出し，SDICのXPath名の部分と置き換える．こうしてJavaで扱える式を作成する．図4.2のようになる．

(a + b)*2 == c　　⇒　　(100+200)*2 == 600

図4.2：XPath名と実際の値の置き換え

そして，式を左辺，右辺，評価記号に分割する．図4.3のようになる．

(100+200)*2 == 600　　⇒　　(100+200)*2　600　==

図4.3：式の分割

最後に左辺と右辺の関係が評価記号を満たすかどうか判定し，真か偽を返す．

真であればSDICに従った正当なデータであり，偽であればSDICに従わない不正なデータを持つということである．

4.4 クラス構成

システムの実装で作成した代表的なクラスは以下の通りである．

ConsistClient class

クライアント側でシステムを起動する際に使うメインプログラム．XML文書のパスを引数として与えると，後述のConsistMServer classにソケットを送る．

ConsistMServer class

ConsistClientから送られてきたソケットを，マルチクライアントを実現するためConsistMSTheread classのスレッドで処理をする．

ConsistMSThread class

ConsistClientから送られてきたソケットを処理する．XML文書，DTDを引数として後述のExpDB classを呼び出す．処理結果をクライアント側に送信する．

ExpToken class

与えられた文字列を，左辺，評価記号，右辺に分ける．左辺，右辺の計算結果を文字列，float型で返す．

・コンストラクタ

ExpToken（String s）

文字列の数式（　(140+3)*2 !=9　のようなもの）を与える．

左辺，評価記号，右辺に分けて，左辺と右辺を計算．

・メソッド

float getLeftValue()

左辺の値を返す．

float getRightValue()

右辺の値を返す．

String getSing()

評価記号を返す．

String getLeftString()
左辺の計算結果を文字列として返す.

String getRightString()
右辺の計算結果を文字列として返す.

TorF class
右辺と左辺の２つの値と評価記号を引数として受け取り，右辺と左辺の関係が評価記号を満たしているかを調べる.

・メソッド
boolean checkTF（float left, String s, float right）
leftとrightをsの評価記号で評価して，真か偽を返す.

XpathOpen class
XpathとXML文書のパスを受け取り，XPathの指し示すノード（またはノード集合）を与えられたXML文書から返す.

・コンストラクタ
XpathOpen（String xmlFilePath）
XML文書のファイルパスを初期値として与える.
そのXML文書をパース.

・メソッド
public ArrayLIst getNodeContents（String xpath）
Xpathを引数としてとり，XPathの指し示すノード（またはノード集合）を与えられたXML文書から探し，ArrayListに追加していく．そのArrayListを返す.

ExpFromDB class
データベースと接続し，DTDを元にデータベースからXPathやSDICを取り出す.

・コンストラクタ
ExpFromDB（String xmldocPath, String dtd）
与えられたXML文書のパスとdtdによって，データベースに格納されているXPathやSDICを

取り出し，XPathOpen classを用いて，SDICの内容をXPathの示すノードの値と置き換え，Javaで扱える状態の評価式にする．

・メソッド
public ArrayList getExpression()
Javaで扱える状態の評価式を返す．

DBConnect class
データベースへの接続を行うためのクラス．

DBElementsOperate class
データベースのデータに関する操作を行うクラス．

・メソッド
public boolean isDTDnew（String dtd）
DTDがテーブル内に存在するかしないかを返す．
引数で与えられたDTDが，存在しない場合は新規DTDとなりtrueを返す．

public boolean isOverlap（String name,String dtd）
XPathNameが重複していないかどうか調べるメソッド．
重複する場合はtrueを返す．

public boolean sdicCheck（String sdicXPathName,String dtd）
SDIC表現に使われているXPathNameが登録されているかどうかを調べるメソッド．
登録されているならtrue，テーブル内に見つからない場合はfalseを返す．

public boolean isSDICexsist（String sdic,String dtd）
SDICがテーブルに存在するかどうか調べるメソッド．
存在するならtrue，存在しないならfalseを返す．

public void addNameAndXPath（String dtd, String name, String xpath）
テーブルにname,xpathを追加するメソッド．

public void addSDIC（String dtd, String sdic）

テーブルにSDICを追加するメソッド.

public void delSDIC（String dtd, String sdic）
テーブルからSDICを削除するメソッド.

public void delNameXPath（String dtd, String name, String xpath）
テーブルからXPathName,XPathを削除するメソッド.
削除しようとしているXPathNameがSDIC表現で使用されている場合,
そのSDIC表現も一緒に削除する.

public ArrayList showCatalogue（String dtd）
テーブルの一覧表示を表示させるメソッド.

ConstAddClient class
SDIC表現をシステムに登録・削除するためのGUI.

第5章　システムの実行例

制約条件登録**GUI**
図5.1は制約条件登録用GUIである.

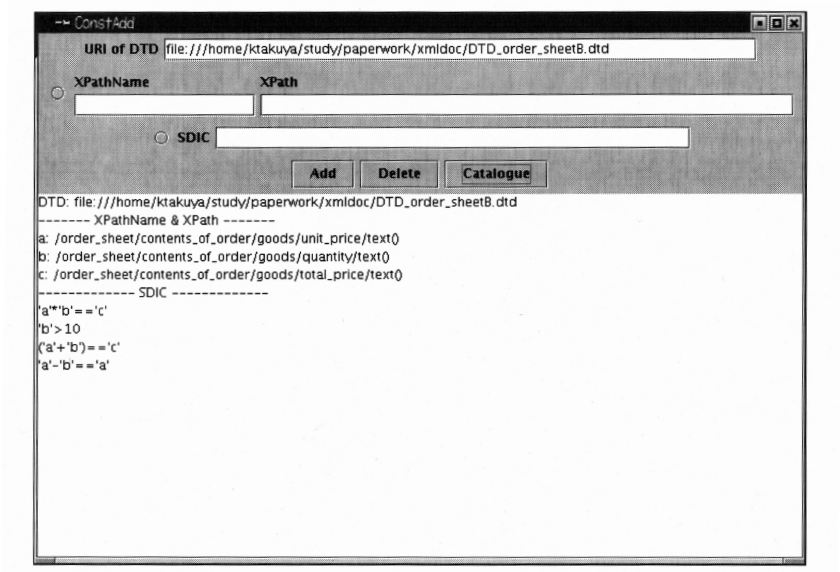


図5.1：制約条件登録GUI

図5.1の一番上の項目にDTDのURIを指定する．指定したDTDに関する操作を行うGUIである．Catalogueボタンを押すと，指定したDTDで現在使用されているXPath名XPathSDIC表現の一覧が表示される．XPath名，XPathを登録する場合は各項目を入力し，上のラジオボタンにチェックを入れ，追加するならAdd，削除するならDeleteボタンを押す．SDIC表現を登録したい場合も同様に，SDICの項目を入力し，その隣のラジオボタンにチェックを入れ，Add，Deleteで操作する．

また，XPath名を追加する場合に，追加しようとする名前が既に指定したDTDで使用されている場合はエラーとなり，エラーメッセージが表示される．

SDICを追加する場合，追加しようとするSDICに使用されているXPath名が既に登録されているかどうかのチェックも行う．登録されていないXPath名を含む場合はエラーとなり，エラーメッセージを表示する．

SDICを削除する場合，そのSDICで使用されているXPath名とそのXPath名が指定するXPathも同時に削除される．

データベースへの保存

まずデータベース作成を行う．以下の図5.1のような発注書データを保存したいとする．

コード	商　品　名	数　量	単　価	金　額	備　考
100	○○○	100	2,000	200,000	
101	△△△	1,000	300	300,000	
102	□□□	1,500	400	600,000	
103	×××	400	200	80,000	
			小　計	1,180,000	
			税　額	59,000	
			合　計	¥1,239,000	

図5.2：データの保存の例

各項目はデータの数がいくつになるか分からないが（図5.1では4つずつ），XPathはノード集合を表すので，各項目の数がいくつになったとしても，それぞれのノードを指し示すことができる．

以下のように表現すれば，各項目のノード集合を表すことができ，内容の数がいくつでもそれを指し示す．

/child::order_sheet/child::contents_of_order/child::goods/child::unit_price/child::text()

/child::order_sheet/child::contents_of_order/child::goods/child::quantity/child::text()

/child::order_sheet/child::contents_of_order/child::goods/child::total_price/child::text()

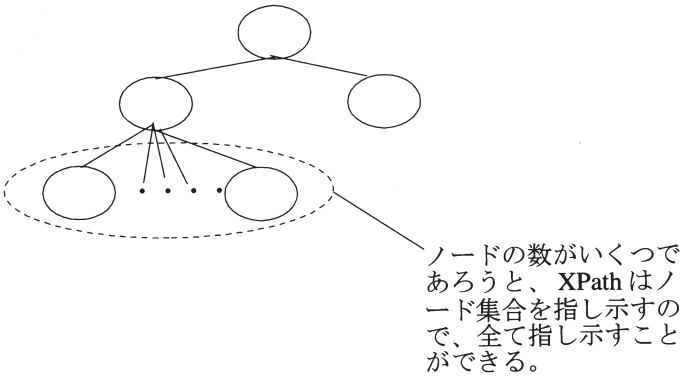


図5.3：ノード集合

逆にノード集合の中から特定のノードを指し示したいとする．その場合は以下のようにXPathを記述する．

/child::order_sheet/child::contents_of_order/child::goods[3]/child::unit_price/child::text()

上記のようにすれば，ノード集合の中から3番目に現れるノードだけを指定できる．

起動

まずサーバプログラムを起動する．

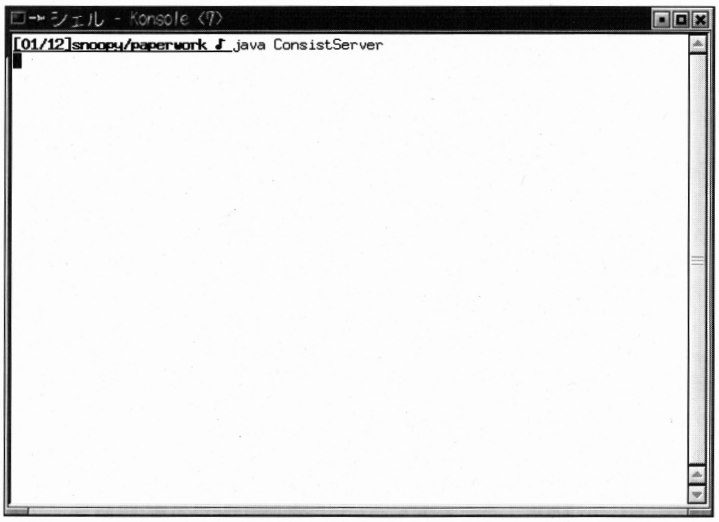


図5.4：サーバプログラムの起動

次にクライアントプログラムを実行する．

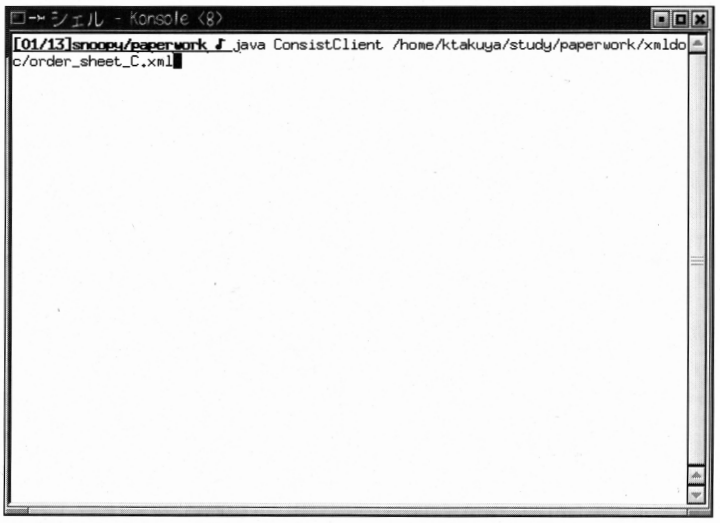


図5.5：クライアントプログラムの実行

図5.4のようにコマンドラインに「java ConsistClient dtdのuri」という形で入力する．

ここで図5.5のXML文書をサンプルとして用いる．

/child::order_sheet/child::contents_of_order/child::goods/child::unit_price/child::text()

/child::order_sheet/child::contents_of_order/child::goods/child::quantity/child::text()

/child::order_sheet/child::contents_of_order/child::goods/child::total_price/child::text()

上記のXPathに上からA,B,Cと名前をつける.

A*B==C

B>10

というSDICがあるとする.

この場合の実行結果は図5.6のようになる.

```
<?xml version="1.0" encoding="EUC-JP"?>

<!DOCTYPE order_sheet SYSTEM
"file:///home/ktakuya/study/paperwork/xmldoc/DTD_order_sheetB.dt
d">

<order_sheet>
 <contents_of_order>
  <goods>
   <name_of_goods>ocarina_t2f</name_of_goods>
   <unit_price>5000</unit_price>
   <quantity>20</quantity>
   <total_price>100000</total_price>
  </goods>

  <goods>
   <name_of_goods>ocarina_t1c</name_of_goods>
   <unit_price>4500</unit_price>
   <quantity>40</quantity>
   <total_price>180000</total_price>
  </goods>

  <goods>
   <name_of_goods>ocarina_t14c</name_of_goods>
   <unit_price>6300</unit_price>
   <quantity>15</quantity>
   <total_price>94500</total_price>
  </goods>

 </contents_of_order>

 <order_company>
  <company_name>Eitettz</company_name>
  <address> 京都 </address>
  <telephone>090xxxxzzzz</telephone>
 </order_company>

 <order_day>2004/01/01</order_day>
 <charge> 小嶋卓也 </charge>

</order_sheet>
```

図5.6：サンプルXML文書

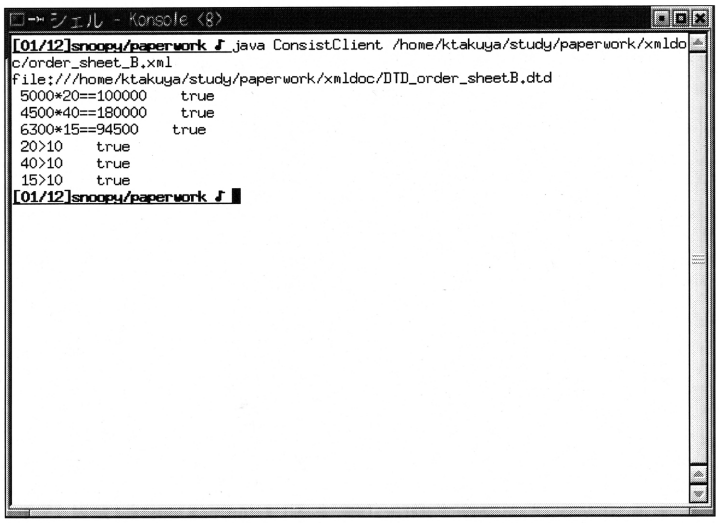


図5.7：実行結果

「SDIC表現にノードの値を代入したもの，チェック結果（true or false)」が一行づつ表示される形で画面に表示される．
図5.6の場合は，サンプルXML文書がSDICに従っているため，全てのチェック結果はtrueとなっている．

次にSDICに一部従っていない場合のXML文書のチェックを同じSDICで行う．
結果は図5.7のように表示される．

4500*20 == 100000　が
A*B==C
の制約を満たしていないため，falseと表示されている．

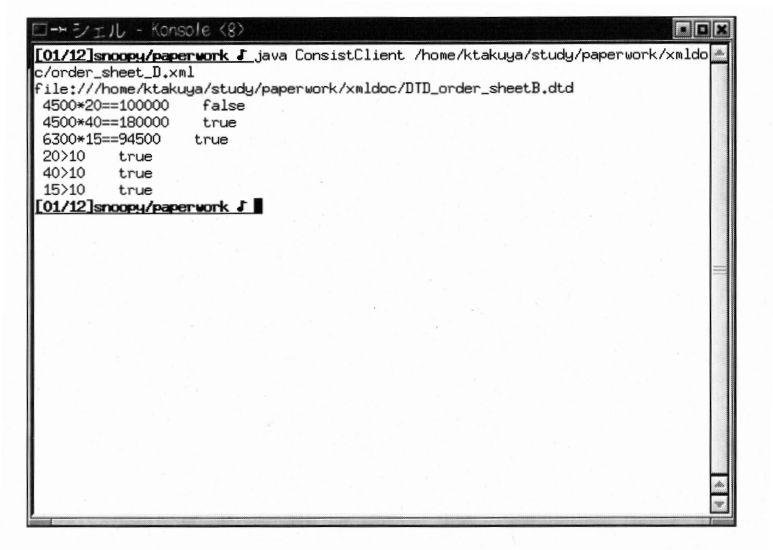


図5.8：実行結果，SDIC制約に従っていない場合．

第6章 結　　論

ネットワークを介した電子商取引，アプリケーション間のデータ交換，データの保管・管理など様々な用途でXML文書は急速に普及しつつある．それはXMLの利用者が自由にタグを定義でき，文書中の文字列に意味付けができる言語構造を持っており，プログラムで自在にXMLデータを自動処理できるというメリットのためである．しかし，実際にプログラムでXML文書を処理するには，XML文書の正当性を表現する意味的制約表現の枠組みが不可欠である．XMLの制約には一般にDTDが用いられるが，DTDの制約だけでは不十分で上手く表現できない意味的制約が存在する．たとえば，XML文書に記述するデータの型などはDTDでは表現できない．

特に，XML文書中の各部分木が示すデータに関する意味的一貫性制約を表現することができない．本研究ではXML文書の木構造中の部分木をXPathを用いて特定し，その部分木同士が満たすべき制約条件の表現する枠組みを提案した．また，それに基づくXML文書の意味的正当性をネットワーク上で一括して集中管理・検証を行うシステムの試作を行った．

今後の課題として次のようなものが考えられる．今のところ，検証結果として返されるものは，正当であるか否かを示す単純な真偽値でしかない．しかし，検証結果をより適切に表現するにはXML文書の何れの部分が正当でないのかを示すXPath式を返すべきである．また制約表現能力の評価も重要である．すなわち，本研究で提案したSDIC表現で表現可能な制約に関する詳細な検討が必要である．