社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－DBS－142 (11)
2007－ＦＩ－ 87 (11)
2007／5／31

スキーマ管理プログラム自動生成システムのＸＭＬモデルのための拡張

水野 正義　　宝珍 輝尚

京都工芸繊維大学院　工芸科学研究科

応用分野独自のデータベース管理システム(ＤＢＭＳ)を構築する際の負荷軽減を目的として，ＤＢＭＳの一部であるスキーマ管理プログラムを，ユーザからのデータモデル仕様書により自動生成するシステムの拡張について検討する．従来のシステムでは，スキーマの宣言的な操作を可能とするスキーマ管理プログラムを生成することができたが，ＸＭＬモデルの下でのスキーマに対しては，適切なスキーマ管理プログラムを生成することができなかった．そこで本研究では，ＸＭＬモデルを表す仕様書においてもスキーマ管理プログラムの生成を可能とし，さらにＸＭＬデータベースにおけるスキーマ管理に必要な，手続き的操作をサポートする関数を生成できるようにした．

An Extension of a Schema Manager Generator for XML Model

MASAYOSHI MIZUNO　TERUHISA HOCHIN

Graduate School of Science and Technology, Kyoto Institute of Technology

This paper presents an extension of the system that generates a schema manager from data model specifications. This system was constructed in order to decrease the burden of building an application specific database management system (DBMS). Conventional system can generate a schema manager that supports declarative schema control. However, it is impossible to generate a proper program for a schema under an XML model. In this study, we enable to generate a schema manager from XML model specifications. In addition, we implement functions that support procedural schema control.

１．はじめに

データベースの適応分野は急速に広がっており，様々な分野に対応した機能を持つデータベース管理システム(ＤＢＭＳ)が望まれている[1]．しかしそのようなＤＢＭＳの構築は難しく，できたとしても，ある分野にとっては不要な機能ばかりのものになる恐れがある．つまり，それぞれの分野に特化したＤＢＭＳが必要であるが，ＤＢＭＳ構築には多大なコストと時間が必要である．このことから，ＤＢＭＳの拡張性が強く求められている．

応用分野独自のＤＢＭＳ構築負荷の軽減を目的として，データモデルを表現した仕様書(データモデル仕様書)を入力することにより，ＤＢＭＳの一部であるスキーマ(データ定義情報)管理プログラムを生成するシステムが提案されている[2][3]．生成されたスキーマ管理プログラムにより，スキーマの操作を宣言的に行うことができる．ここで，スキーマがネットワーク状に構成されている場合は，手続き的にスキーマを巡航する操作のほうが使用しやすいが，このような手続き的な操作は現状では不可能である．ＸＭＬモデルの場合はスキーマがネットワーク状に構成されるため，これまでのスキーマ管理プログラム自動生成システムではＸＭＬモデルに適したスキーマ管理プログラムを生成できない．

そこで本研究では，スキーマ管理プログラム生成システムの適用範囲の拡大を目的として，従来のシステムで考慮されていなかったＸＭＬモデルへの対応を検討する．ここでは，手続き的な操作を可能とする関数群を新設し，これらの関数を必要に応じて生成するように，スキーマ管理プログラム自動生成システムを拡張する．

以降，２．で従来のスキーマ管理プログラム自動生成システムについて述べ，３．でＸＭＬモデルへの対

応について述べる．４．で実装について述べ，５．でシステムの実行例を示す．最後に６．でまとめる．

２．スキーマ管理プログラム自動生成システム

２．１　システムの構成

スキーマ管理プログラム自動生成システムは，データモデル仕様書を解析するデータモデル解析部と，解析結果をもとにソースコードを作成するスキーマ管理プログラム生成部によって構成されている[2)3)]．システムの構成を図１に示す．

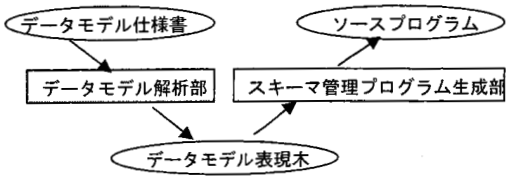

図１　自動生成システムの構成
Fig.1　Components of system.

図１に示すように，システムはデータモデル仕様書ファイルを入力とし，入力された仕様書ファイルのデータ構造がデータモデル解析部において解析され，データモデル表現木が主記憶上に作成される．このデータモデル表現木をもとに，スキーマ管理プログラム生成部が，スキーマの管理を行う関数を出力する．

データモデル解析部はｌｅｘ／ｙａｃｃを用いて実装され，スキーマ管理プログラム生成部はＣ言語を用いて実装されている．

２．２　データモデル仕様書

データモデル仕様書では，使用するデータ型の指定と，データモデルの構造を一種の文法として記述する．使用するデータ型は，最初の「%」の前に「datatype=」に続いて指定する．データモデルの構造は，一行を一つの定義として，最初の「%」と最後の「%」の間に記述する．ただし，「:」，「#」は左辺がデータモデルの主要素である場合に使用し，「=」はその他の要素に使用する．「<>」で囲まれる要素は，別の行で定義されている要素である．右辺には，データ型とその大きさを「[]」で書くか，「<>」で囲まれた要素を用いて表す．ＯＲを表現したいときは「{}」で要素を囲み，要素を区切るのに「|」を用いる．「+」は一回以上の繰り返しを表し，「!」はインスタンスが集合であることを表す．これらは，ともに対象要素の右に記述する．データモデル仕様書の例を図２に示す．

```
datatype = Int, Char
%
name = char[16]
datatype = int
Column = <name>{<datatype>|<Table>}
Table：<name><Column>+
%
```

図２　データモデル仕様書の例
Fig.2　An example of a data model specification.

図２に示したデータモデル仕様書では，以下を規定している．

- データ型として整数型と文字型を用いる（１行目）．
- 名前とは，１６文字の文字列である（３行目）．
- データ値のデータ型は，整数型で管理する（４行目）．
- データモデルの構成要素は，テーブルとカラムである（５，６行目）．
- カラムは，名前と整数型もしくは文字型のデータ値，あるいはテーブルからなる（５行目）．
- テーブルは，名前と一つ以上のカラムからなる（６行目）．

２．３　データモデル表現木

データモデル表現木とは，入力されたデータモデル仕様書を木構造で表現したものであり，主記憶上に生成される．この表現木に付加された情報により，要素の親子関係の参照が容易になる．作成された表現木をもとに，スキーマ管理プログラムが生成される．

２．４　生成されるスキーマ管理関数

スキーマ管理プログラム生成部では，以下の条件を満たす要素に対応する構造体に対して，生成・検索・削除を行う関数が生成される．

1. 繰り返しが生じる要素
2. データ表現の基本となる要素

データモデル仕様書で「:」，「#」や「+」を用いた要素が構造体作成の対象となる．図２のデータモデルの場合，Table や Column である．Table は「:」で定義され，Column は Table において繰り返し出現する．

この他に，要素の整合性を検査する関数なども生成される．スキーマ管理プログラム生成部で生成されるスキーマ管理用関数を表１に示す．ここで，「*」はデータモデル仕様書で定義された要素の名前である．

表 1　スキーマ管理用関数
Table.1　Schema management functions.

関数名	機能
create_*	要素*を定義する
select_*	*の定義情報を参照する
drop_*	*の定義情報を削除する
check_*	*の整合性を確認する

図 2 の仕様書から生成される関数群を，以下に示す．

- int create_Table(int Table_no, char *name)
- int create_Column(int Table_no, int Column_no, char *name, or_inf *oinf)
- or_inf *create_or(int data_type, void *val, int length, int null, int instance_min, int instance_max)
- sysTable *select_Table(int Table_no)
- sysColumn *select_Column(int Table_no, int Column_no)
- int drop_Table(int Table_no)
- int drop_Column(int Table_no, int Column_no)
- int check_Table()
- int check_Column()

生成されたスキーマ管理用関数を用いて作成するスキーマは，図 3 に示すような，リンクを用いたネットワーク構造で表現される．

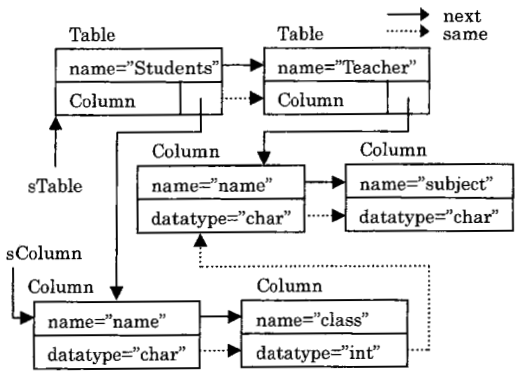

図 3　主記憶上に作られるスキーマ情報
Fig.3　Structure of a schema.

スキーマの操作はリンクをたどるという手続き的な操作ではなく，要素の番号を指定して行う宣言的な操作である．例えば，select_Column では，テーブル番号(Table_no)とカラム番号(Column_no)を入力に指定して，Column のための構造体(sysColumn)へのポインタが出力として返却される．Table のための構造体をたどり，そこから Column のための構造体をたどって，該当する Column の情報を求めるという操作を行うのではない．つまり，生成されたスキーマ管理用関数を用いたスキーマ操作は宣言的な操作である．

3．設計

3．1　XMLモデル

XML(extensible markup language)は，データのみでなく，データの構造もタグを用いて記述するマークアップ言語である [4]．XMLで記述した文書は，開始タグと終了タグの間にテキストデータを挟み込んだ要素(エレメント)からなる．要素同士は入れ子にすることができるので，階層化したデータ構造を持つことができる．XML文書の例を図 4 に示す．

```
<?xml version="1.0" encoding="shift_jis" ?>
<books>
  <book year="1997" isbn="01234567">
    <title>タイトル</title.>
    <author>著者</author>
    <publisher>A社</publisher>
  </book>
</books>
```

図 4　XML文書の例
Fig.4　An example of an XML document.

XML文書は，要素と属性が複数集まって表現され，その構造は木構造をなす．XML文書には以下に示す特徴がある．

- 文書は一つの独立したルート要素でくくられる
- データは全て文字列である
- 要素は子に，入れ子となる要素，あるいはデータ値を 0 個以上持てる
- 要素は属性を 0 個以上持てる

XMLモデルは，XML文書をそのまま木構造の形式で格納可能にするためのモデルである．

3．2　システムの要件

3．1 で述べたXMLモデルを表現するのに，図 5 のようなデータモデル仕様書が考えられる．XMLモデルに特徴的な，自分自身を親に持つ要素を含む仕様となっている．データは文字列なので，一行目のデータタイプ制約は文字型のみを宣言する．「0 個以上持つ」ということは，「省略可能」と「繰り返し」を組み合わせることで表現する．

```
datatype = Char
%
name = char[16]
datatype = int
Attribute = <name><datatype>
Node = {<datatype>|<Element>}
Element = <name>(<Attribute>+)(<Node>+)
Root : <name><Element>+
%
```

図 5　XMLモデル仕様書
Fig.5　An XML model specification.

図5に示したデータモデル仕様書には，非独立要素(Node と Element)が自分自身を親に持つという大きな特徴がある．独立要素が自分自身を親に持つという特徴は，オブジェクト指向モデルにもある．この場合，独立要素はこれまでの方式で生成されたスキーマ管理用関数でも直接操作できるので，特に問題にはならなかった．しかし，XMLモデルの場合，非独立要素はこれまでの関数では直接操作できない．すなわち，独立要素から入れ子をたどって目的の非独立要素を求める必要がある．このためには，操作を開始する独立要素から目的の非独立要素に至る全ての非独立要素を指定する必要がある．しかし，入れ子の段数に制限がないため，このようなことを可能とする関数を作成したとしても，使用するのが非常に困難ではないかと考えられる．これよりも，スキーマを表現するリンク構造を巡航するような関数の方が使用するのが容易であろうと考えられる．そこで本研究では，このような手続き的にスキーマに対する操作を行う関数を生成することとする．

3．3　手続き的操作のための関数

生成される関数は，それらのみでリンク構造を全てたどれ，かつ，わかりやすい機能を持つ必要がある．ここでは以下の関数を生成することとした．

- 次(next ポインタ)をたどれる関数
- 子をたどれる関数
- 親をたどれる関数

さらに，通常生成される select 関数は，要素を番号で参照するという非常に宣言的な機能を持つので，限られた範囲で小規模な探索を実施する関数も生成する．

新たに生成する関数をまとめて表2に示す．「*」はデータモデル仕様書で定義された要素の名前である．

表 2　手続き的操作のためのスキーマ管理用関数
Table.2　Schema management functions for procedural schema control.

関数名	機能
navigateNext_*	指定した*の次があればそれを返す
navigateChild_*	指定した要素の子に*があればそれを返す
navigateParent_*	指定した要素の親に*があればそれを返す
smallselect_*	*の一段下の*まで，指定した要素を探す

また，ここでは，スキーマ操作も宣言的にしてデータ独立性を高めるべきであると考え，表2で示した関数は，手続き的操作が必要なデータモデルの場合のみ生成することとする．すなわち，宣言的な操作のみでよい場合は従来のスキーマ管理用関数のみを生成し，手続き的な操作が必要な場合は表2で示した関数も生成する．

表 2 で示した関数を生成する条件は，「非独立要素が自分自身を親とする場合」である．

4．実装

4．1　スキーマ管理プログラムの追加・変更

スキーマ管理プログラム自動生成システムの拡張は，まず，生成するスキーマ管理プログラムに追加・変更を加え，その後に自動生成システムに変更を加える，という手順で行う．

以下では，生成される関数のうち，新たに追加する関数である「navigateNext」，「navigateChild」，「navigateParent」，「smallselect」，ならびに，再検討を行った「create」，「select」，及び「drop」について説明する．

4．1．1　次をたどる関数「navigateNext」

引数で渡した要素の次，すなわち next ポインタが指す情報を参照する関数である．引数には要素を表す構造体のポインタを指定する．

4．1．2　子をたどる関数「navigateChild」

引数で渡した要素の子を参照する関数である．子を指すポインタが用意されているので，それを返す．

4．1．3　親をたどる関数「navigateParent」

引数で渡した要素の親を参照する関数である．親を指すポインタは存在しないため，独立要素からリンクを全てたどって，参照したい要素を特定する．

XML モデル仕様書から作成されるスキーマは，Element が子に Node を持ち，Node が子に Element

を持つという，深さが一定でない木構造である．そのため，再帰呼び出しによって木構造を深さ優先で探索するアルゴリズムを用いる．

4．1．4　小規模なスキーマ参照関数「smallselect」

引数で渡した要素の一段下の同じ要素まで，スキーマ情報を検索する関数である．引数には，検索の始まりとなる構造体のポインタと，参照したい要素の番号を指定する．

4．1．5　スキーマ作成関数「create」

スキーマ情報を作る関数である．malloc によって動的に構造体が作成され，初期値が設定される．従来の機能のとおり，作成する要素の親となる要素を参照し，見つかればその子を作る．大きな変更点は，関数内部を簡潔にするために，親となる要素を参照するのに select 関数を用いることにした点である．

4．1．6　スキーマ参照関数「select」

作成されたスキーマ情報を参照する関数である．生成部での都合上，引数には独立要素の番号と参照したい要素の番号のみを指定することとした．これにより，一つの独立要素下においてスキーマ情報の番号は重複してはならない仕様となる．

navigateParent と同様に，再帰呼び出しによって木構造を深さ優先で探索するアルゴリズムを用いる．

4．1．7　スキーマ削除関数「drop」

作成されたスキーマ情報を削除する関数である．指定した要素の下に繰り返しのある要素がある場合，先にそれらの要素を削除しなければ削除することはできない仕様である．create 関数同様，親となる要素を参照するのに select 関数を用いる．

4．2　スキーマ管理プログラム生成部の変更

スキーマ管理プログラム自動生成システムは，データモデル解析部とスキーマ管理プログラム生成部からなる．スキーマ管理プログラム生成部は表 3 に示す 7 つのプログラムからなり，それぞれの役割が明確に分かれている．

以下では，再検討を行った「struct」，「create」，「select」，ならびに，「drop」について説明する．

4．2．1　スキーマの構造体生成部「struct」

ヘッダファイルに宣言するクラスや，そのクラスのエントリポインタを保持する大局変数などを作成するプログラムである．

従来のものからの変更点は，select 関数や drop 関数で再帰呼び出しを行う際に用いる大局変数の宣言を新たに出力するようにした点である．スタックを利用して要素の自己参照を判定し，再帰呼び出しが行われるかどうかを確かめる．自己参照の判定は，新たに追加した以下の関数によって行う．

- stac_parent()：指定した要素の親を全てスタックに積む．
- loop_check()：stac_parent 関数で作成したスタックをもとに，自己参照を判定する．

表 3　スキーマ管理プログラム生成部の構成
Table.3　Components of a schema manager generator.

名称	役割
stock	データモデル表現木への情報付加
struct	スキーマデータを保持する構造体(クラス)生成
create	スキーマ作成関数生成
select	スキーマ参照関数生成
drop	スキーマ削除関数生成
check	スキーマ整合性確認関数生成
readwrite	スキーマの二次記憶化に関する関数生成

4．2．2　スキーマ作成関数「create」生成部

create 関数を生成するプログラムである．従来のものに，新たに以下の関数を追加した．

- make_create_loopnumber()：4．1．5で述べた，create 関数の引数を出力する．
- make_create_loopmain()：create 関数の本体を出力する．

構造体生成部「struct」と同様に，要素の自己参照を判定し，その結果によって処理を分岐する．自己参照があれば make_create_loopnumber 関数及び make_create_loopmain 関数を処理する．

4．2．3　スキーマ参照関数「select」生成部

select 関数を生成するプログラムである．create 関数及び drop 関数内においても select 関数が実行されるため，スキーマ管理プログラム生成部の中では重要な部分である．

従来のものに，新たに以下の関数を追加した．要素の自己参照があれば，これらを処理する．

- make_select_loopmain()：4．1．6で述べた，再帰呼び出しによって指定した要素を参照する select 関数を出力する．
- make_navigate()：手続き的操作のための関数を出力する．

3．3で示した手続き的操作のための関数は，性質としては要素を参照する関数と言え，select 関数と類似している．このことから，手続き的操作のための関数も，この select 関数生成部で生成することとする．

この機能は，make_select_loopmain 関数の処理終了後，make_navigate 関数によって行う．

４．２．４　スキーマ削除関数「drop」生成部

drop 関数を生成するプログラムである．従来のものに新たに以下の関数を追加した．

- make_drop_loopmain()：４．１．７で述べた drop 関数を出力する．

要素の自己参照があればこの関数を処理する．

５．実行例

５．１　生成されるスキーマ管理用関数

実際にＸＭＬモデル仕様書からソースプログラムを生成し，そのプログラムを用いてスキーマ管理を行う例を述べる．図５のＸＭＬモデル仕様書から生成されるスキーマ管理用関数を以下に示す．

- int create_Root(int Root_no, char *name)
- int create_Root_Element(int Root_no, int Element_no, char *name)
- int create_Node_Element(int Root_no, int Parent_no, int Element_no, char *name)
- int create_Node(int Root_no, int Element_no, int Node_no, or_inf *oinf)
- int create_Attribute(int Root_no, int Element_no, int Attribute_no, char *name, int datatype, int datatype_length, int datatype_null)
- or_inf *create_or(int data_type, void *val, int length, int null, int instance_min, int instance_max)
- sysRoot *select_Root(int Root_no)
- sysElement *select_Element(int Root_no, int Element_no)
- sysNode *select_Node(int Root_no, int Node_no)
- sysAttribute *select_Attribute(int Root_no, int Attribute_no)
- sysElement *navigateNext_Element(sysElement *Elementp)
- sysNode *navigateNext_Node(sysNode *Nodep)
- sysAttribute *navigateNext_Attribute(sysAttribute *Attributep)
- sysElement *navigateChild_Root_Element(sysRoot *Rootp)
- sysElement *navigateChild_Node_Element(sysNode *Nodep)
- sysAttribute *navigateChild_Attribute(sysElement *Elementp)
- sysNode *navigateChild_Node(sysElement *Elementp)
- sysRoot *navigateParent_Root(sysElement *Childp)
- sysNode *navigateParent_Node(sysElement *Childp)
- sysElement *navigateParent_Element(sysAttribute *Childp)
- sysElement *navigateParent_Element(sysNode *Childp)
- sysElement *smallselect_Element(sysElement *Parentp, int Element_no)
- sysNode *smallselect_Node(sysNode *Parentp, int Node_no)
- int drop_Root(int Root_no)
- int drop_Element(int Element_no)
- int drop_Node(int Node_no)
- int drop_Attribute(int Attribute_no)
- int check_Root()
- int check_Element()
- int check_Node()
- int check_Attribute()

５．２　スキーマ管理の例

スキーマを管理したいＸＭＬデータベースの例として，図４のような本のタイトル，著者，出版社を表すＸＭＬ文書について考える．なお，ＸＭＬ文書の一行目(<?xml version ...)については処理対象外とする．

以下のような条件を満たすスキーマが必要である．

- ルート要素は books である
- books の子に，属性を二つ持つ book 要素がある
- book 要素の子に，title，author，及び publisher 要素がある

スキーマ情報は親から作成しなければならないので，以下のようにして“books”という名前の Root 要素を定義する．

- create_Root(1,"books")

次に Root 要素を親に持つ“book”という名前の

Element 要素を定義する．“books” を親に持つことを示すために，Root 要素を定義した際に指定した番号(ここでは 1)を引数に指定する．

- create_Root_Element(1,1,”book”)

“book” が持つ “year”，“isbn” 属性は，create_Attribute によって，例えば以下のように定義する．

- create_Attribute(1,1,1,”year”,3,16,0)
- create_Attribute(1,1,2,”isbn”,3,16,0)

Element の子に Element を作成するためには，まず Node 要素が必要である．Node は子に Element もしくはデータ値を持つ．そのためにＯＲ要素を用いるので，create_or でＯＲを使えるようにする．以下では，データ型が文字列(番号 3)であり，長さが 16 であると定義している．

- oinf = create_or(3,NULL,16,1,0,0)

“book” の子には “title”，“author”，“publisher” があるので，それぞれの子を持つための Node，そしてデータ値を持つための Node を作成する．

- create_Node(1,1,1,oinf)
- create_Node_Element(1,1,2,”title”)
- create_Node(1,2,2,oinf)
- create_Node(1,1,3,oinf)
- create_Node_Element(1,2,3,”author”)
- create_Node(1,3,4,oinf)
- create_Node(1,1,5,oinf)
- create_Node_Element(1,3,4,”publisher”)
- create_Node(1,4,6,oinf)

以下では，スキーマの参照及び削除を行う．まずルート要素を参照し，そこから子をたどり，さらにその子をたどっている．そしてその次の要素をたどり，最後に Attribute を削除している．ここでは，“isbn” という名前をもつ Attribute を削除している．

- sr = select_Root(1)
- se = navigateChild_Root_Element(sr)
- sa = navigateChild_Attribute(se)
- sa = navigateNext_Attribute(sa)
- r = drop_Attribute(1,sa->Attribute_no)

６．おわりに

データモデルはデータベースにおいて大変重要なものであり，それぞれの応用分野に応じて適切なデータモデルが異なることが考えられる．そこで，利用者が定義したデータモデルによる DBMS の構築を容易にするために，スキーマ管理プログラムを自動生成するシステムが作成されてきた．しかし，従来のシステムでは，XML モデルを表現した仕様書からは，スキーマ管理プログラムを生成することができなかった．そこで本稿では，スキーマ管理プログラム生成システムの適用範囲の拡大を目的として，XML モデル仕様書からのスキーマ管理プログラムの生成を可能とした．まず，XML データベースにおけるスキーマ管理に必要な，子をたどったり，親を参照したりという，手続き的操作をサポートする関数を生成できるようにした．このために，スタックを利用して XML の特徴である要素の自己参照を判定し，適正な関数を出力する．以上により，XML モデルにおけるスキーマ管理が可能となり，従来のシステムよりも多くのデータモデルを定義できるようになった．

本システムは，例で上げた程度のデータモデルでしか検証していない．同じモデルを表現する場合でも，ユーザによって異なる仕様書が書かれる場合もある．多数のユーザに本システムを使用してもらうなどして，あらゆるデータモデルに適応できるかを検証する必要がある．これは今後の課題である．

参 考 文 献

1) M.Stonebraker, U.Cetintemel : “One Size Fits All”: An Idea Whose time Has Come and Gone, Proceedings of the 21st International Conference on Data Engineering (ICDE 2005), pp.2-11 (2005)
2) 平林孝之，宝珍輝尚，都司達夫：データモデルに応じたスキーマ管理部の一生成法，第５８回情報処理学会全国大会 1V-6 (1999)
3) 松浦雅彦，宝珍輝尚，都司達夫：データベース言語処理システムの自動生成に関する一検討，情報処理学会データベースシステム研究会 124-12，pp.89-96 (2001)
4) D.Livingston : Essential XML for Web Professionals, Prentice Hall (2002)