SLIS Technical Report, University of Tsukuba
筑波大学図書館情報メディア系
テクニカルレポート

C-Mapping: 関数従属性と包含従属性を用いた柔軟な XML-RDB マッピング手法

C-Mapping: A Flexible XML-RDB Mapping Method based on Functional and Inclusion Dependencies

太田 壮祐, 森嶋 厚行*, 天笠 俊之**, 品川 徳秀*
筑波大学大学院図書館情報メディア研究科
*筑波大学図書館情報メディア系
**筑波大学システム情報系
Sousuke Ohta, Atsuyuki Morishima*, Toshiyuki Amagasa**, Norihide Shinagawa*
Graduate School of Library, Information and Media Studies, University of Tsukuba
*Faculty of Library, Information and Media Science, University of Tsukuba
**Faculty of Engineering, Information and Systems, University of Tsukuba

SLIS-TR-2012-001

Abstract

これまで，多くの XML-RDB マッピング手法が提案されてきたが，それらは全て XML 要素から RDB 属性値への 1:1 もしくは 1:N マッピングのどちらかを扱うものであった．本稿では，1:1，1:N マッピングだけでなく N:1 マッピングも扱う事が可能な XML-RDB マッピング手法 C-Mapping を提案する．C-Mapping は，入力として XML データとそこに存在する関数従属性と包含従属性を受け取り，出力として XML データを格納するためのリレーションとその上での XML ビュー定義を生成する．C-Mapping は対象 XML データにおける従属性を反映したマッピングを行うため，RDBMS の機能を最大限に活用して XML ビューでのデータ一貫性を維持することができる．本稿は C-Mapping の説明に加え，C-Mapping がこれまで提案されてきた主要な手法をシミュレートすることが可能である事を示す．さらに，実データを用いた検証により，C-Mapping が提供する高い表現力の有効性を示す．

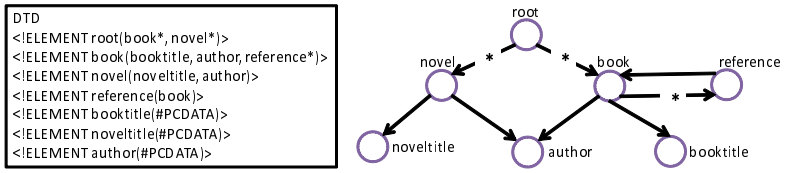

Figure 1: book.dtd(左) とその DTD グラフ (右)

1 はじめに

これまで，XML データを RDB で管理するために，XML データを RDB にマップする手法が数多く研究されてきた．これらのマッピングのアプローチの主要なものとして，構造写像アプローチ [1] やモデル写像アプローチ [2][3]，制約ベースアプローチ [4][5] が存在する．これらの既存の手法は全て異なるアプローチを採用しているが，XML 要素 (あるいは属性) から RDB 属性値への 1:1 マッピングもしくは 1:N マッピングを行うという共通点が存在する．例えば，構造写像アプローチの一つである Basic-Inlining を用いて，book.dtd (図 1) に従う book.xml をマップすると，`book` リレーションと `novel` リレーションが生成される (図 2)．これを見ると，book.xml の各 booktitle 要素は `book` リレーションの一つの `booktitle` 属性値に 1:1 対応していることが分かる．このように，1 つの XML 要素 (あるいは属性) を 1 つの RDB 属性値に対応させるマッピングを，1:1 マッピングと呼ぶ．また，1 つの XML 要素 (あるいは属性) を複数の RDB 属性値に対応させるマッピングも存在し，これを 1:N マッピングと呼ぶ．我々の知る限り，既存のマッピング手法は全て 1:1 もしくは 1:N マッピングを行うものである．

しかし，これらの 1:1/1:N マッピングでは，生成されたリレーションを更新する際に XML ビュー上でのデータ一貫性制約の維持が保証されないという問題がある．この問題について，図 2 の例を用いて説明する．この例では，book.xml が「noveltitle 要素に含まれるテキストは，必ず booktitle 要素のテキストとして存在する」という一貫性制約を持つと仮定する．図 2 のマッピングでは，もし `book` リレーションの `booktitle` 属性の “Sherlock Homes” という値のみを更新してしまうと，XML ビュー上で更新不整合が起きる可能性がある．

本論文では，XML-RDB マッピングにおける一貫性制約の維持を実現するために，制約ベースアプローチを採用した新たなマッピング手法である **C-Mapping** (Consistency-conscious Mapping) を提案する．C-Mapping は，入力として XML データとその XML データに存在する一貫性制約 (関数従属性と包含従属性) を指定する事により，RDB へのマッピング結果としてリレーションの集合を出力する．C-Mapping の特徴は，1:1/1:N マッピングだけでなく N:1 マッピングも実現可能であるという意味で，**完全**であるという事である．さらに，C-Mapping は入力として適切な一貫性制約を与える事により，既存の主要なマッピング手法の多くをシミュレートできる．

ここで，例を用いて N:1 マッピングを説明する．N:1 マッピングとは，複数の XML 要素 (あるいは属性) を 1 つの RDB 属性値にマップすることである．book.xml の N:1 マッピングの結果の例を図 3 に示す．この例では，先程と同様に，book.xml が「noveltitle 要素に含まれるテキストは必ず booktitle 要素のテキストとして存在する」という一貫性制約を持つと仮定する．図 3 に見られるように，各 noveltitle 要素とそれに対応する booktitle 要素は，`book` リレーションの中で同じ属性値にマップされている．したがって，このマッピングでは，マッピング結果のリレーションで更新が行われる際，XML ビュー上の一貫性制約が破られない事が保証される．このように N:1 マッピングを行う事で，XML データの更新時における一貫性維持が容易になる．

本論文の構成は次の通りである．2 章では関連研究について述べる．3 章では C-Mapping で扱う XML の一貫性制約を説明する．4 章では C-Mapping のアルゴリズムを説明する．5 章では C-Mapping の記述力について議論し，既存の主要なマッピング手法の多くをシミュレート可能であることを証明する．6 章では実データを用いた C-Mapping の評価とその結果を示す．7 章にまとめを述べる．

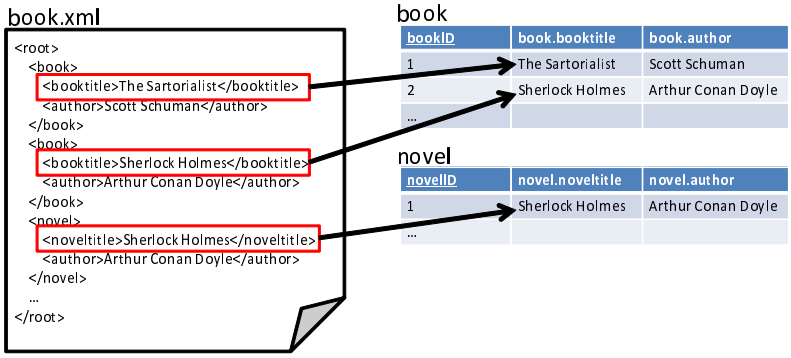

Figure 2: 1:1 マッピングの例

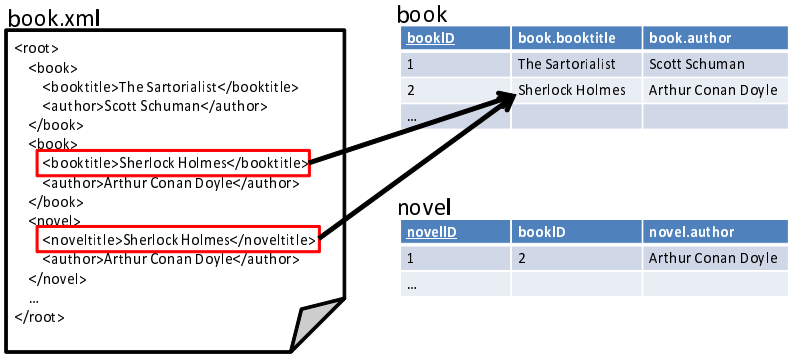

Figure 3: N:1 マッピングの例

2 関連研究

既存の XML-RDB マッピング手法は，(1) 構造写像アプローチ，(2) モデル写像アプローチ，(3) 制約ベースアプローチの 3 つに分類することができるが，我々の知る限り，既存の XML-RDB マッピング手法は全て 1:1 マッピングもしくは 1:N マッピングである．5 章で証明するように，本論文で提案する C-Mapping は下記のマッピング手法を全てシミュレートする事ができる．

(1) 構造写像アプローチでは，XML データを格納するためのリレーショナルスキーマは，個別の XML データの DTD によって決まる．構造写像アプローチを用いる主な手法としては，basic-inlining や shared-inlining，hybrid-inlining [1] がある．

(2) モデル写像アプローチでは，構造写像アプローチとは異なり，XML データを格納するためのリレーショナルスキーマは，DTD ではなく XML データモデル [7] によって決まる．したがって，生成されるリレーショナルスキーマは個別の XML データの DTD に依存しない固定のリレーショナルスキーマとなり，任意の整形式 XML データを格納できる．モデル写像アプローチを用いる主な手法としては，XRel [2] や枝アプローチ [3] がある．

(3) 制約ベースアプローチでは，XML データに存在する一貫性制約に基づいて XML データをリレーションにマップする．制約ベースアプローチを用いる主な手法としては，RRXS [4] や X2R [5]，我々が提案する C-Mapping がある．RRXS は，入力として XML データに存在する関数従属性を指定する事により，その関数従属性が維持されるリレーショナルスキーマを生成する[1]．X2R は，XML スキーマ [6] における key/keyref 制約が破られないリレーショナルスキーマを生成する．key/keyref 制約は包含従属性の一種であり，N:1 マッピングを行う事無く XML データの包含従属性の維持が可能である．したがって，X2R は我々が提案する

[1]RRXS では入力された関数従属性の集合をそのまま使わず，最初に XML 要素のノード ID を表す変数を除去して極小被覆を計算するが，これはマッピングの前処理と見なすことができる．

仮想的な属性	概要
n/@nodeName()	ノード n の名前
n/@path()	ノード n の絶対パス
n/@pre()	ノード n の開始バイトオフセット値
n/@post()	ノード n の終了バイトオフセット値
n/@edges()	ノード n を始点とするエッジ集合
n/@type	ノード n の型
n/@nodeID()	ノード n の ID (ID 型の属性値)
n/@pathID()	ノード n の絶対パスの ID
e/@parentNode()	エッジ e の始点
e/@childNode()	エッジ e の終点
x/@docID	XML データ x の ID

Table 1: 仮想的な属性の例

C-Mapping で対処する問題を部分的に扱っていると言える．

3 XML の一貫性制約

本章では，C-Mapping で用いる，XML データの 3 つの一貫性制約 (XML functional dependencies (XFD) [4], XFD+, XML inclusion dependencies (XIND)) について説明する．

3.1 XML Functional Dependencies

XML Functional Dependencies (XFD) は論文 [4] で定義された XML における関数従属性である．

一般には，関数従属性はリレーショナルデータにおける次のようなデータ一貫性制約として良く知られている．$R(\ldots, A, \ldots, B, \ldots)$ というリレーションスキーマがある時，関数従属性 $A \rightarrow B$ とは，R における任意の 2 つのタプル t と t' に関して，もし $t[A] = t'[A]$ ならば必ず $t[B] = t'[B]$ であるという制約を表す．この時，A を決定子，B を被決定子と呼ぶ．

XFD は，XML インスタンス木 (以下 XML 木) において上記と同じ考えを適用したものである．XML 木は XML データのインスタンスの階層構造を表した木である．XML 木のノード (以下 XML ノード) は XML の要素，属性，テキストに対応し，エッジは XML ノード間の包含関係に対応している．XFD は，リレーションの属性間の制約ではなく，XML パス式で指定された XML ノード間の制約である．具体例として，図 2 に示す book.xml における XFD の例を説明する．この例では，book.xml に「各 book 要素が決まれば，その book 要素に含まれる booktitle 要素の内容であるテキストが決まる」という制約が存在すると仮定する．この時，その制約は XFD で次のように記述する．

```
for $x in book.xml/root/book      ··· (1)
   $x →$x/booktitle/text()        ··· (2)
```

(1) は，XML パス式 “book.xml/root/book” によって特定したノード集合の中から一つずつノードを取り出し変数 $\$x$ に束縛することを表している．また，(2) では，$\$x$ に束縛されたノードが決まれば XML パス式 “$\$x$/booktitle/text()” によって特定されるテキストが一意に決まる，という関数従属性が存在することを記述している．

3.2 XFD+

通常，XFD の決定子と被決定子は，対象となる XML 木に存在する XML ノード (要素，属性，テキスト) である．本論文では，決定子と被決定子に計算で求められる仮想的な属性の使用を許可した XFD+と呼ばれる制約のクラスを導入する．仮想的な属性は，接頭辞 “@” と関数形式 (関数名+“()”) で指定する．

表 1 に仮想的な属性の例を示す．例えば，book.xml(図 2) において，ある要素の絶対パスが決まればその要素名が一意に決まる，という制約は次のように表現できる．

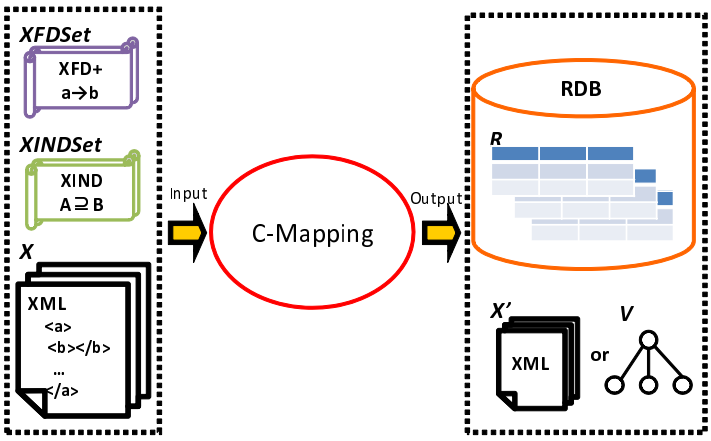

Figure 4: C-Mapping の入出力

```
for $x in book.xml//*
    $x/@path() → $x/@nodeName()
```

3.3 XML Inclusion Dependencies

XML Inclusion Dependencies (XIND) は，XML における包含従属性である．包含従属性も一般にリレーショナルデータにおけるデータ一貫性制約として良く知られている．$R(\ldots, A, \ldots)$ と $S(\ldots, B, \ldots)$ という 2 つのリレーションスキーマがある時，包含従属性 $R[A] \supseteq S[B]$ とは，S の属性 B の全ての値は R の属性 A の値として存在しなければならないという制約を表す．ここで，$R[A]$ は R の属性 A による射影を意味している．

XIND は，XML 要素が持つテキストあるいは属性値の間の包含従属性を表したものであり，2 つの XML パス式を用いて表記する[2]．具体例として，図 2 に示す book.xml における XIND の例を説明する．この book.xml に「noveltitle 要素が持つ全てのテキストは booktitle 要素のテキストとして存在していなければならない」という制約が存在すると仮定すると，その制約は次の XIND で表す事ができる．

```
book.xml/root/book/booktitle/text()
  ⊇ book.xml/root/novel/noveltitle/text()
```

本論文で扱う XIND は，論文 [10] で定義している包含従属性と本質的に同じである．上記例では単項リレーション間の包含従属性を表現しているが，一般には n 項リレーション間の包含従属性を表す．

4 C-Mapping

4.1 概要

本章では，提案する XML-RDB マッピング手法 C-Mapping (Consistency-conscious Mapping) について説明する．C-Mapping の入出力を図 4 に示す．入力は，XML データの集合 X，および X に関する XFD+の集合 $XFDSet$，X に関する XIND の集合 $XINDSet$ である．この入力は次の 2 つの制約を持つ．(1) $XINDSet$ 中の XIND の左右各辺の XML パス式で参照される XML ノード集合は全て，$XFDSet$ 中のいずれかの XFD+における XML パス式によって参照されていなければならない．さらに，XIND の各辺から参照される XML ノード集合は，それぞれ同一の XFD+に含まれる XML パス式で参照されていなければならない．この理由は，後述するマッピング手順により，XIND の各辺がそれぞれ一つのリレーションにマップされることを保証するためである．(2) あるノードが XFD+によって参照されている場合，そのノード

[2] ここでは自明な document() 関数の記述は省略する

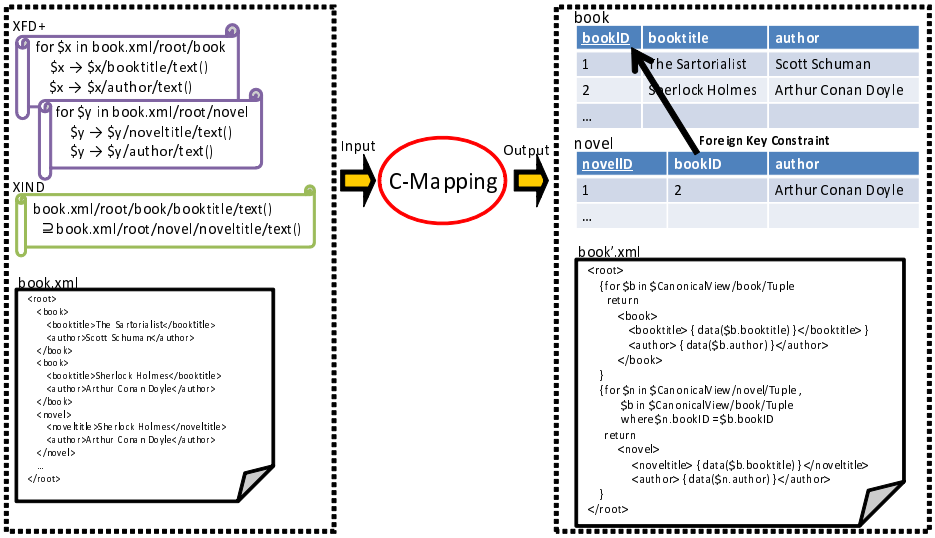

Figure 5: 図 3 のマッピングを行う入出力

を根とする部分木の全てのノードもいずれかの XFD+において参照されていなければならない．この理由は，次に説明する XML テンプレートの生成時に，リレーションから生成される XML 木が全体の XML 木の部分木となることを保証するためである．

C-Mapping の出力は，X のマッピング結果であるリレーションの集合 R と，XML テンプレートの集合 X' である．X' が必要な理由は，C-Mapping は後述するように入力 $XFDSet$ によって参照される XML 要素のデータのみを含むリレーションを生成するため，参照されない XML データを保持する必要があるからである．

X' 中の各 XML テンプレートは次のように作成される．まず，元の XML 木から RDB に格納される部分木を削除する．次に，削除された部分木の箇所に，元の XML 木を RDB データから構築するための問合せを挿入する．この問合せは，RXL [11] によって記述される．RXL は SQL と XQuery 風の言語を組み合わせた構文を持つ．リレーションから XML ビューを構築する言語であり，データの挿入や削除に対して柔軟に対応できる．この構築方法に関しては 4.4 節で説明する．

C-Mapping の入出力の例として，図 3 のマッピングを行うための入出力を図 5 に示す．入力された XIND が，N:1 マッピングとして反映されていることがわかる．

C-Mapping は次の 2 つのフェーズから構成される．

(フェーズ 1) $XFDSet$ と $XINDSet$ を用いて，リレーショナルスキーマ RS を生成する．

(フェーズ 2) X を用いて，RS に従うリレーションインスタンスの集合 I と，XML テンプレート X' を生成する．

4.2 C-Mapping のアルゴリズム

フェーズ 1 はさらに次のステップに分けられる．
ステップ 1 $XFDSet$ を用いて，リレーショナルスキーマ RS' を生成する．
ステップ 2 $XINDSet$ を用いて，RS' を $XINDSet$ を考慮に入れたリレーショナルスキーマ RS に変形する．

次から各ステップについて説明する．

4.2.1 ステップ 1

RS' は $XFDSet$ の各 XFD+を用いて生成される．ステップ 1 の手順を図 6 に示す．基本的には，$XFDSet$ の各 XFD+である xfd_i に対して，xfd_i の各関数従属性の決定子に対応する主

```
1. Procedure Step1 {
2.   //input:XFDSet
3.   //output:RS'
4.   let XFDSet={xfd1, xfd2, ...};
5.   let xfdn={"det1→res1", "det2→res2", ...};
6.   RS'=empty;
7.   for each xfdi ∈ XFDSet {
8.       rs=empty;
9.       for each "det→res" ∈ xfdi {
10.          if rs==empty {
11.              rs.union({det(as primarykey)});
12.          }
13.          rs.union({res});
14.      }
15.      RS'.union(rs);
16.  }
17.  return RS';
18. }
```

Figure 6: ステップ 1 の手順

キー属性と，被決定子に対応する属性を持つリレーションスキーマを生成する．

例えば，図 2 の book.xml に関する下記の $XFDSet$ が与えられたとする．

$XFDSet = \{xfd_1, xfd_2\}$

xfd_1 : `for $x in book.xml/root/book`
`$x → $x/booktitle/text()`
`$x → $x/author/text()`

xfd_2 : `for $y in book.xml/root/novel`
`$y → $y/noveltitle/text()`
`$y → $y/author/text()`

上記 $XFDSet$ が与えられると，ステップ 1 では図 6 に従って処理を行う．具体的には各 XFD+ に対して 7〜16 行目の処理が実行される．まず，xfd_1 に対して処理を行う．9〜14 行目で xfd_1 の決定子に対応する主キー属性 (`book`) と被決定子に対応する属性 (`booktitle` と `author`) を持つリレーションスキーマを生成する．そして 15 行目で作成したリレーションスキーマをリレーショナルスキーマ RS' に追加する．次に，xfd_2 のためのリレーションスキーマも同様に生成する．最終的なリレーショナルスキーマ RS' は下記のようになる．

```
book(bookID, booktitle, author)
novel(novelID, noveltitle, author)
```

生成されるリレーションの属性の名前とドメインは下記のルールに従って与えられる．(1) XFD+の XML パス式の最後の構成要素が `text()` である場合，属性名はその構成要素の親ノードの名前とし，ドメインはテキストとする．(2) XFD+の XML パス式の最後の構成要素が要素の型 (例えば要素名等) である場合，属性名をその要素名+"`ID`" とし，ドメインはノード ID とする．なお，格納されるノード ID はフェーズ 2 で生成するものであり，明示的に指定された XML ノード ID とは異なる．(3) XFD+の XML パス式の最後の構成要素が仮想的な属性 (表 1) である場合，属性名とドメインは，それぞれ仮想的な属性を計算するための関数の名前と，その関数の値域とする．

4.2.2 ステップ 2

. ステップ 2 では，SQL データベースで一般にサポートされている**外部キー制約**を有効に利用して，$XINDSet$ をマッピング後も維持できるように RS' を RS に変形する．

```
1. Procedure Step2 {
2.   //input: RS', XINDSet
3.   //output: RS
4.   RS=RS'
5.   let XINDSet={dep_1 ⊇ ref_1, dep_2 ⊇ ref_2 ,...}
6.   for each "dep_i ⊇ ref_i"∈ XINDSet {
7.       let R[A] = CorrespondingAttr(dep_i) in RS
8.       let S[B] = CorrespondingAttr(ref_i) in RS
9.       S.addAttribute(R.primarykey)
10.      S.removeAttribute(B)
11.  }
12.  return RS;
13. }
```

Figure 7: ステップ 2 の手順

外部キー制約は，参照される属性を主キー属性とする，包含従属性の一種である．例えば，$R(\underline{K_R}, \ldots, A, \ldots)$ と $S(\underline{K_S}, \ldots, B, \ldots)$ という 2 つのリレーションスキーマがある時，外部キー制約 $R[K_R] \supseteq S[B]$ とは，$S[B]$ の値は必ず $R[K_R]$ にも存在していなければならないという包含従属性を表す．一般的な RDBMS は，指定された外部キー制約をリレーションが満たすようチェックする機能を持っている．

しかし，外部キー制約には，参照される属性がキー属性 (主キーもしくは一意キー) でなければならないという制限が存在する [9]．すなわち，$R[A] \supseteq S[B]$ のように参照される属性 A が R におけるキー属性ではない一般の包含従属性の維持は，SQL データベースではサポートされていない．C-Mapping は入力の $XINDSet$ の中に一般の包含従属性 "$e_1 \supseteq e_2$" が存在しても受理する．しかし，リレーションへのマッピング時には，入力された "$e_1 \supseteq e_2$" に対応するリレーション属性間の包含従属性 $U[A_{e_1}] \supseteq V[A_{e_2}]$ において，A_{e_1} が必ずしも U におけるキー属性になるとは限らない．したがって，ステップ 1 の結果そのままでは，与えられた XIND を SQL データベースにおける外部キー制約として実装することはできない．

この問題に対処するために，本論文では，入力された XIND に対応する一般の包含従属性 $R[A] \supseteq S[B]$ を外部キー制約 $R[K_R] \supseteq S[K'_R]$ に置き換える手法を導入する．ここで，K_R は R におけるキー属性であり，K'_R は S に追加された，K_R のコピーを格納する属性である．

新たに生成された外部キー制約が元の制約のセマンティクスを保持するためには，S に追加された K'_R の値は，リレーション $T = R \bowtie_{K_R=K'_R} S$ において $\forall t \in T(t[A] = t[B])$ が成立するように選ぶ必要がある．この条件を満たす時，R において K_R がキー属性である事から $K_R \rightarrow A$ が成立し，S において $K'_R \rightarrow B$ が成立する．したがって，$R[K_R] \supseteq S[K'_R]$ ならば $R[A] \supseteq S[B]$ を満たす．

以上の議論を反映して，ステップ 2 では次の 2 つの処理を行う．(1) 上記の条件を満たすように，R のキー属性 K_R のコピー属性 K'_R を S に追加する．(2) $S[B]$ に格納される属性値は R と S の結合演算で求める事ができるため，S から属性 B を除去し，冗長性を除去する．図 7 は具体的なステップ 2 の手順である．まず，$R[A]$ と $S[B]$ が，XIND の左右の XML パス式に対応するリレーション属性とする (7, 8 行目)．9 行目で上記 (1)，10 行目で (2) の処理を行っている．

例えば，3.3 節で示した XIND がステップ 2 の入力として与えられると，図 5 に示すリレーショナルスキーマが生成される．

自明な XIND の扱い. 複数の XFD+によって参照される同一の XML ノード集合がある場合，その XML ノード集合は複数のリレーション属性値集合にマップされるため，その集合間には自明な包含従属性が存在する．例えば，$XFDSet = \{xfd_1, xfd_2\}$ であり，

xfd_1 : `for` $x `in history/person`
`return` $x → $x`/birthdate/text()`

xfd_2 : `for` $x `in history/person`
$x`/name/text()` → $x`/birthdate/text()`

とする．xfd_1 と xfd_2 をそれぞれ $p_1 \rightarrow b_1, n_2 \rightarrow b_2$ と略記すると，b_1 と b_2 は同一の XML

ノード集合を表すため，これらの間には自明な XIND が存在する．これらの包含従属性の向きは次の性質を利用して決定できる．一般に，この形式の XFD が存在する場合には，$p_1 \rightarrow n_2$ もしくは $n_2 \rightarrow p_1$ が成立することが多い (証明は，n_2 と p_1 が N:M 関係になると birthdate が単一の同値類になりやすい事を利用する)．この例では，$p_1 \rightarrow n_2$ が成立する．その場合，xfd_1 の被決定子が含まれる形の $b_1 \subseteq b_2$ に対応するリレーションの包含従属性が適用される．

4.3 リレーションインスタンスの構築

フェーズ 1 で生成されたリレーションスキーマに対するインスタンスは，そのスキーマの基となった XFD+を再び用いて構築する．その考え方は単純である．すなわち，これらの XFD+の決定子と被決定子の各 XPath 式を，元の XML データに適用すれば，そのスキーマに格納すべきタプルの集合が生成できる．例えば，3.1 節に示した XFD を用いてリレーションスキーマ book(bookID, booktitle) を作成したが，このインスタンスに含まれるタプル ((1, "The Sartorialist") 等) は，book.xml(図 2) にその XFD の各 XPath 式を適用することによって生成される．

リレーションインスタンスの構築にあたり注意すべき点が 2 つある．第一の注意点は，フェーズ 1 のステップ 1 で生成された “要素名+ID” 属性に格納する属性値を求める必要がある事である．これは，要素 (XML 木ノード) の ID を表す値であるが，生成されたリレーションの中だけで利用されるため，何らかの単純な方法で生成すれば十分である．本提案手法を実装した現システムでは，入力された XML 木の深さ優先順に従って各 XML 木ノードに ID 値を割り振っている．

第二の注意点は，フェーズ 1 のステップ 2 において，一般の包含従属性 $R[A] \supseteq S[B]$ を対応する外部キー制約 $R[K_R] \supseteq S[K'_R]$ に置き換える処理を行った場合，S に追加した属性 K'_R に格納する値を特定する必要がある事である．この場合，S から冗長な属性 B を除去する前に，4.2 節の条件を満たすような S の K'_R の値を計算する必要がある．

この計算を行うための単純な方法は，$t'[B] = t[A]$ となるタプル $t' \in S$ と $t \in R$ に対して，$t'[K'_R]$ に $t[K_R]$ の値を挿入する事である．しかし，$t'[B] = t[A]$ を満たす t は必ずしも一意ではないため，$t'[K'_R]$ の値が必ずしも 1 つに特定されるとは限らない．この問題を示す例を，図 5 の book.xml を用いて説明する．図中には存在しないが，仮に，この book.xml に “The Little Prince” という同名タイトルの小説と絵本が，それぞれ個別の book 要素として存在すると仮定する．また，そのうち小説版のみが novel 要素にも存在すると仮定する．さらに，book.xml のマッピング結果として，Book リレーションと Novel リレーションが生成されたと仮定する (Book が R に，Book[booktitle] が先の $R[A]$ に該当する．また，Novel が S に，Novel[noveltitle] が $S[B]$ に，Novel[bookID] が $S[K'_R]$ に該当する)．

この時，Book リレーションは図 5 の Book リレーションのタプルに，さらに 2 つのタプル (3, The little Prince, Saint-Exupery) と (4, The Little Prince, Saint-Exupery) が追加されたリレーションとなる．ここで，t'[noveltitle]=“The Little Prince” となる Novel のタプル t' に対して，t'[bookID] に挿入する値は t[booktitle]=“The Little Prince” となる Book のタプル t のキー属性値であるべきである．しかし，その条件を満たす Book のタプルは 2 つあるため，t は一意には決定できない．

この問題への対策は複数考えられる．最も簡単な対策は，ユーザにインタラクティブに問合せを行い，適切な値の選択を求めるというものであるが，本論文ではこの対策に関するより詳細な議論は今後の課題とする．

4.4 XML テンプレートの構築

マップされたリレーションから XML データへの復元が必要な時には，XML テンプレートを構築する必要がある．XML テンプレートとは，元となる XML データを表す XML 木からリレーションにマップされる部分木を除去し，その部分木に，元の XML 部分木を構築するための RXL 問合せを追加したものである．XML テンプレートの構築に必要な入力は，XML から RDB へのマッピングに利用した XFDSet と，復元部分の XML データの DTD である．本節では，これらを用いて XML テンプレートを構築する方法を説明する．

図 8 は，出版社毎の書籍情報を格納した publisher.xml を対象として，$XFDSet_1$ を用いてマップした例である．図 9(左) はこの例において構築される XML テンプレートであり，図

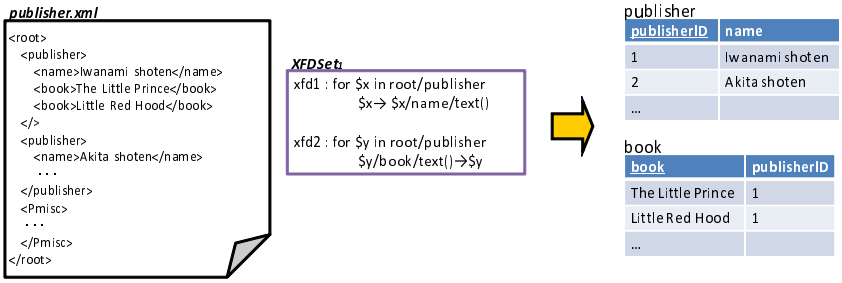

Figure 8: マッピング例

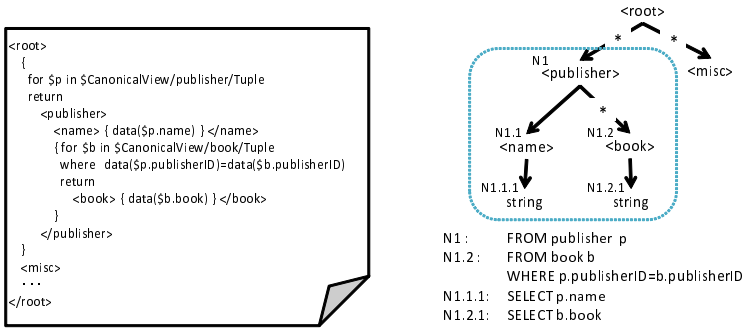

Figure 9: 図 8 のマッピング時に生成される XML テンプレート

9(右) はこの XML テンプレートを等価な木表現で表したものである．ルートノードは，元の XML のルート要素に対応する．点線内の部分木は，テンプレート中の各 RXL 問合せを木構造で表現したものであり，View 木と呼ばれる [11]．したがって，XML テンプレートの生成の問題は，RDB にマップされた部分の View 木をどのように生成するかという問題に帰着できる．

View 木は，RXL 問合せにより生成する XML の DTD 構造を木構造で表し，各ノードを計算するための SQL 断片を併記した物である．View 木の各ノードは Dewey エンコーディングによるノード ID (N1 等) と，SQL 断片として from 節を持つ．あるノード n がある時，ノード n に対応する XML 要素は次のように生成される．まず，ルートノードから n へのパスに存在する全ての from 節を組み合わせてリレーションの結合演算を行う．次に，その結果に含まれる各タプルに対して XML 要素を生成する．例えば，N1.2 に対応する XML 要素は，N1 と N1.2 の SQL 断片を組み合わせて作成した SQL 問合せ “`select * from publisher p, book b where p.publisherID=b.publisherID`” の実行結果に含まれる各タプル毎に生成する．View 木のリーフノードは，さらに SQL 断片に select 節を指定する事ができる．これは，そのノードの値が，結合されたリレーションの select 節で指定された属性値となることを示す．View 木の詳細は論文 [11] にある．

本手法では，View 木を次の手順で構築する．(1) View 木の木構造を構築する．(2) View 木の各ノードに対応する SQL 断片を決定する．

(1) View 木の木構造の生成. まず，与えられた DTD から木構造を作成し，次に，入力 $XFDSet$ から参照されているノードによって構成される部分木を特定する．特定された各部分木が，それぞれ 1 つの View 木の木構造となる．

(2) SQL 断片の生成. SQL 断片の生成が最も簡単な場合は，与えられた $XFDSet$ が次の XFD から構成されている場合である．まず，View 木の構造において，ノード n から 1:1 で接続される子ノード n_i と，1:N で接続されている (すなわち，経路に*ノードが存在する) 子ノード n_j が存在するとする．この時，$XFDSet$ は $e(n) \rightarrow e(n_i)$ と $e(n_j) \rightarrow e(n)$ から構成されていると仮定する．ここで $e(n)$ は n を表す XPath 式である．図 8 の $XFDSet_1$ は，この条件を満たしている．

この場合，ノード n の SQL 断片は，(一般に複数の)$e(n) \rightarrow e(n_i)$ から構成されるリレーション R_n を利用した “`from` R_n” となり，各ノード n_j の SQL 断片は，$e(n_j) \rightarrow e(n)$ に基

づき生成されるリレーション R_{n_j} に基づく “`from` R_{n_j} `where` $R_{n_j}.n$ の `ID`=$R_n.n$ の `ID`” となる．例えば，図 9 の場合には，ノード n に対応する publisher ノードの SQL 断片は “`from publisher`” となり，ノード n_j に対応する book ノードの SQL 断片は “`from book where book.publisherID=publisher.publisherID`” となる．

一般には，入力された XFDSet に応じて，上記とは異なるリレーションが生成される．その場合には，与えられた関数従属性と，先に説明した XML 構造から導かれる XFD を利用する事により，それらのリレーションから上記のリレーション相当物を導出する必要がある．これらは，与えられた関数従属性と，XML 構造から導ける関数従属性を組み合わせることによって導出できるが，次の点に注意が必要である．(1) 与えられた関数従属性が XML データ構造と矛盾する場合には，必要なリレーションが導出できない．しかし，一般的なアプリケーションではそのような事が起こることは無いと考えられる．(2) テンプレート構築に必要な関数従属性が明示的に指定されていない場合には，暗黙の関数従属性の存在を利用者に再確認する事により，リレーションを導出する事ができる．例えば，出版社のノード ID が明示的に格納されていない場合に，出版社名が決まればノード ID が決まるという関数従属性の存在を確認できれば，出版社名をノード ID と見なしてリレーションを導出可能である．

4.5 C-Mapping の制限

フェーズ 1 のステップ 2 において，一般の包含従属性を外部キー制約に置き換える手法を提案した．これが適用可能であるための前提条件は，包含従属性の両辺に対応するリレーション属性が，URA [8] の universal instance において同一の属性であると見なされ，別々に格納した際には冗長と見なされる事である．これ以外の場合には，C-Mapping では扱う事ができない．例えば，任意の文字列を格納するための属性 $String$ と，任意の学生の名前を格納するための属性 $Name$ 間の包含従属性 $String \supseteq Name$ は，それぞれ本質的に異なる属性であり冗長ではないため，C-Mapping では扱う事ができない．しかし，実用上はこのような包含従属性を扱う必要は無いと考えられる．

5 C-Mapping の記述力

本章では C-Mapping の記述力について議論し，C-Mapping が既存のマッピング手法の多くをシミュレート可能である事を証明する．C-Mapping は，XFD+と XIND をサポートすることにより，これらのマッピングに加えて N:1 マッピングを実現している．C-Mapping の記述力をまとめたものを図 10 に示す．図 10 では，それぞれ円で囲ったものが，入力として与える制約のクラスにより C-Mapping がシミュレート可能な既存手法を表している．まず，C-Mapping がサポートするデータ一貫性制約が，他の制約ベースアプローチがサポートするデータ一貫性制約を包含していることから，これらをシミュレート可能であることを示すことは容易である．C-Mapping が RRXS をシミュレート可能であることは，入力として XFD だけを与えられたときの C-Mapping は RRXS と同等である事から自明である．ただし，RRXS は冗長性を考慮して自動的に入力 XFD の書き換えを行うが，C-Mapping には書き換え後の XFD を与えることになる．次に，C-Mapping が X2R のマッピングをシミュレート可能である事は，key/keyref 制約が XIND の特別な場合であることからわかる．

他のアプローチに関しては自明ではないため，本章で証明を示す．具体的には，モデル写像アプローチの各手法に関しては，決定子を**構造に関する情報** (すなわち，`text()` のようなコンテンツデータではない情報．ノード ID やパス等) に限定した XFD+を用いる事でシミュレートする事が可能である．また，構造写像アプローチの各手法に関しては，決定子を構造に関する情報に限定した XFD を用いる事でシミュレートする事が可能である．

5.1 構造写像アプローチのシミュレート

構造写像アプローチは，DTD の構造を反映したリレーショナルスキーマを生成するアプローチである．このアプローチでは，DTD で定義された XML 要素型はリレーションあるいはリレーションの属性のどちらかにマップされる．

この基本的な方法は次の通りである．すなわち，ある XML 要素型 a がリレーションにマップされる時，そのリレーションのキー属性は必ず a の ID となり，それ以外の属性は DTD で定義

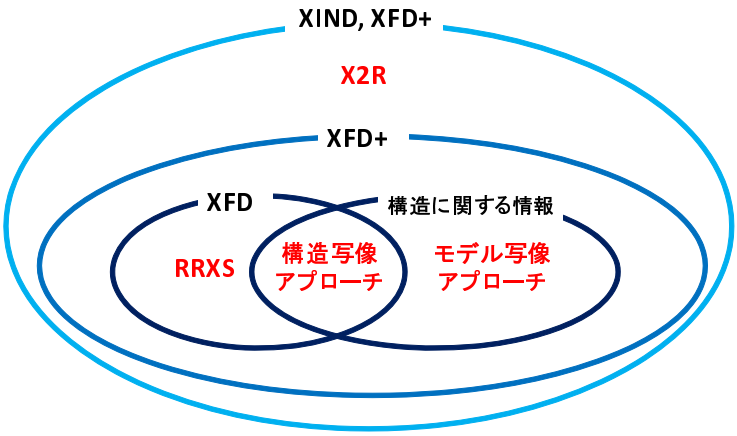

Figure 10: C-Mapping の記述力

root

rootID
1

book

bookID	book.parentID	book.booktitle	book.author
1		The Sartorialist	Scott Schuman
2		Sherlock Holmes	Arthur Conan Doyle
...			

root.book

root.bookID	root.book.parentID	root.book.booktitle	root.book.author
1	1	The Sartorialist	Scott Schuman
2	1	Sherlock Holmes	Arthur Conan Doyle
...			

novel

novelID	novel.noveltitle	novel.author
1	Sherlock Holmes	Arthur Conan Doyle
...		

root.novel

root.novelID	root.novel.parentID	root.novel.noveltitle	root.novel.author
1	1	Sherlock Holmes	Arthur Conan Doyle
...			

noveltitle

noveltitleID	noveltitle
1	Sherlock Holmes
...	

author

authorID	author
1	Scott Schuman
2	Arthur Conan Doyle
3	Arthur Conan Doyle
...	

booktitle

booktitleID	booktitle
1	The Sartorialist
2	Sherlock Holmes
...	

reference

referenceID	reference.parentID	reference.booktitle	reference.author
...			

book.reference

book.referenceID	book.reference.parentID
...	

Figure 11: Basic-Inlining による book.xml のマッピング結果

された要素間の関係を表した DTD グラフにおいて，a から到達可能なリーフノードを格納するための属性となる．例えば，図 2 で示した Basic-Inlining のマッピング結果である `book` リレーションにおいて，キー属性は `bookID` であり，それ以外の属性は図 1 の DTD グラフにおいて book 要素ノードから到達可能なリーフノードを格納するための属性 (`booktitle`, `author`) である．論文 [1] では，手法によってリレーションにマップする XML 要素型とリレーションの属性にマップする XML 要素型を選択するための基準が異なる．例えば，Shared-Inlining は `book` リレーションに対して，book 要素ノードから到達可能である author を格納するための属性を追加しない．

次項から，C-Mapping が構造写像アプローチの主な手法である Basic-Inlining, Shared-Inlining, Hybrid-Inlining をシミュレート可能である事を示す．ただし以降は，説明を簡潔にするため次の XFD を $x \rightarrow \{y_1, y_2, ..., y_n\}$ と記述する．

for $\$x$ in P_x/x
  $\$x \rightarrow \$x/P_1/y_1$
  $\$x \rightarrow \$x/P_2/y_2$
  ...
  $\$x \rightarrow \$x/P_n/y_n$

5.1.1 Basic-Inlining

Basic-Inlining は DTD で定義された XML 要素型全てをリレーションにマップする手法である．book.dtd(図 1) に従う book.xml(図 2) のマッピング結果のリレーションを図 11 に示す．

Basic-Inlining の詳細

Basic-Inlining は最初に DTD グラフの (要素型を表す) 各要素ノード毎に要素グラフと呼ばれるグラフを生成し，その要素グラフを用いてリレーションを生成する．要素グラフは，DTD グラフにおいて対象となる要素ノードから深さ優先順で探索を行うことで生成される．例として図 1 の DTD グラフの book 要素ノードに対し生成される要素グラフを図 12 に示す．図 12 の backpointer エッジは，探索中に既に訪問済みのノードに到達した際に張られるエッジである．

Basic-Inlining は要素グラフを用いて次の手順でリレーションを生成する．(1) 1 つの要素グラフ G に対し 1 つの A リレーションを生成する．A の主キー属性は要素グラフのルート要素ノードの ID であり，それ以外の属性は次の 2 つのいずれかを満たす要素ノードを訪問せずに到達可能なリーフノードである．(a) “ * ”ノードの子供である要素ノード．(b) backpointer エッジの始点である要素ノード．(2) (a) あるいは (b) の条件を満たす各要素ノード (e_i) を格納するため，B_i リレーションを生成する．B_i の主キー属性は e_i の ID であり，それ以外の属性は G においてルート要素ノードを訪問せずに到達可能なリーフノードである．(3) 親を持つ要素 (ルートでは無い要素) に対応する全てのリレーションに親を特定するための parentID 属性を追加する．各タプルの parentID に格納される値は，親となる要素ノードの ID である．

図 12 の要素グラフを用いて `book` リレーションを生成する例を説明する．まず，ルート要素ノードである book のために `book` という名前のリレーション (上記 A に対応) を生成する．`book` リレーションの属性は，`bookID` (主キー属性), `author`，`booktitle` である．次に，reference 要素ノードが (a) (さらに (b)) を満たすので，`book.reference` という名前の B_i リレーションを生成する．最後に `book` リレーションと `book.reference` リレーションに `parentID` 属性を追加する．最終的に `book` リレーションと `book.refrence` リレーションは図 11 のようになる．

Basic-Inlining のシミュレート

定理 1 *C-Mapping* が *Basic-Inlining* のマッピングをシミュレート可能である． □

(証明)　Basic-Inlining は A リレーションと B_i リレーションの生成を必要とする．ここでは DTD で定義されたある要素型 a の要素グラフ G において，(a) あるいは (b) を満たす要素ノード b が存在すると仮定する．この時，A リレーションは次の XFD を C-Mapping に与える事で生成される．

$$a \rightarrow \{n | n \in V(G) \wedge (P_1(n) \vee P_2(n, a))\}$$

ここで，$P_1(n)$ は，n が G において (a) あるいは (b) を満たす要素ノードを訪問せずに a から到達可能なリーフノードである時に真となる述語であり，$P_2(n_1, n_2)$ は n_1 が G において n_2 の親ノードである時に真となる述語である．また，$V(G)$ はグラフ G に存在するノードの集合を返す関数である．

次に，B_i リレーションは次の XFD を C-Mapping に与える事で生成される．

$$b \rightarrow \{n | n \in V(G) \wedge (P_3(n) \vee P_2(n, b))\}$$

ここで，$P_3(n)$ は，n が G において a を訪問せずに b から到達可能なリーフノードである時に真となる述語である． □

5.1.2 Shared-Inlining

Shared-Inlining は Basic-Inlining のように DTD で定義された XML 要素型全てをリレーションにマップする事はせず，Shared-Inlining における条件を満たした XML 要素型のみをリレーションにマップする手法である．book.dtd (図 1) に従う book.xml (図 2) のマッピング結果を図 13 に示す．

Shared-Inlining の詳細

Shared-Inlining は DTD グラフにおいて，次の条件のいずれかを満たす XML 要素ノードのみをリレーションにマップする手法である．(a) ルート要素ノード，(b)“ * ”ノードの子供である要素ノード，(c) 入次数が 2 以上の要素ノード，(d) DTD グラフの強連結成分に含まれ，かつ入次数が 1 の要素のうちのどれか 1 要素ノード．図 1 の DTD グラフにおいて (a) は root 要

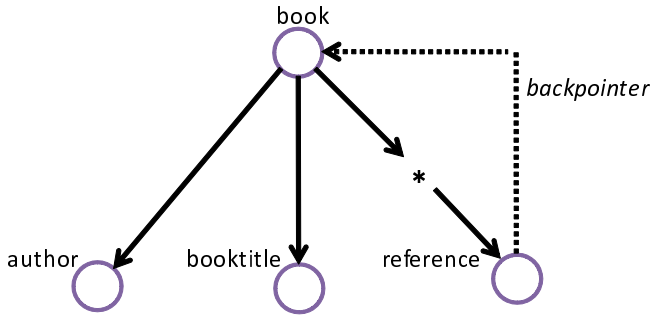

Figure 12: book 要素グラフ

root

rootID
1

novel

novelID	novel.parentID	novel.parentCODE	novel.noveltitle
1	1	1	Sherlock Holmes
...			

book

bookID	book.parentID	book.parentCODE	book.booktitle
1	1	1	The Sartorialist
2	1	1	Sherlock Holmes
...			

reference

referenceID	reference.parentID	reference.parentCODE
...		

author

authorID	author.parentID	author.parentCODE	author
1	1	1	Scott Schuman
2	2	1	Arthur Conan Doyle
3	1	2	Arthur Conan Doyle
...			

Figure 13: Shared-Inlining による book.xml のマッピング結果

素ノード，(b) は book, novel, reference 要素ノード，(c) は book, author 要素ノード，(d) は book か reference どちらかの要素ノード，となる．

Shared-Inlining は次の手順でマッピングを行う．(1) DTD グラフの要素ノードを (a)〜(d) に分類する．(2) 分類された各要素ノード e_i に対応するリレーションを生成する．具体的には，主キー属性が e_i の ID，それ以外の属性が DTD グラフにおいて e_i から (a)〜(e) に分類された他の要素ノードを訪問せずに到達可能なリーフノードであるリレーションを生成する．ただし，(b), (c), (d) の要素に対応するリレーションは上記の属性に加え，親を特定する parentID と parentCODE という属性を持つ．具体的なマッピング例として，図 1 の DTD グラフにおける book 要素ノードをマップすると，主キー属性が book 要素ノードの ID，それ以外の属性が book 要素ノードから (a)〜(e) に分類された他の要素ノードを訪問せずに到達可能な booktitle 要素ノードと，親を特定する parentID，parentCODE 属性である `book` リレーションを生成する．最終的に `book` リレーションは図 13 のようになる．

Shared-Inlining のシミュレート

定理 2 *C-Mapping が Shared-Inlining のマッピングをシミュレート可能である．* □

(証明)　Shared-Inlining のマッピングは分類された各要素ノード毎に対応するリレーションの生成を必要とする．ここでは，マップする XML データの DTD グラフを D と仮定する．

(a) を満たす要素ノード a に対応するリレーションは次の XFD を C-Mapping に与える事で生成される．

$$a \rightarrow \{n | n \in V(D) \wedge P_4(n, a)\}$$

ここで，$P_4(n_1, n_2)$ は，n_1 が D において (a), (b), (c), (d) に該当する要素を訪問せずに n_2 から到達可能なリーフノードである時に真となる述語である．

(b), (c), (d) を満たす要素ノード b に対応するリレーションは次の XFD を C-Mapping に与える事で生成される．

root

rootID
1

reference

referenceID	reference.parentID	reference.parentCODE	reference.book.booktitle	reference.book.author
...				

novel

novelID	novel.parentID	novel.parentCODE	novel.noveltitle	novel.author
1	1	1	Sherlock Holmes	Arthur Conan Doyle
...				

book

bookID	book.parentID	book.parentCODE	book.booktitle	book.author
1	1	1	The Sartorialist	Scott Schuman
2	1	1	Sherlock Holmes	Arthur Conan Doyle
...				

Figure 14: Hybrid-Inlining による book.xml のマッピング結果

$$b \to \{n | n \in V(D) \wedge (P_4(n, b) \vee P_5(n))\}$$

ここで，$P_5(n)$ は，n が D において b の親ノードである時に真となる述語である． □

5.1.3 Hybrid-Inlining

Hybrid-Inlinig は基本的には Shared-Inlining とマッピング規則は同じだが，5.1.2 項で述べたリレーションにマップする XML 要素型の条件 (c) が除去されている．book.dtd (図 1) に従う book.xml (図 2) のマッピング結果を図 14 に示す．

Hybrid-Inlining のシミュレート

定理 3 *C-Mapping が Hybrid-Inlining のマッピングをシミュレート可能である．* □

定理 3 の証明は，定理 2 の証明においてリレーションにマップする XML 要素型の条件 (c) を除去したものと同じであるため，省略する．

5.2 モデル写像アプローチのシミュレート

モデル写像アプローチは，XML データモデルを反映したリレーションスキーマを生成するアプローチである．そのため，生成されたリレーションには任意の整形式 XML データを格納することができる．

次項から，モデル写像アプローチの主な手法である XRel [2]，枝アプローチ [3] を C-Mapping がシミュレート可能である事を示す．以降は，入力として与えられる整形式 XML データを test.xml と仮定する．

5.2.1 XRel

XRel の主なアイデアは，XML 木のルートノードから全てのノード (要素，属性，テキスト) へのパスを格納するためのリレーションを生成することである．さらに，ノード (要素，属性，テキスト) の XML テキスト上での文字列の位置を表すバイトオフセットを格納するリレーションを生成する．book.dtd(図 1) に従う book.xml(図 2) のマッピング結果を図 15 に示す．

XRel の詳細

XRel は XML データを次の 4 つのリレーションにマップする．(a) `path`, (b) `element`, (c) `attribute`，(d) `text`.

(a) `path` リレーションは，主キー属性として `pathID` (各パスの ID)，それ以外の属性として `pathexpression` (ルートからノードへのパス) を持つ．(b) `element` リレーションは，主キー属性として `docID` (格納された要素を含む XML データの ID)，`start` (XML 要素の (XML テキスト上での位置における) 開始バイトオフセット)，`end` (XML 要素の終了バイトオフセット)，それ以外の属性として `pathID` (格納された要素に到達するためのパスの ID)

path

pathID	pathexpression
1	#/root
2	#/root#/book
3	#/root#/book#/booktitle
4	#/root#/book#/author
5	#/root#/novel
6	#/root#/novel#/noveltitle
7	#/root#/novel#/author
...	

element

docID	pathID	start	end
1	1	0	...
1	2	6	87
1	3	12	50
1	4	51	80
1	2	88	173
1	3	94	131
1	4	132	166
...			

text

docID	pathID	start	end	value
1	3	23	38	The Sartorialist
1	4	59	71	Scott Schuman
1	3	105	119	Sherlock Holmes
1	4	40	57	Arthur Conan Doyle
1	6	193	207	Sherlock Holmes
1	7	229	246	Arthur Conan Doyle
...				

attribute

docID	pathID	start	end	value
...				

Figure 15: XRel による book.xml のマッピング結果

を持つ．(c) `attribute` リレーションは，主キー属性として `docID` (格納された属性を含む XML データの ID)，`pathID` (格納された属性に到達するためのパスの ID), `start` (XML 属性の開始バイトオフセット)，`end` (XML 属性の終了バイトオフセット)，それ以外の属性として `value` (XML 属性値) を持つ．(d) `text` リレーションは，主キー属性として `docID` (格納されたテキストを含む XML データの ID)，`start` (テキストの開始バイトオフセット)，`end` (テキストの終了バイトオフセット)，それ以外の属性として `pathID` (格納されたテキストに到達するためのパスの ID)，`value` (テキスト) を持つ．

XRel のシミュレート

定理 4 *C-Mapping が XRel のマッピングをシミュレート可能である．* □

(証明)　XRel は上記で説明した 4 つのリレーションを生成する必要がある．
(a) `path` リレーションは次の XFD+を C-Mapping に与える事で生成される．
for $\$n$ in test.xml//* | test.xml//@*
　$\$n$/@pathID() → $\$n$/@path()
(b) `element` リレーションは次の XFD+を C-Mapping に与える事で生成される．
for $\$e$ in test.xml//*
　test.xml/@docID(), $\$e$/@pre(), $\$e$/@post() → $\$e$/@pathID()
(c) `attribute` リレーションは次の XFD+を C-Mapping に与える事で生成される．
for $\$a$ in test.xml//@*
　test.xml/@docID(), $\$a$/@pathID(), $\$a$/@pre(), $\$a$/@post() → $\$a$
(d) `text` リレーションは次の XFD+を C-Mapping に与える事で生成される．
for $\$t$ in test.xml//text()
　test.xml/@docID(), $\$t$/@pre(), $\$t$/@post() → $\$t$/@pathID(), $\$t$ □

5.2.2　枝アプローチ

枝アプローチは，XML データを木構造と見なし，ノードとエッジの組で XML データを表現するようなリレーションを生成する手法である．book.dtd(図 1) に従う book.xml(図 2) のマッピング結果を図 16 に示す．

枝アプローチの詳細

　枝アプローチは XML データを次の 2 つのリレーションにマップする．(a) `Edge`，(b) `Vtype`．(a) `Edge` リレーションは，主キー属性として `source` (ノードの ID)，`ordinal` (`source` を始点としたエッジの ID)，それ以外の属性として `name` (ノードの名前), `flag` (`ordinal` に対応するエッジの終点の型)，`target` (`ordinal` に対応するエッジの終点の ID) を持つ．(b) `Vtype` リレーションは，主キー属性として `vid` (テキストノードまたは属性ノードの ID)，それ以外の属性として `value` (テキストまたは属性値) を持つ．`Vtype` リレーションはテキストまたは属性値の型毎に存在する．例えばテキストが string 型であれば `Vstring`，int 型であれば `Vint` という `Vtype` リレーションが存在する．

枝アプローチのシミュレート

Edge

source	ordinal	name	flag	target
1	1	book	ref	2
1	2	book	ref	5
1	3	novel	ref	8
2	1	booktitle	string	v1
2	2	author	string	v2
5	1	booktitle	string	v3
5	2	author	string	v4
8	1	noveltitle	string	v5
8	2	author	string	v6
...				

Vstring

vid	value
v1	The Sartorialist
v2	Scott Schuman
v3	Sherlock Holmes
v4	Arthur Conan Doyle
v5	Sherlock Holmes
v6	Arthur Conan Doyle
...	

Figure 16: 枝アプローチによる book.xml のマッピング結果

定理 5 *C-Mapping* が枝アプローチのマッピングをシミュレート可能である． □

(証明)　枝アプローチは上記で説明した 2 つのリレーションを生成する必要がある．
(a) `Edge` リレーションは次の XFD+を C-Mapping に与える事で生成される．
for $\$n$ in test.xml//*
for $\$e$ in $\$n$/@edges()
$\$n, \$e \rightarrow \$n$/@nodeName()
$\$n, \$e \rightarrow \$e$/@childNode()/@type()
$\$n, \$e \rightarrow \$e$/@childNode()/@nodeID()
(b) `Vtype` リレーションは次の XFD+を C-Mapping に与える事で生成される．
for $\$v$ in test.xml//text()[./@type()=string] | test.xml//@*[./@type()=string]
$\$v$/@nodeID() $\rightarrow \$v$
上記 (b) は string 型の `Vtype` リレーションへのマッピングを表現しているが，他の型の `Vtype` リレーションは述語の string という部分を各型に変える事で生成できる． □

6　実データを用いた評価

本章では，5 章で理論的に示した記述力の違いが，実データを用いた場合にどのように影響を及ぼすのかを調査した結果を説明する．

6.1　評価概要

本評価では，Wikipedia [12] が提供しているダンプデータ (XML データ) [13] を対象とし，Wikipedia のバックエンド DB へのマッピングを制約ベースの各マッピング手法を利用してどの程度シミュレート可能かを検証する．Wikipedia を対象とした理由は，XML データとバックエンドの DB スキーマが公開されており，かつ一般的に良く知られているからである．

評価手順は次の通りである．(1) Wikipedia バックエンド DB のスキーマを調査．これは，Wikipedia ダンプ XML データを RDB に格納するための専用ツール [14] を調査することにより入手した．(2) 制約ベースアプローチの各手法の入力として適切な制約を与える事で，バックエンド DB のスキーマをどの程度実現できるかを調査．(3) 結果を考察．

対象 XML データ． Wikipedia はダンプデータとして様々な XML データや SQL データを提供している．本評価では，その中で，日本語 Wikipedia のダンプ XML データを選択した．具体的には，過去の履歴や利用者ページを含まない最新の日本語の Wikipedia ページの全記事がダンプされている jawiki-latest-pages-articles.xml (以降，jawiki.xml と表記．2011 年 12 月 17 日時点) を選択した．この XML データは，日本語 Wikipedia の内容そのものを持つ中心的なデータである．

対象 XML データの一部を図 17 に示す．

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.5 ...>
  ...
  <page>
    <title>レイトン教授シリーズ</title>
    <id>1256598</id>
    <revision>
      <id>37185679</id>
      <timestamp>2011-04-14T20:44:06Z</timestamp>
      <contributor>
        <username>Zorrobot</username>
        <id>216502</id>
      </contributor>
      <minor/>
      <comment>r2.5.2) (ロボットによる 追加: [[nn:Professor Layton]]</comment>
      <text xml:space="preserve" bytes="10429">
        {{Redirect|レイトン教授|本項シリーズの主人公・"'エルシャール・レイトン"'|レイトン教授シリーズの用語集}}
        "'レイトン教授シリーズ'"(レイトンきょうじゅシリーズ)は、[[レベルファイブ]]から発売されている&lt;!--3部構成と
        なっている--&gt;[[ニンテンドー DS]]および[[ニンテンドー 3DS]]用[[アドベンチャーゲーム|アドベンチャー]]ゲームシ
        リーズ、およびそれを主軸とした[[メディアミックス]]作品のシリーズである。
        ...
      </text>
    </revision>
  </page>
  ...
</mediawiki>
```

Figure 17: 対象 XML データ (一部)

XFDSet = {xfd1, xfd2, xfd3}, XINDSet = {xind1, xind2}

xfd1 : for $x in jawiki.xml/mediawiki/page
- $x/id/text() → *$x/title/@namespace()*
- $x/id/text() → *$x/title/@title()*
- $x/id/text() → *$x/revision/text/@redirect()*
- $x/id/text() → *$x/revision/timestamp/@timestamp()*
- $x/id/text() → *$x/revision/id/text()*
- $x/id/text() → *$x/revision/text/@bytes()*

xfd2 : for $x in jawiki.xml/mediawiki/page/revision
- $x/id/text(), $x/../id/text() → *$x/comment/text()*
- $x/id/text(), $x/../id/text() → *$x/contributor/username/id/text()*
- $x/id/text(), $x/../id/text() → *$x/contributor/username/username/text()*
- $x/id/text(), $x/../id/text() → *$x/timestamp/@timestamp()*
- $x/id/text(), $x/../id/text() → *$x/@minor()*
- $x/id/text(), $x/../id/text() → *$x/text/@bytes()*

xfd3 : for $x in jawiki.xml/mediawiki/page/revision
- $x/id/text() → *$x/text/text()*
- $x/id/text() → *$x/text/@textType()*

xind1 : jawiki.xml/mediawiki/page/revision/id/text() ⊇ jawiki.xml/mediawiki/page/revision/id/text()
xind2 : jawiki.xml/mediawiki/page/id/text() ⊇ jawiki.xml/mediawiki/page/id/text()

Figure 18: C-Mapping に入力する $XFDSet$ と $XINDSet$

6.2 生成対象のリレーション

上記で述べた専用のツールによって，対象 XML データから生成されるリレーションは次の通りである．

- `page` リレーション: Wikipedia の各ページの情報を格納するリレーション
- `revision` リレーション: Wikipedia の各ページの更新履歴を格納するリレーション
- `text` リレーション: Wikipedia の各ページの内容 (テキスト) を格納するリレーション

また，`page`, `revision`, `text` リレーション間には次の外部キー制約が存在する．

- `revision[rev_text_id]` ⊆ `text[old_id]`
- `revision[rev_page]` ⊆ `page[page_id]`
- `page[page_latest]` ⊆ `revision[rev_id]`

6.3 結果

対象 XML データ jawiki.xml と，jawiki.xml に関する図 18 の $XFDSet$ と $XINDSet$ を C-Mapping に与えることで，図 19 のリレーションが生成された．
オリジナルの `page`, `revision`, `text` リレーションと，C-Mapping のマッピング結果 (図 19) を比較すると，C-Mapping ではマップする事ができない属性が存在する事がわかる．C-Mapping で

page

id	namespace	title	redirec	timestamp	revision_id	bytes
…						
1256598	0	レイトン教授シリーズ	0	20110414204406	37185679	10429
…						

text

id	text	textType
…		
37185679	{{Redirect|レイトン教授|本項シリーズの主人公…	utf8
…		

revision

id	page_id	comment	username_id	username	timestamp	minor	bytes
…							
37185679	1256598	r2.5.2) (ロボット…	216502	Zorrobot	20110414204406	1	10429
…							

Figure 19: マッピング結果

はマップする事ができない属性としては page リレーションの page_counter, page_is_new, page_random, revision テーブルの rev_parent がある．これらは次のように分類できる．
元の XML からデータを生成できないもの． page_counter, page_is_new, rev_parent 属性は，対象 XML データからは取得できない値を持つため，マップする事ができなかった．page_random はページの内容に関係無く 0〜1 の値をランダムに生成して格納する属性であるため，マッピングとは独立した値でありマップする事ができなかった．
冗長な情報を持つもの． revision リレーションの rev_text_id 属性は同リレーションの rev_id のコピーである．C-Mapping では冗長な属性は削除されるため，これらは一つの属性として出力された．

まとめると，C-Mapping によって実用的なマッピングはほぼ実現可能であると考えられる．

6.4 他の制約ベースアプローチの手法との比較

制約ベースアプローチの各マッピング手法に対象 XML データと適切な制約を与える事で生成可能な page, revision, text リレーションの属性の数は，23 個の属性のうち，C-Mapping が 19 個，RRXS が 10 個，X2R が 10 個であった．また，C-Mapping のみが，Wikipedia の RDB に必要な 3 つの外部キー制約を実現できた．C-Mapping が他手法よりも生成可能なリレーションの属性の数が多い理由は，他手法では入力 XML データのテキスト中に存在しないデータをマッピングする事ができないが，C-Mapping では XFD+において仮想的な属性を使用する事が可能であるためである．また，他の手法で外部キー制約を実現できなかった理由は，RRXS は関数従属性のみを扱っているからであり，X2R においては外部キー制約を持つリレーションの形式が key/keyref 制約に従う形に制限されているからである．

7 おわりに

本稿では，関数従属性と包含従属性を用いた XML-RDB マッピング手法 C-Mapping の提案を行った．C-Mapping は次の特徴を持つ．(1) 1:1 と 1:N マッピングだけでなく，N:1 マッピングも実現できるという意味で完全なマッピング手法である．(2) 現在まで提案された XML-RDB マッピング手法の多くをシミュレート可能な記述力を持つ．

今後の課題としては，マッピングに必要な関数従属性と包含従属性を提案するサポートシステムの開発等がある．

References

[1] Jayavel Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David De Witt, Jeffrey Naughton: Relational Databases for Querying XML Documents: Limitations and Opportunities, the 25th VLDB Conference: 302-314 (1999).

[2] M. Yoshikawa, T. Amagasa, T. Shimura, S. Uemura: XRel: A path-based approach to storage and retrieval of XML documents using relational databases, ACM Transactions on Internet Technology 1(1): 110-141 (2001).

[3] Daniela Florescu, Donald Kossmann: Storing and Querying XML Data using an RDBMS, Bulletin of the Technical Committee on Data Engineering 22(3): 27-34 (1999).

[4] Yi Chen, Susan Davidson, Carmem Hara, Yifeng Zheng: RRXS: Redundancy reducing XML storage in relations, the 29th VLDB Conference: 189-200 (2003).

[5] Yi Chen, Susan B. Davidson, Yifeng Zheng: Constraint Preserving XML Storage in Relations, WebDB 2002.

[6] David C. Fallside, Priscilla Walmsley: XML Schema Part 0: Primer Second Edition, 2004, http://www.w3.org/TR/xmlschema-0/, (accessed 2011-2-10).

[7] B. Box: The XML Data Model, 1997, http://www.w3.org/XML/Datamodel.html, (accessed 2011-2-10).

[8] Serge Abiteboul, Richard Hull, Victor Vianu: Foundations of Databases, Addison-Wesley (1995).

[9] ISO/IEC 9075:1999: Information Technology-Database Language-SQL-Part 1-5 (1999).

[10] Michael Karlinger, Millist W. Vincent, Michael Schrefl: Inclusion Dependencies in XML: Extending Relational Semantics, In Database and Expert Systems Applications 2009: 23-37.

[11] Mary F. Fernandez, Yana Kadiyska, Atsuyuki Morishima, Dan Suciu, Wang-Chiew Tan: SilkRoute: A Framework for Publishing Relational Data in XML, ACM TODS 27(4): 438-493 (2002).

[12] Wikipedia, http://ja.wikipedia.org/, (accessed 2011-12-1).

[13] Wikimedia Downloads, http://download.wikimedia.org/, (accessed 2011-12-17).

[14] Manual Database layout, http://www.mediawiki.org/wiki/Manual:Database_layout, (accessed 2011-12-1).