DEIM Forum 2011 E6-3

関数従属性と包含従属性を用いたXML-RDBマッピング

太田　壮祐†　　森嶋　厚行††　　天笠　俊之†††

† 筑波大学大学院図書館情報メディア研究科　〒305–8550 茨城県つくば市春日 1–2
†† 筑波大学大学院図書館情報メディア研究科/知的コミュニティ基盤研究センター　〒305–8550 茨城県つくば市春日 1–2
††† 筑波大学大学院システム情報工学研究科/計算科学研究センター　〒305–8573 茨城県つくば市天王台 1–1–1
E-mail: †sousuke.ohta.2009b@mlab.info, ††mori@slis.tsukuba.ac.jp, †††amagasa@cs.tsukuba.ac.jp

あらまし　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:n マッピングを行うものであった．本稿では，XML データ上に存在する関数従属性と包含従属性をマッピング時に指定することにより，n:1 マッピングを実現可能とするマッピング手法 C-Mapping を提案する．C-Mapping により，既存のマッピング手法では困難であった，バックエンド RDB によるデータの一貫性管理が支援可能となる．本稿ではさらに，C-Mapping を用いれば，これまで提案された主要な手法である構造写像とモデル写像が実現可能である事を示し，C-Mapping が高い記述力を持つことを明らかにする．

キーワード　XML-RDB マッピング, 関数従属性，包含従属性

An XML-RDB Mapping Method using Functional and Inclusion Dependencies

Sosuke OTA†, Atsuyuki MORISHIMA††, and Toshiyuki AMAGASA†††

† Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba　1–2 Kasuga, Tsukuba, Ibaraki,Japan 305–8550 Japan
†† Grad. Sch. of Library, Information and Media Studies/Research Center for Knowledge Communities, Univ. of Tsukuba., Univ. of Tsukuba　1–2 Kasuga, Tsukuba, Ibaraki,Japan 305–8550 Japan
††† Grad. Sch. of Sys. and Info. Eng./Center for Computational Sciences, Univ. of Tsukuba　1–1–1 Tennohdai, Tsukuba, Japan 305–8573
E-mail: †sousuke.ohta.2009b@mlab.info, ††mori@slis.tsukuba.ac.jp, †††amagasa@cs.tsukuba.ac.jp

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 にマッピングする手法が数多く研究されてきた．これらのマッピング手法の主要なものとして構造写像アプローチ [1]，モデル写像アプローチ [2] [3]，RRXS [4] が存在する．

これらの手法は全て XML Data Model [5] における各ノード (以下 XML ノード) から RDB 属性値への 1:1 もしくは 1:n マッピングを行うものであった．具体例として，構造写像アプローチに基づくマッピングの一つである Shared-Inlining [1] によるマッピング例を説明する (図 1, 図 2)．Shared-Inlining によって，book.dtd (図 1 上) に従う book.xml (図 2 左) をマップすると，ある条件を満たす XML 要素に対応する幾つかのリレーションが生成される．その中の一つとして，book 要素に対応する book(bookID, book.booktitle) というスキーマを持つリレーションが生成される (図 2 右)．このリレーションでは，book 要素ノード (の ID) とその子要素である booktitle 要素ノードのテキストの情報が，それぞれあるタプルの bookID 属性値と book.booktitle 属性値として格納される．このように，book.xml の booktitle 要素のテキストノードと book リレーションの booktitle 属性値は 1:1 対応となっている．このような，一つの XML ノードがマッピング結果のリレーションの一つの属性値に対応するようなマッピングを，1:1 マッピングと呼ぶ．また，一つの XML ノードが複数の RDB 属性値に 1:n 対応となるマッピングを 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．

本稿では，新たな XML-RDB マッピング手法である C-

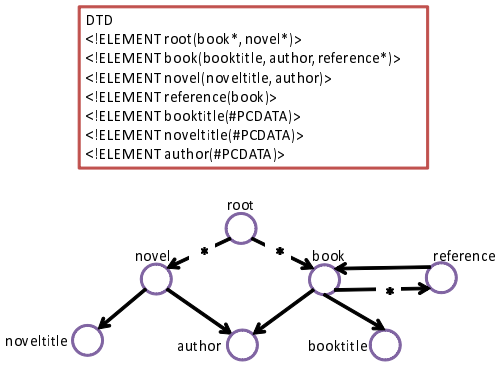

図 1　book.dtd(上) とその DTD グラフ (下)

Mapping (Consistency-conscious Mapping) を提案する．これは，入力として，XML データとそのデータにおける一貫性制約 (関数従属性と包含従属性) を指定することにより，RDB へのマッピング結果を出力するものである．C-Mapping は，1:1 と 1:n 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．また，C-Mapping では，マッピングの際に与える関数従属性や包含従属性を変更することにより，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 ノードを 1 つの RDB 属性値にマップすることである．図 1 の book.dtd に従う XML データ book.xml を n:1 マッピングを行った結果の例を図 3 に示す．この例では，図 3 の book.xml データには「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」という一貫性制約が存在すると仮定する．図 3 のリレーションでは，book.xml の booktitle 要素と noveltitle 要素に対応する XML ノードが 1 つの RDB 属性値にマップされている．このように n:1 マッピングを行うことで，XML データの更新時における一貫性維持が容易になる．すなわち，先に示したような 1:1 や 1:n のマッピングでは，`book` リレーションの `booktitle` 属性の “Sherlock Holmes” という RDB 属性値を更新すると，一貫性制約が破られ不整合が起きてしまうが，n:1 マッピングを実現すれば，このような問題に対応可能である．

本稿の構成は次のとおりである．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 の一貫性制約について述べる．4 章では C-Mapping のアルゴリズムを説明する．5 章では，C-Mapping を用いれば，これまで提案された主要な手法である構造写像アプローチ (Basic-Inlining，Shared-Inlining) とモデル写像アプローチ (経路アプローチ，枝アプローチ) を実現できる事を示し，C-Mapping の記述力を明らかにする．6 章はまとめと今後の課題である．

2. 関 連 研 究

我々の知る限り，既存の XML-RDB マッピング手法は全て 1:1 もしくは 1:n マッピングである．まず，構造写像アプローチは，個別の DTD に基づいてマップするアプローチである．このアプローチで生成されるデータベーススキーマは個別の XML スキーマに依存する．構造写像アプローチの主な手法としては Basic-Inlining, Shared-Inlining, Hybrid-Inlining [1] が

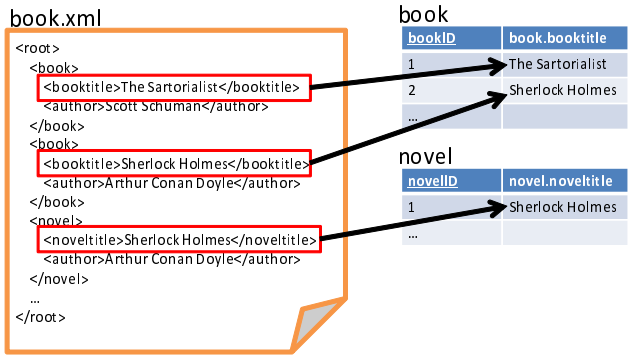

図 2　1:n マッピングの例

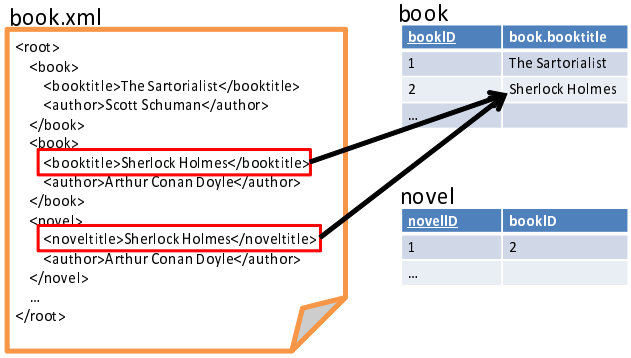

図 3　n:1 マッピングの例

ある．次に，モデル写像アプローチは XML データモデルに基づいてマップするアプローチである．このアプローチは固定したデータベーススキーマを用いるため個別の XML スキーマに依存せず，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はさらに経路アプローチ [2]，枝アプローチ [3] に分けることが出来る．最後に RRXS [4] は XML データのノード間に存在する関数従属性に基づいたマッピング手法である．マッピング時に，ユーザが各 XML データに存在する関数従属性 (詳細は 3.1 節) を与えることで，その関数従属性に従ったリレーションを生成する．

本稿で提案する C-Mapping は，以上のマッピングを全て実現することができ，さらに n:1 マッピングも実現することができることが特徴である．

3. XML の一貫性制約

本章では，C-Mapping で用いる，XML の一貫性制約について説明する．具体的には論文 [4] で定義された XML における関数従属性である XML Functional Dependencies と，それを拡張した制約である XFD+，そして XML における包含従属性である XML Inclusion Dependencies について説明する．

3.1 XML Functional Dependencies

XML Functional Dependencies (以下 XFD) は論文 [4] で定義された XML における関数従属性である．一般に関数従属性とは RDB における一貫性制約である．RDB における関数従属性 $A \rightarrow B$ とは，リレーションスキーマ $R(\ldots, A, \ldots, B, \ldots)$ がある時，その任意のインスタンス中の任意の 2 タプルに関して，もしその A の値が等しいならば B の値も必ず等しいという制約を表す．この時，A を**決定子**，B を**被決定子**と呼ぶ．

仮想的な属性	概要
n/@nodeName()	ノードnのlocal-name
n/@path()	ノードnへのルートからのパス
n/@pre()	ノードnの開始バイトオフセット
n/@post()	ノードnの終了バイトオフセット
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
n/@pathID()	ノードnへのルートからのパスの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

図 4　XFD+における仮想的な属性の例

XFD は，リレーションの属性間の関数従属性ではなく，XML のノード間の関数従属性である．XFD は XML パス式を用いて表記する．例えば book.xml (図 2 左) において book 要素のノードが決まれば booktitle 要素のテキストが決まる，という XFD を表記したい場合は次のように記述する．

for $\$x$ in book.xml/root/book　　…(1)

$\$x \rightarrow \$x$/booktitle/text()　　…(2)

まず (1) で in 以下の XPath 式により抽出した book 要素ノード集合から，for 文にて一つずつ book 要素ノードを取り出し変数$\$x$ に束縛する．次に (1) で定義した変数$\$x$ を用いて，(2) で各ノードに対する関数従属性を記述している．

3.2　XFD+

通常の XFD では，決定子と被決定子は，対象となる XML に存在するノード (要素，属性，テキスト) である．本稿では，決定子と被決定子に，ノードの属性の自然な拡張として，計算で求められる仮想的な属性の使用を許可した制約のクラスを XFD+と呼ぶことにする．これは “@” の後ろに関数形式 (関数名+“()”) で指定する．図 4 に仮想的な属性の例を示す．

例えば，book.xml (図 2 左) において要素のパスが決まれば要素名が決まるという制約は，XFD+によって次のように表記する．

for $\$x$ in book.xml//*

$\$x$/@path() $\rightarrow$ $\$x$/@nodeName()

また，book 要素ノードと book 要素ノードを始点とするエッジが決まれば子供のノードが決まるという制約は，XFD+によって次のように表記する．

for $\$x$ in book.xml/root/book

for $\$e$ in $\$x$/@edges()

$\$x$, $\$e \rightarrow$ $\$e$/@childNode()

このように図 4 の仮想的な属性の使用を許可することで，XFD では対応できなかった XML の関数従属性に対応できる．

3.3　XML Inclusion Dependencies

XML Inclusion Dependencies (以下 XIND) は XML における包含従属性である．一般に，RDB における包含従属性 $R[A] \supseteq S[B]$ とは，リレーションスキーマ $R(\ldots, A, \ldots)$, $S(\ldots, B, \ldots)$ がある時，B の値は全て A の値に含まれていなければならないという制約を表す．本稿で扱う XML データを対象とした包含従属性 XIND は，XML の要素が持つテキストあるいは属性値間の包含従属性であり，2 つの XML パス式を用いて表記する．例えば，book.xml (図 2 左) の noveltitle 要素のテキストは全て，booktitle 要素のテキストに含まれていなければならない，という XIND は次のように表記する．

book.xml/root/book/booktitle/text()

$\supseteq$ book.xml/root/novel/noveltitle/text()

本稿で扱う XIND は，論文 [8] で使われている包含従属性において，属性を一つに限定したものと等価である．

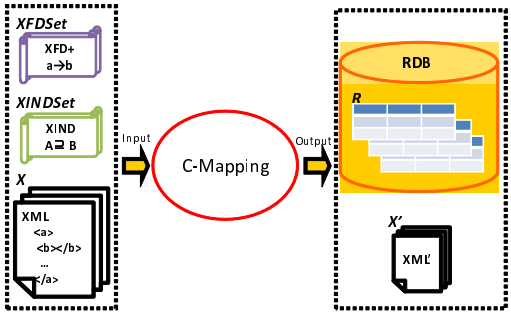

図 5　マッピング手法の入出力

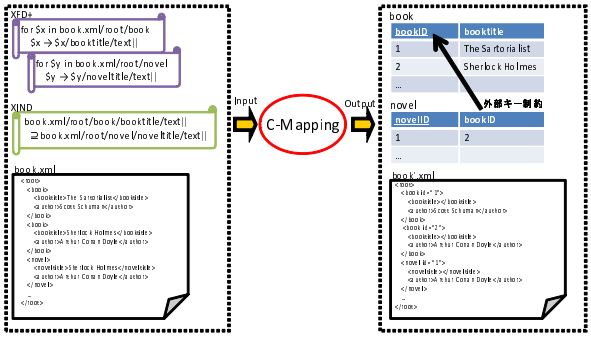

図 6　図 3 のマッピングを行う入出力

4.　C-Mapping

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．まず C-Mapping の概要を述べ，次に C-Mapping のマッピング手法について述べる．次にインスタンスの構築の問題について述べ，最後に C-Mapping の制限について述べる．

4.1　概　　要

C-Mapping の入出力を図 5 に示す．入力は，XML データの集合 X，X に関する XFD+の集合 $XFDSet$，そして X に関する XIND の集合 $XINDSet$ である．これらには次の制約が存在する．すなわち，$XINDSet$ に含まれる包含従属性の XML パス式で参照されるデータは全て，$XFDSet$ 中の関数従属性の XML パス式で参照されていなければならない．

出力は，X をマッピングした結果であるリレーションの集合 R と XML テンプレート集合 X' である．C-Mapping では XML データの全てを RDB にマップする事は行わず，XFD+で指定されたデータのみを RDB にマッピングし，マッピングされなかった部分は XML テンプレート集合 X' として出力する．XML テンプレートは，XFD+で指定された被決定子のデータ

を除外したものである．ただし決定子が要素ノードの場合は．その要素に id 属性を付与する．この XML テンプレートを用いて，マッピング後の主キー属性値と XML テンプレートの決定子のデータを照合することで，XML データへの復元処理が可能である．

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 6 に示す．このように，C-Mapping では，XIND を用いることで既存手法では実現できなかった n:1 マッピングを実現できる．

C-Mapping は次の手順で処理を行う．(1) $XFDSet$ と $XINDSet$ を用いてデータベーススキーマ RS を生成する．(2) X を，スキーマ RS に従うインスタンス I と，XML テンプレート集合 X' に変換する．

4.2 C-Mapping におけるマッピング

上記 (1) は次のステップに分けられる．

ステップ 1 入力として与えられた $XFDSet$ を用いて，データベーススキーマ RS' を生成する．

ステップ 2 入力として与えられた $XINDSet$ を用いて，RS' を $XINDSet$ を考慮したデータベーススキーマ RS に変形する．

次から各ステップについて具体的に説明する．

ステップ 1． RS' は入力 $XFDSet$ の各 XFD+に従って生成される．ステップ 1 の処理の手順を図 7 に示す．これは，与えられた $XFDSet$ の各 XFD+に対して，XFD+の各関数従属性の決定子をリレーションの主キーとし，被決定子をそのリレーションの属性としたリレーションスキーマを生成する処理を行う．

例として図 2 の book.xml に関する次の $XFDSet$ が与えられたとする．

book.xml に関する $XFDSet = \{xfd_1, xfd_2\}$

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

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

上記の $XFDSet$ を与えると，図 7 に従って処理が行われる．7〜16 行目で各 XFD+に対して処理を行う．まず xfd_1 に対して処理を行う．9〜14 行目で xfd_1 の関数従属性の決定子である book 要素を主キー属性に，被決定子である booktitle 要素をそのリレーションの属性とした新しいリレーションスキーマを作成し，15 行目でデータベーススキーマに加える．次に xfd_2 に対して同様の処理を行う．最終的に上記の $XFDSet$ を与えて生成されるデータベーススキーマは次のようになる．

book(<u>bookID</u>, booktitle)
novel(<u>novelID</u>, noveltitle)

このように，決定子あるいは被決定子のノードがテキストノードの場合，リレーションスキーマにおける対応する属性名は，そのテキストノードの親要素名とする．この属性に格納する値はテキストである．決定子あるいは被決定子のノードがテキストノードでは無い場合，リレーションスキーマにおける対応する属性名は，要素名+“ID”とする．この属性に格納する値は，本手順で独自に割り当てるノードの ID である．ノードの

```
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.          end if
13.          rs.union({res});
14.      end for
15.      RS'.union(rs);
16.  end for
17.  return RS';
18. }
```

図 7　ステップ 1 の手順

ID の割振り方は複数考えられるが，本手法では対応する XML データを深さ優先順で走査し，出現順に 1 から割り振る．また，XFD+の仮想的な属性 (図 4) を用いた場合は，その指定された関数を用いて計算した結果を格納する属性を用意する．

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**を利用して XML データの包含従属性を維持できるよう，データベーススキーマ RS' を RS に変形する．外部キー制約は包含従属性の一種である．すなわち，2 つのリレーションスキーマ $R(\underline{K_R}, \ldots, A, \ldots)$，$S(\underline{K_S}, \ldots, B, \ldots)$ がある時，外部キー制約 $R[K_R] \supseteq S[B]$ は，属性 B の全ての値が，属性 K_R の値に必ず存在していなければならないという制約を表す．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，SQL データベースでサポートされる外部キー制約には，参照先の属性が主キーもしくは一意キー (unique key) でなければならないという制限が存在する [7]．したがって，$R[A] \supseteq S[B]$ のように A が主キーでは無い一般の包含従属性は，SQL データベースではサポートしていない．本稿で扱う XIND $e_1 \supseteq e_2$ の両辺は XPath で表現された XML データの要素集合あるいは属性集合であるが，マッピング後のリレーションにおける包含従属性 $U[A_{e_1}] \supseteq V[A_{e_2}]$ における属性 Ae_1 が必ずしも SQL における一意キーとは限らない．したがって，単純な方法では，与えられた XIND を SQL データベースにおける外部キー制約で実装することはできない．

したがって，本手法では，与えられた XIND に対応するリレーションスキーマ上での包含従属性が $R[A] \supseteq S[B]$ である時，外部キー制約 $R[K_R] \supseteq S[K'_R]$ に置き換える．ただし，K'_R は，K_R と同等の属性を S に追加したものである．

これが可能な条件は，S に K'_R を追加したと仮定した時，リレーション $T = R \bowtie_{K_R = K'_R} S$ において $\forall t \in T(t[A] = t[B])$ が成立する事である．この時，K_R が主キーであることから R において $K_R \to A$ であり，S において $K'_R \to B$ である．したがって，$R[K_R] \supseteq S[K'_R]$ ならば $R[A] \supseteq S[B]$ を満たす．

したがって，ステップ 2 では次の二つを行う．(1) 上記の条件を満たすように，S の属性に K'_R を追加する．(2) 冗長性を除去するため，リレーション S から属性 B を除去する (除去された属性が後に必要になった場合には，その時に T を計算して A を参照する)．

```
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. }
```

図 8　ステップ 2 の手順

ステップ 2 の具体的な処理手順を図 8 に示す．ここでは，7, 8 行目で XIND の両辺に対応するリレーション属性を求め，9 行目で (1)，10 行目で (2) の処理を行っている．

例えば，3.3 節の XIND がステップ 2 に与えられると，図 6 の出力リレーションと同じデータベーススキーマが生成される．

4.3　リレーションインスタンスの構築の問題

4.2 節の手順で生成したスキーマに従うインスタンスは，各スキーマに対して，そのスキーマの基となった XFD+を用いて構築する．具体的には，XFD+の決定子の XPath 式と被決定子の XPath 式を XML データに適用して，その結果から各タプルを作る．

ただし，ステップ 2 における外部キー制約への書き換えを行う際には，S の冗長な属性 B を除去する前に S の各タプルに対して，追加された属性 K'_R の値を求める必要がある．これは次のように行う．すなわち，S の各タプルに対して，S のタプルと R のタプルを比較し，もし $S[B]$ の値と $R[A]$ の値が同じであれば，その R のタプルの $R[K_R]$ の値を $S[K'_R]$ に挿入する．しかし $R[A]$ の値は必ずしも一意では無いため，$S[K'_R]$ に挿入する主キー値が一意に定まらない可能性がある．例えば，図 6 の book.xml に "星の王子さま" という題名の小説と絵本それぞれが book 要素に存在し，小説版のみが novel 要素に存在する場合を考える．そしてマッピング結果の book リレーション (上記 R に該当．また，book[booktitle] が $R[A]$ に該当) と novel リレーション (上記 S に該当．また，novel[noveltitle] が $S[B]$, novel[bookID] が $S[K'_R]$ に該当) が存在すると仮定する．この時，book リレーションは図 6 の book リレーションに (3, 星の王子さま) と (4, 星の王子さま) のタプルが追加されたリレーションとなる．ここで，novel[noveltitle] が "星の王子さま" であるタプルの novel[bookID] に挿入する値は，book リレーションにおいて book[booktitle] が "星の王子さま" であるタプルの主キー値だが，該当するタプルは 2 つ存在するため，一意に定まらない．

この問題に対する対応は複数考えられる．最も簡単なものは，ユーザに問合せを行い適切な主キー値を選択してもらうというものであるが，どのような対応が最も効率的であるかは今後の課題である．

4.4　制　　限

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変換する手法を提案したが，これが適用可能なケースは，包含従属性の両辺に対応する属性が，URA [6] における Universal Instance において同一の属性と見なされ，別々に格納したときに冗長と見なされる場合のみである．例えば，あらゆる文字列を格納する属性 $string$ と，学生の名前リストを格納する属性 $name$ 間の包含従属性 $string \supseteq name$ は入力される XIND としてあり得るが，これらの情報は冗長でなく，本手法では対応できない．しかし，実用上はこのような包含従属性が与えられることは少ないと考えられる．

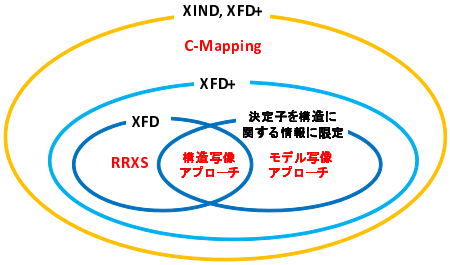

図 9　C-Mapping の記述力

5.　C-Mapping の記述力

本章では C-Mapping の記述力について議論する．C-Mapping の記述力をまとめたものを図 9 に示す．XFD があれば RRXS のマッピングを実現できることは自明である．構造写像アプローチは，決定子を**構造に関する情報**に限定した XFD で記述できる．ここで，構造に関する情報とは，ノード ID や経路情報等といった XML データの構造を表す情報 (すなわち，テキスト等のコンテンツデータではない情報) を指す．モデル写像アプローチは，決定子を構造に関する情報に限定した XFD+で記述できる．C-Mapping は，XFD+に加えて XIND をサポートすることにより，これらの全てのマッピングに加えて n:1 マッピングが可能である．次節から具体的に各マッピング手法が C-Mapping で実現できることを示す．

5.1　構造写像アプローチの実現

構造写像アプローチは各 XML データの DTD の構造に従って，DTD で定義された XML 要素をリレーションあるいは RDB 属性にマップするアプローチである．構造写像アプローチでは，ある XML 要素 a をリレーションにマップする時，マッピング結果のリレーションの主キー属性は必ず a 要素ノードの ID となる．また，マッピング結果のリレーションのそれ以外の属性は，DTD グラフにおいて a 要素ノードから到達可能なリーフノードとなる．例えば，図 2 では book 要素に対応する book リレーションの主キー属性は book 要素ノードの ID となり，book リレーションのそれ以外の属性は book 要素ノードから到達可能なリーフノードである booktitle 要素ノードとなる．ただし，author 要素ノードのように，到達可能であってもそのリレーションの属性として追加しない XML 要素もあり，詳細は各手法によって異なる．

次項からは構造写像アプローチの主な手法である Basic-Inlining, Shared-Inlining, Hybrid-Inlining が，C-Mapping で実現可能であることを示す．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
　　$\$x \rightarrow \$x/P_1/y_1$

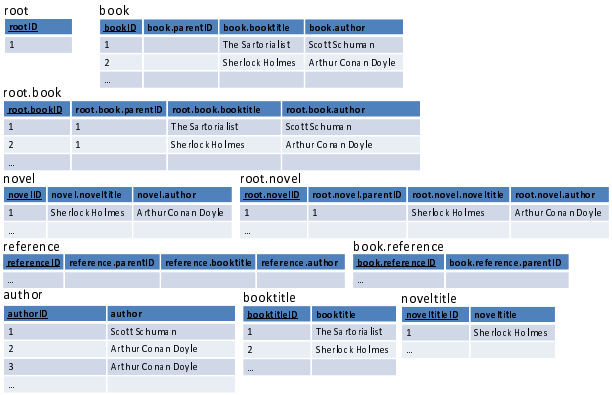

図 10　Basic-Inlining による book.xml (図 2 左) のマッピング結果

$$\$x \rightarrow \$x/P_2/y_2$$

...

$$\$x \rightarrow \$x/P_n/y_n$$

5.1.1　Basic-Inlining

Basic-Inlining は DTD で定義された全ての XML 要素をリレーションにマップする手法である．book.dtd (図 1 上) を満たした XML データ book.xml (図 2 左) をマッピングして出来るリレーションを図 10 に示す．

Basic-Inlining のマッピング手法

Basic-Inlining は DTD グラフの各要素毎に要素グラフと呼ばれるグラフを生成し，それを用いてマッピングを行う．要素グラフとは，DTD グラフの各要素ノードをルートとし，深さ優先順で探索を行うことで生成されるグラフである．例として図 1 の DTD グラフから生成される book 要素の要素グラフを図 11 に示す．図 11 の backpointer エッジとは，探索時に既に訪問済みのノードに到達した場合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マッピングを行う．(1) 要素グラフ毎にリレーションを生成する．具体的には，主キー属性が要素グラフのルート要素ノードの ID，それ以外の属性が次の 2 つに該当する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 A を生成する．(a) “ * ”ノードの子供である要素ノード，(b) backpointer エッジを持つ要素ノード．(2) (a) あるいは (b) に該当する要素を格納するためのリレーションを生成する．具体的には主キー属性が該当する要素ノードの ID，それ以外の属性が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リレーション B を生成する．(3) 親を持つ要素に対応するリレーションに親を特定するための parentID 属性を追加する．parentID に格納される値は，親となる要素ノードの ID である．

図 11 を用いて book 要素をマップする例を説明する．まず，図 11 の book 要素グラフのルート要素ノードである book 要素ノードの ID を主キー属性とし，リーフノードである author 要素ノードと booktitle 要素ノードをそれ以外の属性とした book リレーションを生成する．次に (a)，(b) に該当する reference 要素のために，新たに book.reference 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 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 10 のようになる．

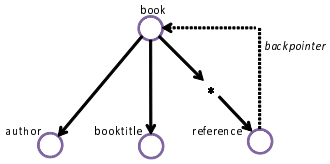

図 11　book 要素グラフ

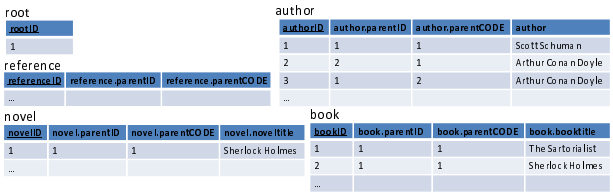

図 12　Shared-Inlining による book.xml (図 2 左) のマッピング結果

Basic-Inlining の実現

［定理 1］　C-Mapping を用いて Basic-Inlining のマッピングが実現可能である．　□

(証明)　Basic-Inlining のマッピングは要素グラフ毎にリレーション A と B を生成すれば良い．ここでは DTD で定義されたある要素 a の要素グラフ G において，(a) あるいは (b) に該当する要素 b が存在すると仮定する．

この時，リレーション A へのマッピングは次の XFD で表現できる．

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

ここで，$P_1(n)$ は，n が G において (a) あるいは (b) に該当する要素を訪問せずに 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，$P_2(n, a)$ は n が G において第 2 引数のノードの親ノードである時に真となる述語である．

また，リレーション B へのマッピングは次の XFD で表現できる．

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

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 で定義された全ての XML 要素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要素のみリレーションにマップする手法である．book.dtd (図 1 上) を満たした XML データ book.xml (図 2 左) をマップして出来るリレーションを図 12 に示す．

Shared-Inlining のマッピング手法

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，(b)“ * ”ノードの子供である要素，(c) 入次数が 2 以上の要素，(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一要素．図 1 の DTD グラフにおいて (a) は root 要素，(b) は book, novel, reference 要素，(c) は author 要素，(d) は book 要素か reference 要素のどちらかの要素，となる．

Shared-Inlining は次の手順でマッピングを行う．(1) DTD

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
…				

図 13　Hybrid-Inlining による book.xml (図 2 左) のマッピング結果

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
…				

図 14　経路アプローチによる book.xml (図 2 左) のマッピング結果

で定義された要素を (a)〜(d) に分類する．(2) 分類された要素毎にリレーションを生成する．具体的には，主キー属性がリレーションにマップする要素ノードの ID，それ以外の属性が DTD グラフにおいてマップする要素ノードから他のリレーションとなる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属性に親を特定する parentID を持つ．例えば図 1 の DTD における book 要素は図 12 の book リレーションにマップされる．これは主キー属性が book 要素ノードの ID，それ以外の属性が book 要素ノードから他のリレーションとなる要素ノードを訪問せずに到達可能な booktitle 要素ノードと parentID であるリレーションである．

Shared-Inlining の実現

［定理 2］　C-Mapping を用いて Shared-Inlining のマッピングが実現可能である．　□

(証明)　Shared-Inlining のマッピングは条件毎のリレーションを生成すれば良い．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 a に対応するリレーションへのマッピングは次の XFD で表現できる．

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

ここで，$P_4(n, a)$ は，n が D において (a),(b),(c),(d) に該当する要素を訪問せずに第 2 引数のノード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 b に対応するリレーションへのマッピングは次の XFD で表現できる．

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

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 Basic-Inlining の “マッピング結果に対する問合せが効率的である”という特徴と Shared-Inlining の “マッピング結果における冗長性が Basic-Inlining と比べ低い”という特徴の両方を兼ね備えたマッピング手法である．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とする要素の条件の (c) が除去されている．book.dtd (図 1 上) を満たした XML データ book.xml (図 2 左) をマッピングして出来るリレーションを図 13 に示す．

Hybrid-Inlining の実現

［定理 3］　C-Mapping を用いて Hybrid-Inlining のマッピングが実現可能である．　□

定理 3 の証明は，Shared-Inlining におけるリレーションとする条件の (c) が除去された以外は同じであるため，省略する．

5.2　モデル写像アプローチの実現

モデル写像アプローチは，XML のデータモデルに着目してマップするアプローチである．XML データの DTD には依存しないデータベーススキーマが生成され，任意の整形式 XML データをマッピング可能である．

次項からはモデル写像アプローチの主な手法である経路アプローチと枝アプローチが，C-Mapping で実現可能であることを示す．以下の説明中に現れる test.xml は，マッピング対象となる整形式 XML データである．

5.2.1　経路アプローチ

経路アプローチは，XML データの各ノード (要素，属性，テキスト) のそれぞれに関して，ルートからの経路 (path) と，テキスト上での文字列の位置を表すバイトオフセットによって，XML データを表現する手法である．book.xml (図 2 左) をマッピングして出来るリレーションを図 14 に示す．

経路アプローチのマッピング手法

論文 [2] における経路アプローチでは，次の 4 つのリレーションに XML データをマップする．(a) path リレーション，(b) element リレーション，(c) attribute リレーション，(d) text リレーション．(a) path リレーションは，主キー属性が pathID (ルートから各ノードへの XPath 式の ID)，それ以外の属性が pathexpression (ルートから各ノードへの XPath 式) で構成されている．(b) element リレーションは，主キー属性が docID (XML データの ID)，start (XML 要素の (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)，それ以外の属性が pathID (path リレーションにおける pathID) で構成されている．(c) attribute リレーションは，主キー属性が docID (XML データの ID)，pathID (path リレーションにおける pathID), 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)，それ以外の属性が value (XML 属性値) で構成されている．(d) text リレーションは，主キー属性が docID (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)，それ以外の属性が pathID (path リレーションにおける pathID)，value (テキスト) で構成されている．

経路アプローチの実現

［定理 4］　C-Mapping を用いて論文 [2] における経路アプローチのマッピングが実現可能である．　□

(証明)　経路アプローチのマッピングは上記の 4 つのリレーションを生成すれば良い．

(a) path リレーションへのマッピングは次の XFD+で表現で

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
...	

図 15　枝アプローチによる book.xml (図 2 左) のマッピング結果

きる．

for $n in test.xml//* | test.xml//@*

$n/@pathID() → $n/@path()

(b) element リレーションへのマッピングは次の XFD+で表現できる．

for $e in test.xml//*

test.xml/@docID(), $e/@pre(), $e/@post() → $e/@pathID()

(c) attribute リレーションへのマッピングは次の XFD+で表現できる．

for $a in test.xml//@*

test.xml/@docID(), $a/@pathID(), $a/@pre(), $a/@post()

→ $a

(d) text リレーションへのマッピングは次の XFD+で表現できる．

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 データを表現する手法である．book.xml (図 2 左) をマッピングして出来るリレーションを図 15 に示す．

枝アプローチのマッピング手法

論文 [3] における枝アプローチでは，次のリレーションに XML データをマップする．(a) Edge リレーション，(b) Vtype リレーション．

(a) Edge リレーションは，主キー属性が source (ノードの ID)，ordinal (source を始点としたエッジの ID)，それ以外の属性が 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) で構成されている．(b) Vtype リレーションは，主キー属性が vid (テキストノードまたは属性ノードの ID)，それ以外の属性が value (テキストまたは属性値) で構成されている．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．

枝アプローチの実現

［定理 5］　C-Mapping を用いて論文 [3] における枝アプローチのマッピングが実現可能である．　□

(証明)　枝アプローチのマッピングは上記のリレーションを生成すれば良い．

(a) Edge リレーションへのマッピングは次の XFD+で表現できる．

for $n in test.xml//*

for $e in $n/@edges()

$n, $e → $n/@nodeName(), $e/@childNode()/@type(), $e/@childNode()/@nodeID()

(b) Vtype リレーションへのマッピングは次の XFD+で表現できる．

for $v in test.xml//text()[./@type()=string] | test.xml//@*[./@type()=string]

$v/@nodeID() → $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 という部分を他の型に変える事で表現できる．　□

6.　まとめと今後の課題

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) 複数の XML ノードを 1 つの RDB 属性値にマップする n:1 マッピングが実現できる．(2) 主要な既存手法によるマッピングを実現できる．

今後の課題としては，本手法を実装したプロトタイプシステムの開発と性能評価がある．また，現在の手法では，入力される XIND に制限があり，タプル間の包含従属性を許していないが，この制約の緩和を検討する．

謝　　辞

本研究の一部は科学研究費補助金若手研究 (B)(#20700076) による．

文　　献

[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 and S. Uemura: “ XRel: A path-based approach to storage and retrieval of XML documents using relational databases ”, ACM Transactions on Internet Technology(TOIT), 1(1): 110-141, 2001.

[3] Daniela Florescu, Donald Kossmann: Storing and Querying XML Data using an RDMBS. 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] B. Box, ”The XML Data Model,” 1997, http://www.w3.org/XML/Datamodel.html.

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

[7] ISO/IEC 9075:1999, “ Information Technology-Database Language-SQL-Part 1-5 ”1999.

[8] Michael Karlinger, Millist W. Vincent, Michael Schrefl: Inclusion Dependencies in XML: Extending Relational Semantics. In Database and Expert Systems Applications, volume 5690 of Lecture Notes in Computer Science: 23-37, 2009.