データ変換のためのスキーママッチング支援機構

大河原俊明† 田中 淳一†† 森嶋 厚行††† 杉本 重雄†††

† 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 図書館情報専門学群 〒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
E-mail: †{okwr,m188,mori,sugimoto}@slis.tsukuba.ac.jp

あらまし 我々は，近年重要度が増加しているデータ変換の問題に着目している．データ変換のためには，スキーママッチングを行う必要がある．スキーママッチングとは，変換元のスキーマと変換先のスキーマの構成要素間を対応付けることである．既存のアプローチとは異なり，我々のアプローチでは直接スキーママッチングを行うのではなく，まず変換元と変換先の各スキーマからそれぞれ概念モデルを抽出して，それらの概念モデルの構成要素間のマッチングを行う．ポイントは，概念モデルを抽出することによりスキーマの各構成要素のグルーピングを行い，直接スキーママッチングを行うよりも可能な対応関係の数を減らすことである．

キーワード データ変換，スキーママッチング，概念モデル，問合せ生成

A Support Mechanism for Schema Matching Processes in Data Transformations

Toshiaki OHKAWARA†, Jun'ichi TANAKA††, Atsuyuki MORISHIMA†††, and Shigeo SUGIMOTO†††

† Grad. Sch. of Library, Information and Media Studies, Univ. of Tsukuba,
1–2 Kasuga, Tsukuba, Ibaraki, 305–8550 Japan
†† Sch. of Library and Information Science, Univ. of Tsukuba,
1–2 Kasuga, Tsukuba, Ibaraki, 305-8550 Japan
††† Grad. Sch. of Library, Information and Media Studies/Research Center for Knowledge Communities, Univ. of Tsukuba,
1–2 Kasuga, Tsukuba, Ibaraki, 305-8550 Japan
E-mail: †{okwr,m188,mori,sugimoto}@slis.tsukuba.ac.jp

Abstract We focus on the problem of data transformations whose importance is increasing in recent years. In the context of databases, data transformations require schema matching. Schema matching is the process of matching components of the source and target schemas. Existing approaches require the user to give, or try to find, direct relationships between database schema components. In contrast, our approach first extracts conceptual schemas from the database schemas and then proceeds to matching of components. A point here is that, the extraction of conceptual schemas results in grouping of attributes of database schemas, which reduces the search space for finding appropriate relationships between the schemas.

Key words data transformations, schema matching, conceptual schemas, query generation

1. はじめに

我々は，近年重要度が増加しているデータ変換の問題に着目している．ここで言う**データ変換**とは，2 つのデータベーススキーマ S_A と S_B，および S_A のインスタンス I_A が与えられたとき，データ変換 F を用いて $I_B = F(I_A)$ を求めることである．データ変換の典型的な例としては，旧システムのデータベースからの新システムのデータベースへのデータの移行，業界標準

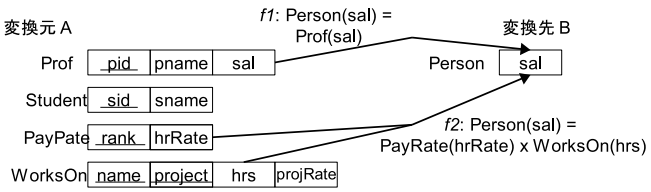


図 1　Clio による対応付け

の XML スキーマに従ってデータベース中のデータを出力するための変換，Web サービスを利用するために必要な XML データ変換などが挙げられる．

データ変換 F は，しばしばデータベースに対する問合せとして実装される．この問合せの作成は労力を要するため，問合せの作成を半自動的に行う**データ変換支援システム**の研究開発が行われている．一般にスキーマ S_A と S_B だけから変換 F(を実装した問合せ) を求めることは不可能であるため，既存のデータ変換支援システムではスキーマの情報に加え，スキーマの構成要素間の**対応付け**を入力することが多い．図 1 は，データ変換支援システムの 1 つである IBM の Clio [7] を用いた例である．この例は，ある大学で利用されているデータベースから Prof と Student の給与 (sal) を求めるためのデータ変換である．この変換を行うために Clio の利用者が指定する対応付けは，図 1 の $f1$ と $f2$ である．Clio における対応付けは，**value correspondence** と呼ばれる．各 value correspondence は次の 2 つの情報を表す．(1) 変換元の属性と変換先の属性の対応，(2) 変換先属性の値を計算するための式．例えば，図 1 の $f1$ は (1) 変換先の Person(sal) 属性と変換元の Prof(sal) 属性が対応し，(2) Person(sal)=Prof(sal) で計算できることを表す．また，$f2$ は (1) 変換先の Person(sal) 属性が変換元の PayRate(hrRate) 属性および WorksOn(hrs) 属性に対応し，(2) Person(sal)=PayRate(hrRate)×WorksOn(hrs) で計算できることを表す．

以上のように，異なるスキーマの構成要素間の対応付けを求める操作を，一般に**スキーママッチング** [9] と呼ぶ．

実は，次の理由によりスキーママッチングを行うことはそれほど簡単ではない．(1) スキーマの構造の深い理解が必要である．この例では Prof の給与を発見するのは簡単であるが，Student の給与が PayRate(hrRate)×WorksOn(hrs) で計算できることを知るためには，リレーション Student，PayRate および WorksOn の間に外部キー制約が存在することを利用者が知っていなければならない．(2) 属性数が増加すると急激にマッチングが困難になる．変換元と変換先の属性数をそれぞれ m，n とすると，対応付けの可能性は $2^m \times n$ 通りある．しかも，$f2$ のような計算式を伴う value correspondence の自動的な発見は自明ではない．(3) 既に説明したように，各 value correspondence は対応付けと計算式を同時に指定するが，この 2 つは本来は完全に独立しているものである．例えば，図 1 の $f2$ の value correspondence は，変換先の Person(sal) には変換元の「学生の給与」が対応する，ということと，その給与は PayRate(hrRate)×WorksOn(hrs) によって計算される，という 2 つの話を一度に指定しなければならない．このため，利用者は個々の問題に関心を集中しにくい．

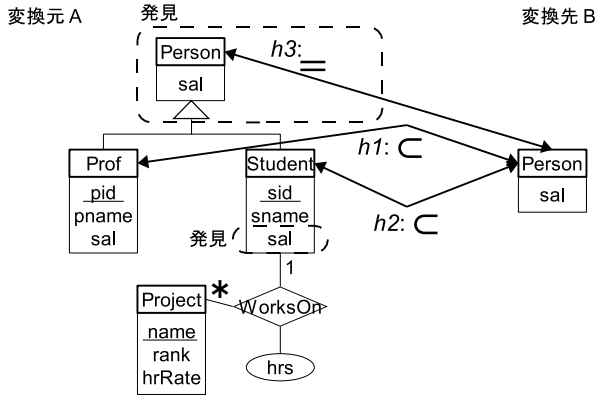


図 2　SMART による対応付け

我々は，XML データを対象としたデータ変換支援システム SMART を開発中である [2] [8]．SMART でもスキーママッチングが必要であるが，これまで開発してきたシステムはスキーママッチング機能を持っていなかった．本稿では，新たに開発した，スキーママッチングをより簡単に行えるように支援するための，SMART におけるスキーママッチング支援機構について説明する．基本的なアイデアは，直接スキーマ間のマッチングを行うのではなく，まず各スキーマから概念モデルを抽出し，それを用いたマッチング支援を行うことである．

与えられた対応付けからデータ変換問合せを生成する手法については別稿に譲る．

2. 基本的な考え方

SMART のスキーママッチング支援における基本的な考え方を説明するために，前章と同じデータ変換の例を用いる．SMART を用いた場合の操作は次のようになる．

(1) システムはスキーマを直接利用者に見せるのではなく，まず各スキーマから概念モデルを抽出し，その概念モデルを利用者に提示する (図 2 の点線で囲まれていない部分)．各概念モデルはクラス図として表現される．

(2) 利用者は，提示された概念モデルのクラス間に対して**ラベル付き対応関係**を与える (図 2 の $h1$ および $h2$)．この例の場合，Person クラスのインスタンス集合は Prof クラス，Student クラスのインスタンスを包含しているので，$h1$ および $h2$ のラベルとして “⊂” を付ける．ラベルとして “=” を持つクラス間の対応関係を特に**等価な対応関係**と呼ぶ．

(3) システムはこれらの対応関係を基に推論を行い，図 2 の点線内部の情報を発見する．この例では次が発見される．(3a) 変換元の Prof と Student にスーパクラスが存在して，それは変換先の Person と対応するはずである．これは $h1$ および $h2$ からわかる．(3b) 変換元の Student クラスは属性 sal を持つはずである．これは，変換先の Person クラスの属性と，Student クラスの属性を比較することにより，sal に対応するものがないことからわかる．

(4) システムは，新たに Student クラスに追加され

た属性 sal を計算するための計算式 (この例の場合，Student.sal=sum(this.WorksOn.[hrs×Project.hrRate]) の入力を利用者に求める．

(5) システムは以上の入力からデータ変換 F を計算する XQuery 問合せを求め，出力する．

以上の例からわかるように，我々のアプローチにおける基本的なアイデアは次の通りである．(1) 直接スキーマの構成要素間に対する対応付けを行うのではなく，まず概念モデルを抽出し，クラス間の対応関係を与える．(2) 変換先の概念モデルの構成要素に対して，1 対 1 で対応する変換元の概念モデルの構成要素が存在することを仮定し，Clio の value correspondence 等では不可分になっている属性間の対応付けと値の計算を分離する．この分離は，概念モデルを導入することにより初めて可能になる．なぜなら，直接スキーマの構成要素間で対応付けを行う場合には，2 つのスキーマの構造が全く異なる可能性があるため，1 対 1 で対応する変換元の構成要素の存在を前提とすることが困難であるからである．したがって，その場合の対応付けは，図 1 の $f2$ のように多対 1 で与えてもよいと設計せざるを得ない．

我々のアプローチは，直接スキーマの構成要素間のスキーママッチングを行うアプローチと比べて次のような利点がある．(1) 概念モデルの抽出により，利用者がデータの内容を理解することを支援する．また，変換元と変換先の概念モデルはある程度類似していることが期待されるため，対応付けを比較的行いやすいと考えられる．(2) 最初に属性ではなく，クラス間の対応関係を与える．後述するように，概念モデルの利用と 1 対 1 の対応関係を前提とすることによって，その後の属性間の対応付けが容易になる．(3) 対応関係と計算式の指定が分離されており，それぞれを個別に処理できる．例えば，システムは変換先の Person クラスの sal に変換元の「学生の給与」が対応することを発見する．利用者はそれを計算するための式を入力する．

2.1 スキーママッチングの問題の形式化

スキーママッチングの問題は結合演算のアナロジで捉えることができる [9]．本稿では，スキーママッチングの問題を次のように定義する．

定義． 変換元スキーマ S に含まれる属性集合を A_S，変換先スキーマ T に含まれる属性集合を A_T とする．このとき，スキーママッチングの問題とは，A_S のべき集合と A_T の直積 $\mathcal{P}(A_S) \times A_T$ の各要素に対し**マッチ数** 1 か 0 を割り当てることである．□

$\mathcal{P}(A_S) \times A_T$ を表に表したものを**マッチ表**と呼ぶ．属性数 $|A_S| = m$，$|A_T| = n$ のとき，マッチ表のサイズは $2^m \times n$ となる．スキーママッチングの結果 (各組合せにマッチ数を割り当てた結果) を $MATCH_{S,T}$ で表す．

図 3 は，図 1 に示す変換元および変換先のスキーマに対してスキーママッチングを行った結果 $MATCH_{A,B}$ の一部である．属性 C はマッチ数を表す．1 が割り当てられている場所は，その行が表す変換元の構成要素群が，変換先の構成要素に対応付けられることを表す．

変換元	C	変換先
{pid}	0	sal
{pname}	0	sal
{sal}	0	sal
	⋮	
{pid, pname}	0	sal
{pid, sal}	0	sal
	⋮	
{sal, hrRate, hrs}	1	sal
	⋮	
{pid, ..., projRate}	0	sal

図 3　マッチ表の例

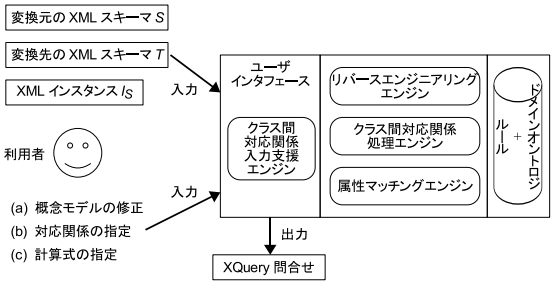


図 4　アーキテクチャ

2.2 関連研究

Cupid [6] は，2 つのスキーマの構成要素間のマッチングを自動的に行うシステムである．スキーマの属性名のマッチングとスキーマの構造を利用したマッチングを組み合わせたアルゴリズムを採用している．ただし，直接のスキーママッチングでは避けられない多対 1 のマッチングには対応していない．Cupid のアルゴリズムは我々のアプローチと組み合わせて利用することが可能であり，補完する関係にあると考えられる．

データベースからの概念モデル抽出はデータリバースエンジニアリング技法 [1] として知られており，DB-MAIN [4] をはじめとして多くの研究が行われている．本システムはこれらの技法と組み合わせて利用することを想定している．

3. データ変換支援システム SMART

図 4 に本システムのアーキテクチャを示す．本システムは入力として変換元の XML スキーマ S と変換先の XML スキーマ T，および S の XML インスタンス I_S をとり，$I_T = F(I_S)$ を計算する XQuery 問合せを出力する．本システムの主要な構成要素は以下の 4 つである．

リバースエンジニアリングエンジン: リバースエンジニアリングエンジンは入力した 2 つの XML スキーマ S と T からそれぞれ概念モデルを抽出する．その結果は必ずしも正しいとは限らないため，利用者から指示された概念モデルの修正 (図 4(a)) を受け付け，処理する．

クラス間対応関係入力支援エンジン: クラス間対応関係入力支援エンジンは利用者から与えられた変換先と変換元のクラス間の対応関係 (図 4(b)) のラベルを基に，等価な (ラベル “=” を持つ) 対応関係を発見する．場合によっては，図 2 の変換元の Person クラスのように新たなクラスを発見することもある．本エンジンでは，クラス間の関係を推論可能な応用論理の 1 つで

ある記述論理 (descrpition logics) [3] を応用して処理を行う．

クラス間対応関係処理エンジン: クラス間対応関係処理エンジンは，クラス図の形であらかじめ与えられたドメインオントロジ等を利用して，等価な対応関係を発見する．

属性マッチングエンジン: 属性マッチングエンジンは，入力として等価な対応関係で結ばれた 2 つのクラス $C1$ と $C2$ をとり，$C1$ と $C2$ の属性間の対応付けを出力する．

入力される等価な対応関係は利用者に直接与えられたものか，上のクラス間対応関係入力支援エンジンによって計算されたものである．属性間の対応付けを行う処理には，既存のスキーママッチングの技法 [9] を応用する．一般には完全な対応付けは求められないため，利用者は結果を修正することができる．必要であれば，2 章 (4) のように，システムは利用者に計算式の入力を求めることもある (図 4(c))．

4. マッチ表のサイズの削減

一般に，スキーマの各構成要素に対して直接スキーママッチングを行う場合には，$MATCH_{S,T}$ のサイズは $2^m \times n$ である．しかし，SMART では，概念モデルを利用すること，および変換先の構成要素に 1 対 1 で対応する構成要素が変換元に存在すると仮定することによって，このサイズを大幅に削減している．本章ではこれについて説明する．

4.1 対応関係と計算の分離による削減

2 章で説明したように，SMART では対応関係と計算式を分離しており，変換元と変換先の構成要素はすべて 1 対 1 で対応付けられる．2 章 (3) で触れた Student の sal 属性のように，変換元に「欠けている」属性を新たに追加しなければならない場合もあるが，対応を 1 対 1 に限定することにより，マッチ表のサイズは大幅に削減できる．例えば，変換元の属性集合を A_S(ただし $|A_S| = m$)，変換先の属性集合を $A_T(|A_T| = n)$ とする．このとき，A_T 中の属性に直接対応する属性が A_S 中に全くなく，すべて A_S に追加する必要があったとしても，マッチ表 $MATCH_{S,T}$ のサイズは $2^m \times n$ ではなく，$(m+n) \times n$ になる．直接対応する属性がすべて存在する場合は，$m \times n$ である．

4.2 クラス間の対応関係を利用することによる削減

SMART では，まず各スキーマ S と T からそれぞれ概念モデルを抽出し，クラス間の対応関係を求める．この情報を利用することにより，属性のマッチ表のサイズを削減できる．なぜなら，対応するクラスの属性間のマッチングだけを考えればよいからである．削除後のマッチ表のサイズは次のようになる．まず，S と T からそれぞれ抽出された概念モデルに含まれるクラスの数が共に p であるとする．また，S の概念モデルに含まれる isa 関連の数を k とする．その場合，S の概念モデルの各クラスは m/p 個の属性を持ち，T の概念モデルの各クラスは n/p 個の属性を持つ．また，クラス間の対応関係 (等価な対応関係および包含関係) の最大値は $p+k$ となる．属性間のマッチングにおいては，このクラス間の対応関係ごとに，各クラスに含まれる属性間のマッチングを行う必要がある．したがって，検討すべき属性のマッチ表のサイズは $(m/p \times n/p) \times (p+k)$ となる．これは，4.1 節の $MATCH_{S,T}$ の $(p+k)/p^2$ のサイズとなる．

```
1.  Table[変換元, C, 変換先] t = AS x { null } x AT;
2.  Set[Classes] CS = {};
3.  Set[Classes] CT = {};
4.
5.  void reduce() {
6.    CS = {cs_1, ..., cs_p} = AS.getClasses();
7.    CT = {ct_1, ..., ct_q} = AT.getClasses();
8.    Table[変換元, C, 変換先] tc = CS x { null } x CT;
9.    tc = tc.getMatch(); //MATCH_{CS,CT}
10.   t = π_{attrs(t)}(t ⋈_{class(t.AS)=tc.CS ∧ class(t.AT)=tc.CT} σ_{C=1}(tc));
11. }
```

図 5　マッチ表のサイズの削減

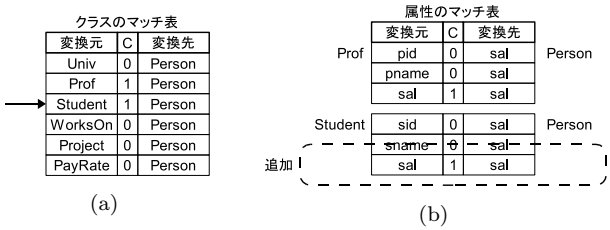


図 6　クラスのマッチ表 (a) と属性のマッチ表 (b)

図 5 はこれを抽象アルゴリズムにより表現したものである．変換元のスキーマに含まれる属性集合を A_S とし，変換先のスキーマに含まれる属性集合を A_T とする．また，$|A_S| = m$ とし，$|A_T| = n$ とする．まず 1 行目では，A_S および A_T 中の属性間のすべての組合せを表すマッチ表を作成する (サイズは $m \times n$ となる)．次に，6〜7 行目では，それぞれのスキーマから概念モデルを抽出する．各概念モデルに含まれるクラスをそれぞれ cs_i，ct_i で表す．以降，ある属性 a が属するクラスは $class(a)$ と表現する．8 行目では，変換元と変換先の概念モデルのクラス間の等価な対応関係を表現する特別なマッチ表を作成し，9 行目では，2 章における (2) の操作などを通じて，$MATCH_{CS,CT}$ を求め (この処理は getMatch() と表している)，tc に代入する．最後に，10 行目で tc と t の結合演算により，対応する可能性のない属性の組合せを t から除去する．図 6 は，図 2 の例を用いてこの処理を説明したものである．図 6(a) が $MATCH_{CS,CT}$ であり，属性のマッチ表は図 6(b)(点線部は除く) のようになる．図からわかるように，$MATCH_{CS,CT}$ において C=1 である各クラスの属性間の組合せの行だけが属性のマッチ表図 6(b)(点線部は除く) に残される．

5. 対応付けの自動発見

本手法では，概念モデルの利用，および 1 対 1 で対応する構成要素の存在を仮定するという特徴を生かし，次のような対応付けの発見支援を行うことが可能である．

(1) 記述論理を用いたクラス間の対応関係の発見: 記述論理 (description logics) を利用して，入力として与えられたクラス間の対応関係から，どのクラスの間に対応関係があるかを推論することができる．ここで発見された対応関係に応じて，クラスのマッチ表に 1 がセットされる．

(2) ドメインオントロジを用いたクラス間対応関係の発見: ク

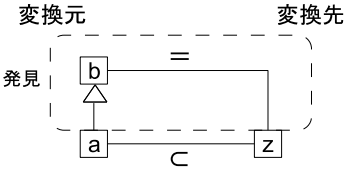
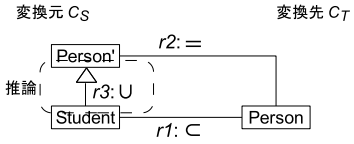


図 7　対応関係の入力によるスーパクラスの発見



(a)

($r1$) Student $\sqsubseteq$ Person
($r2$) Person' $\equiv$ Person
↓ 推論
($r3$) Student $\sqsubseteq$ Person'

(b)

図 8　クラス間対応関係 (a) と記述論理表記 (b)

ラス図の形で与えられたドメインオントロジ等を利用して，クラス間の対応関係を発見できる．例えば，図 2 では Student と Person の対応関係はユーザによって与えられたが，もしドメインオントロジにこの関係が記述されていれば，それを利用することができる．システムは発見された対応関係を用いて，4 章で説明したクラスのマッチ表に 1 を追加する (図 6(a) の矢印)

(3) 欠けているクラス，属性の発見: SMART では，変換先の概念モデルの構成要素に 1 対 1 で対応する変換元の構成要素が存在しなければならないという制約がある．このため,「欠けている」クラスや属性の発見が比較的容易である．なぜなら,「現在，直接対応するクラス (属性) が存在しない ⇒ クラス (属性) が欠けている」という推論が可能であるからである．多対 1 の対応関係を許すと，この推論は不可能である．

欠けているクラスの発見は，(1) と同様に記述論理を応用して行われる．単純な例を図 7 で示す．これは，変換元のクラスと変換先のクラスの対応関係から，変換元のクラスに欠けている新しいクラスを発見している．図 2 の変換元の Person クラスはこのルールを用いて発見されている．

欠けている属性の発見は，ドメインオントロジを用いて行われる．図 2 を例にすれば，変換元の Student の属性としてはもともと sid，sname があるが，これらはいずれも Person の sal には対応付けできないとドメインオントロジを用いて判定されたとする．さらに，必ず 1 対 1 で対応付けられる属性が変換元に存在すると仮定されていることから，最終的に属性 sal が欠けていると判定される．ここで発見された新しい属性は属性のマッチ表に追加され，対応する組合せに対して 1 がセットされる (図 6(b) の点線内部).

6. 本システムが扱うスキーママッチングのクラス

SMART においてスキーママッチングを行う目的は，その情報を利用してデータ変換を行うことである．したがって，リバースエンジニアリングを用いて概念モデルを抽出し，対応関係を得る場合に，必ず最終的には「1 対 1」関係を求める必要がある．変換元のスキーマ S および変換先のスキーマ T の概念モデル C_S および C_T が与えられ，さらに，C_T のすべての構成要素に 1 対 1 対応する C_S の構成要素が存在する場合，C_S は C_T に**互換性がある** (compatibile である) と定義する．本論文で提案するスキーママッチング支援機構は，C_S が C_T に互換性がある場合のみを対象としている．

7. SMART におけるスキーママッチング支援機構の設計と実装

本章では，スキーママッチング支援機構の設計と実装について説明する．まず，図 4 における各エンジンのポイントについて説明し，次にプロトタイプシステムの実装の詳細について説明する．

7.1 クラス間対応関係入力支援エンジンにおけるクラス間対応関係の導出

クラス間対応関係入力支援エンジンでは，利用者に入力として与えられたクラス間の対応関係から，他のクラス間の対応関係を推論する．これは記述論理を用いて行われる．記述論理の推論には，記述論理エンジンである RACER [5] を用いる．

本エンジンでは，GUI によって与えられた対応関係を記述論理表記に変換し，RACER に投入する．この変換は次のように行われる (図 8 の例を用いて説明する).

（1） 変換元スキーマ S の概念モデル C_S，変換先スキーマ T の概念モデル C_T，これらの概念モデルのクラス間の対応関係を記述論理表記に変換する (例えば，図 8 の $r1$ および $r2$).

（2） 変換された記述論理表記の式を RACER に投入し，推論結果を得る (図 8(b) の $r3$).

（3） (2) で得た結果を変換元の概念モデル C_S に反映させる (図 8(a) の $r3$).

7.2 リバースエンジニアリングエンジンにおける概念モデルの修正

SMART では，概念モデル抽出の過程で，既存のデータリバースエンジニアリング技法を用いることを仮定しているが，一般にデータリバースエンジニアリング技法の結果は必ずしも正しいとは限らない．なぜなら，データベーススキーマには，必ずしもすべての制約が記述されているとは限らないからである．また，データベースインスタンスから制約を推論するにしても，一般には有限のインスタンスからすべての制約を導出することはできない．このため，SMART のスキーママッチング支援機構では，利用者による明示的な概念モデルの修正機能を提供する．例えば，キー制約の指定による自動修正機能がある (図 9)．リバースエンジニアリングの結果が図 9(a) であるとする．このとき，実際にはプロジェクトの hours は学生がそのプロジェクトに何時間従事したかを表す属性なので，Student クラスと Project クラス間の関連の属性になるべきである (図 9(b))．本システムでは，利用者は Project のキーが $name$ であると指定する．するとシステムは，Project のインスタンスを調べ，Project の各属性 A_i について関数従属性 $name \rightarrow A_i$ が成立するかを調べる．もし成立しないならば，A_i をインスタ

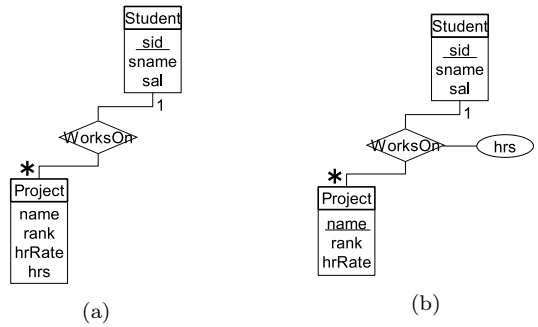


図 9　自動修正前 (a) と自動修正後 (b)

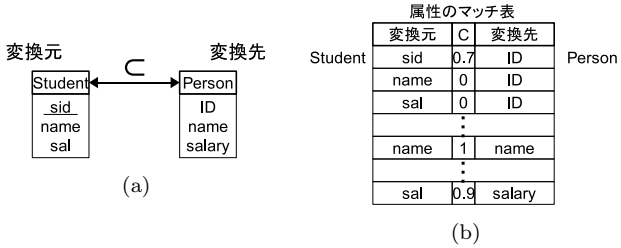


図 10　対応関係の例 (a) とその場合の属性のマッチ表 (b)

ンスに矛盾しない位置に移動する．この例では，$name \rightarrow hrs$ が成立せず，$sid \rightarrow hrs$ も成立しないが，$name, sid \rightarrow hrs$ が成立するため，Student と Project の間の関連属性とする (図 9(b))．

7.3　属性マッチングエンジンにおける属性マッチング候補の発見

属性マッチングエンジンでは，対応するクラスの属性間でのマッチングを行う．前述したとおり，このマッチングは既存のスキーママッチング技法を応用するが，一般には完全な結果が得られない．このため，システムによる自動的なスキーママッチングの結果のマッチ数の値として，0 から 1 の間の任意の値をとることにする (図 10(b))．1 に近いほど，マッチングの可能性が高いことを表す．この値は GUI 上ではマッチングされた属性間に引かれる線の色として表現され，利用者が参考にしやすいように工夫される．

7.4　クラス間対応関係処理エンジンにおけるクラス間対応関係の発見

クラス間対応関係処理エンジンでは，クラス間の対応関係を発見する．これには，5 章 (2) で説明したように，あらかじめ用意されたドメインオントロジ等を利用する．7.3 節と同様に，マッチ数に 0 から 1 の任意の値をとることができ，GUI 上ではクラス間に引かれる線の色として表現する．

7.5　プロトタイプシステムの実装の詳細

本節では，SMART プロトタイプシステムの実装の詳細について説明する．本プロトタイプシステムでは，図 4 のアーキテクチャを図 11 のように実装している．本プロトタイプシステムは，大きく分けて，SMART 本体部と GUI 部から構成される．

7.5.1　SMART 本体部の実装

SMART 本体部は，図 4 で示した 3 つのエンジンに対応する 3 つのコンポーネントと XQuery 問合せを生成するためのコンポーネントで構成されている．

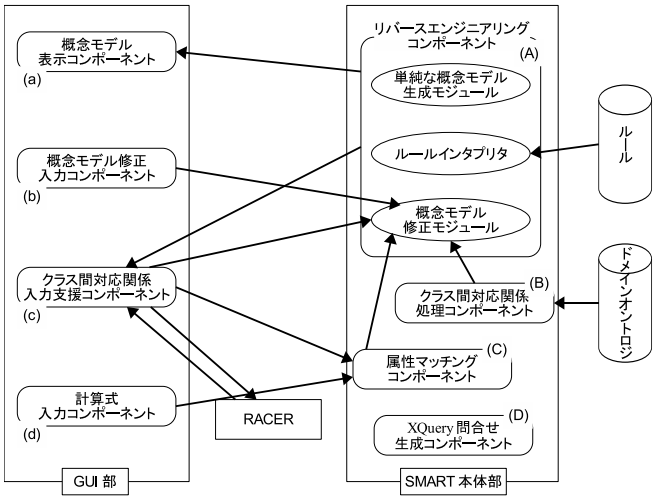


図 11　SMART プロトタイプシステムの実装

(A) リバースエンジニアリングコンポーネント: リバースエンジニアリングコンポーネントは，単純な概念モデル生成モジュールと，ルールインタプリタおよび概念モデル修正モジュールにより構成されている．

- **単純な概念モデル生成モジュール**は入力された XML スキーマと 1 対 1 対応するような概念モデルを生成する．XML スキーマと 1 対 1 対応する概念モデルとは，XML スキーマにおける要素が概念モデルのクラスに，XML スキーマにおける属性が概念モデルの属性に，それぞれ 1 対 1 対応した概念モデルのことである．
- **ルールインタプリタ**は単純な概念モデル生成モジュールで生成された XML スキーマと 1 対 1 対応する概念モデルを，リポジトリ中のルールに従って修正する．
- **概念モデル修正モジュール**は SMART 本体部や GUI 部の各コンポーネントから概念モデル修正の命令を与えられ，それに応じて，概念モデルを修正する．

(B) クラス間対応関係処理コンポーネント: 5 章 (2) で説明したように，クラス間の対応関係の発見はクラス図の形で与えられたリポジトリ中のドメインオントロジ等を利用することが考えられる．本コンポーネントで発見した対応関係は，SMART 本体部の概念モデル修正モジュールに渡す．現バージョンのプロトタイプシステムでは未実装である．

(C) 属性マッチングコンポーネント: GUI 部のクラス間対応関係入力支援コンポーネントから等価な対応関係で結ばれた 2 つのクラス $C1$ および $C2$ が与えられると，$C1$ および $C2$ の属性間の対応付けを行う．また，対応付けの結果によっては図 2 の Student クラスの sal 属性のように新たに変換元の概念モデルに属性を追加する必要があるので，その情報をリバースエンジニアリングコンポーネントの概念モデル修正モジュールに渡す．

(D) XQuery 問合せ生成コンポーネント: 変換元の XML インスタンスを，変換先の XML スキーマに従った XML インスタンスにデータ変換するための XQuery 問合せを生成する．

7.5.2　GUI 部の実装

SMART における GUI 部は，図 4 で示した 1 つのエンジン

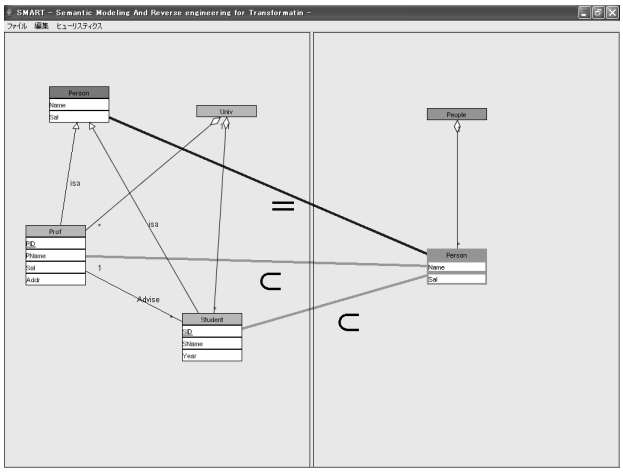


図 12　SMART のプロトタイプシステム

に対応する 1 つのコンポーネントを含む次の 4 つのコンポーネントで構成されている．

(a) 概念モデル表示コンポーネント: SMART 本体部のリバースエンジニアリングコンポーネントから概念モデルが与えられると，利用者に概念モデルを表示する．概念モデルが修正された場合など，概念モデルに変更が生じたときは，このコンポーネントにより利用者に変更された概念モデルを表示する．

(b) 概念モデル修正入力コンポーネント: 利用者から概念モデル修正の命令が与えられると，SMART 本体部の概念モデル修正コンポーネントに渡す．修正された概念モデルは，概念モデル表示コンポーネントにより，利用者に表示する．

(c) クラス間対応関係入力支援コンポーネント: 7.1 節で説明したように，SMART では，記述論理を用いた推論に既存の記述論理処理エンジンである RACER を利用している．本コンポーネントは，SMART 本体部のリバースエンジニアリングコンポーネントから概念モデルの情報，利用者からラベル付き対応関係の情報がそれぞれ与えられると，それらを記述論理表記に変換し，RACER に投入する．その結果を，SMART 本体部の概念モデル修正コンポーネントに渡す．

(d) 計算式入力コンポーネント: 利用者に計算式の入力を求める．入力された計算式は SMART 本体部の属性マッチングコンポーネントに渡す．

7.6　プロトタイプシステム

前節までの設計に基づき，SMART のプロトタイプシステムを開発した．記述言語は Java であり，Java のクラス数は約 30，約 10,000 行のコードで構成される．入力となる XML スキーマは RELAX NG で記述する．

図 12 は本プロトタイプシステムの実行画面である．この実行画面は，利用者がラベル付き対応関係の入力をし，SMART が記述論理を用いて推論した結果を示している．

8. おわりに

本稿では，データ変換のためのスキーママッチング支援機構について説明した．本機構の特徴は，直接スキーマ間のマッチングを行うのではなく，まずそれぞれのスキーマから概念モデルを抽出することにより，マッチングを容易にすることである．今後の課題としてはある程度の規模を持つスキーマを用いた適用実験などが挙げられる．

謝　辞

ゼミなどでご議論いただきました筑波大学図書館情報メディア研究科田畑孝一教授，阪口哲男助教授，永森光晴講師に感謝いたします．本研究の一部は日本学術振興会科学研究費補助金若手研究 (B)(課題番号 15700108) による．

文　献

[1] P. Aiken. Data Reverse Engineering Staying the Legacy Dragon, McGraw-Hill, Inc., New York, 1996.

[2] 古川夏子，上村匡稔，大河原俊明，森嶋厚行，杉本重雄．XML データからの意味情報抽出支援プロトタイプシステムの実装．日本データベース学会 Letters，Vol.3, No.2, pp.129-132，2004 年 6 月．

[3] Benjamin N. Grosof, Lan Horrocks, Raphael Volz, Stefan Decker: Description Logic Programs: Combining Logic Programs with Description Logic. WWW 2003: 48-57.

[4] Jean-Luc Hainaut: Research in Database Engineering at the University of Namur. SIGMOD Record Vol.32, No.4, December 2003: 124-128.

[5] Volker Haarslev, Ralf Möller: Description of the RACER System and its Applications. International Workshop on Description Logics (DL-2001).

[6] Jayant Madhavan, Philip A. Bernstein, Erhard Rahm: Generic Schema Matching with Cupid. VLDB 2001: 49-58

[7] Renée J. Miller, Laura M. Haas, Mauricio A. Hernández: Schema Mapping as Query Discovery. VLDB 2000: 77-88.

[8] 大河原俊明，森嶋厚行，杉本重雄．ユーザ定義ルールを用いた XML データからの概念モデル抽出．日本データベース学会 Letters，Vol.3，No.2，pp.53-56，2004 年 9 月．

[9] Erhard Rahm, Philip A. Bernstein: A survey of approaches to automatic schema matching. VLDB J. 10(4): 334-350, 2001.