A6-5

Schemalet: XML データとスキーマの翻訳モデリング言語

倉光君郎
東京大学大学院情報学環
113-0033 東京都文京区本郷 7-3-1
TEL: 03-5841-2483 / FAX: 03-5841-8459
kuramitsu@iii.u-tokyo.ac.jp

概要

今日、異なった組織間の XML データとそのスキーマのセマンティック相互運用性に対する必要性が高まっている。本論文では、セマンティック相互運用性に対する翻訳アプローチを報告する。Schemalet は、ドメインタイプと相互関係によって、XML データとスキーマを再構成するモデリング言語である。Schemalet を用いれば、情報システムは自動的にデータとアプリケーションスキーマの意味的な対応をチェックすることが可能になる。もしデータの意味がアプリケーションで必要な目的を含んでいた場合、翻訳は次の通り進む。まずスキーマレットから、構造と名前に関する対応ルールを導出する。続いて、そのルールにドメインタイプの変換式を組み合せる。最終的にターゲットスキーマに対する翻訳式を自動生成する。Schemalet の貢献は、最悪ケースで m 対 n になるスキーマ変換の対応数を減らし、変換における意味的な一貫性を保証する点である。

Keywords

XML, XML スキーマ、セマンティック相互運用性、データ翻訳、オントロジー

1. はじめて

今日では、多くの産業が自分たちのインターネットサービスにおいて、XML を採用し始めている。この主な理由のひとつは、XML スキーマ言語の標準化が進み、XML データに関する構造的な合意が形成し易くなった点にあるだろう。XML スキーマは、XML データが合意に合致しているかどうか機械的にチェックすることを可能にする。これは、結果として異なった組織間の情報システムの接続性を高めることになる。

しかしデータ交換性にはまだ大きな課題が残っている。スキーマ合意は、それ自体で正規化されない点である。つまり、異なった組織が同じ情報を異なったスキーマで仕様化できてしまう。これはスキーマ、そしてそのデータ表現の不統一を招く。現在、多くの中立的な組織がスキーマの標準化を進めているが、組織や各アプリケーションごとの記述要求は大きく異なり、標準化自体なかなか進んでいない。更にグローバルスキーマの柔軟性の欠如と比較すると、インターネット上に複数のスキーマが存在することは自然である。必要なのは、スキーマ相互運用性(schema interoperability)の技術である。

本論文では、スキーマという用語を特定のスキーマ言語のことではなく、データ表現の抽象的な構造という意味で用いる。問題は、そのようなスキーマをどのように相互運用させるかである。まず、我々はスキーマ相互運用に関する技術を、結合(incorporation)、翻訳(translation)、演繹(deduction)の 3 段階に分類する。スキーマ結合は、異なった組織が独立的に定義した既存のスキーマを組合せることで新しいスキーマを作ることである。例えば、アリスは出版社が定義する Book スキーマと金融サービスが提供する Payment スキーマを取り入れることで、新しい BookCatalog スキーマを作ることができる。スキーマ翻訳は、あるスキーマから別のスキーマに意味的な一貫性を保持しながら変換することである。ボブは、アリスの BookCatalog を自分の BookCatalog に翻訳することができるが、出版社の Book スキーマは支払い情報が欠如しているため、翻訳することができない。最後の段階の演繹は、既存のスキーマやデータ表現から新しい構造や知識を獲得することである。例えば、「東京は日本の首都である」というデータ表現に対して、「大阪は日本の首都か？」という問合せが可能になる。

我々の分類にしたがえば、XML スキーマ言語(XSD や RELAX, TREX[14, 18, 19, 20])はスキーマ結合を促進する。基本的には、将来の課題ともいえるスキーマ言語間の統合問題もスキーマ統合に属する。これに対して、セマンティック Web や多くの XML オントロジープロジェクトは、演繹なデータ処理を究極的な目標としている。知的なデータ処理が与える影響は大きいが、そこまでの過程もまだ緒に付いたばかりといえる。これらの試みに対し、本論文ではスキーマ翻訳に議論を集中させる。これは、セマンティック B2B 統合など、[10] 現在のインターネットアプリケーションが直面している現実的な問題がスキーマ翻訳技術によって解決できると考えるからである。

スキーマ翻訳では、明らかな要望として、個々のスキーマ間について翻訳ルールを記述することを避けたい。そのために、我々はスキーマをドメインタイプと相互関係の二つの独立した意味論から再構成するところから始める。Schemalet(スキーマレット)は、ドメインタイプの相互関係を定義する概念スキーマ言語と言える。Schemalet を用いることで、我々はドメインと相互関係に起因する意味的なミスマッチを検出することができる。そして、最終的なスキーマ翻訳のルールは、それぞれの対応ルールから自動的に合成することが可能になる。この機構は、仕様化作業を単

純化だけでなく、翻訳における意味的な非一貫性を排除できる利点がある。

本論文では、Schemalet の仕組みとそれを XML データとスキーマに適用する方法について述べる。残りは、次の通り構成される。第 2 節では、Schemalet システムの導入である。第 3 節は、Schemalet のデータモデルについて定義する。第 4 節では、翻訳モデルについて定義する。第 5 節では XML への適用を述べる。第 6 節では関連研究と比較する。第 7 節では本論文を総括する。

2. スキーマとデータ表現の翻訳

我々は、スキーマとデータ表現に対する翻訳について、動機、基本的なアプローチ、そして目標を述べることから始める。

2.1 背景

XML スキーマ言語は、ある特定のドメインで用いられる XML データの構造に関する合意を仕様化する。スキーマ検証器は、与えられたデータがその合意に準拠しているかどうか、自動的にチェックできる。しかし、その検証はシンタックス的な準拠に限定される。つまり、同じアプリケーションドメインを対象に記述されたデータであっても、異なるスキーマで仕様化されている場合、不正なデータと見なされる。異種スキーマが混在している環境では、スキーマ合意だけでは、データの交換性を保証することはできない。

図 1 は、XML データ表現間の可能性のある相違について表示したものである。データ (a)と(b) は、スキーマ検証器は認めないが、我々人間は同じ情報とみなすことができる。もしこれらのデータ表現を同様に扱いたい場合は、何かしらの手作業を加える必要がある。しかし、これらが本当に同じ情報なのか実は怪しい。例えば、データ(a)は図書館の蔵書録かも知れない。そのようなショッピングエージェントが誤解して発注しないように防がなければならない。更に、もし知らない言語(例えばクリンゴン語[1])でタグ付けされていた場合、我々人間であってもデータの意図を認識することはできない。

```
<book>                                  <sales product="book">
 <author> McLuhan </author>              <name> McLuhan:
 <title> Understanding Media            Understanding Media </name>
</title>                                 <code>0262631598</code>
 <isbn>0262631598</isbn>                  <price currency="yen"> 5200
 <price> $17.95 </price>                 </price>
<book>                                   <sales>
```

(a) (b)

図 1 XML における表現の違い

与えられたデータは一体何なのか？データ表現とスキーマの概念的な意味が重要になる。異なった形式で記述された同じ概念のデータは同様に処理されなければならないし、同じ構造でも異なった概念をベースにしている場合は違った処理がされるべきである。未来の XML 情報システムは、概念を理解する方法で、データ表現やスキーマを扱えるようになるべきである。

2.2 ドメインタイプと相互関係

最初の問題は、XML データ表現の概念的な意味(以後、単に概念と呼ぶ)をどのように捉えるかである。我々は、概念を 2 つの分離した意味論(ドメインタイプと相互関係)から再構成する。ドメインタイプは、データ表現の構造自体が持つ意味であり、ここでは(直感的な理解のため)実世界で使われる単位(例 円やメートル)、スケール(kg や MB)、パターン(日付の yyyy-mm-dd 形式や電話番号)などが相当するとする。この種のタイプは、比較的組織の壁を越えて意味の合意が成り立つと想定できる。ただし、ドメインタイプによって全ての意味を捉えることはできない。例えば、980 円の価格と 49 円の税金を比較してみれば明らかである。データ表現の意味は、内在的な構造に由来するだけでなく、文脈中における相互関係(interrelation)にも依存する。

データの意味を交換するため、XML はこれらの 2 種類の意味論をマークアップする必要がある。次の可能性のある XML データとそのスキーマ定義をまず見てみよう。

```
<product> <price> 980 </price> </product>
                                                    (in XML)
<simpleType name="price" type="integer"/>
                                                    (in XSD)
```

上記の XML データは、「商品の価格は 980 円」と解釈できる。しかし 980 が US ドルでなく、日本円であるという保証はどこにもない。ここでは、”integer”というエンコーディングしか明らかになっていないが、データタイプで単位を明らかにする必要がある。更に、product や price などの要素名は何かの意味を伝えるように見えるが、これも自然言語と同じように同義語 (synonyms) や同音異義語 (homonyms)の曖昧さがある。 例えば、価格は消費税が含まれているかどうか？要素名では、データ値の相互関係を厳密に宣言するのは難しい。そこで、我々は述語論理の記法を用い、2 つの要素の関係を定義するものとする。

sell(x, y): x sells y

sell_for(x, y): x is sold for y without tax and service charges.

この述語と定数(もしくは XML 要素名)を用いれば、XML データ表現の相互関係を記述することができる。

$$\text{sell}(\text{merchant}, \text{product}) \wedge \text{sell\_for}(\text{merchant}, \text{price})$$

2.3 Schemalet のアーキテクチャ

我々は、データ表現を 2 つのセマンティック部品に分解するアプローチを採用する。Schemalet は、これらの部品を再構成することでデータ表現やスキーマを再構造化するモデル言語である。Schemalet のアーキテクチャは、図 2 に示す。ここで我々はドメインタイプと相互関係(述語)は、インターネット上でオントロジーとして共有管理されるものと想定する。Schemalet は、メタデータとして、述語論理の変数に XML 要素名を割り当て、それぞれにドメインタイプを連想することで、新しい概念スキーマを定義することができる。Schemalet 定義の間では、ドメインタイプや相互関係

[1] スタートレックに登場するエイリアンの言語

の対応ルールから自動的に翻訳ルールを導出することが可能になる。

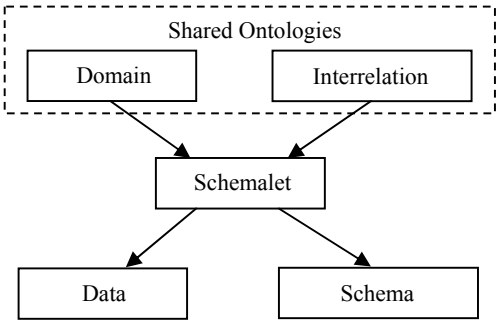

図 2 スキーマレットのアーキテクチャ

我々は、データ表現やスキーマに付属させる小さなスキーマ部品という意図で Schemalet を名付けた。そのため、Schemalet の仕様は、XML スキーマ言語とはいくつかの点で異なる。特に、Schemalet はデータ表現のフラグメントに対して意味を補足するが、スキーマ言語はデータ全体上において数々の制約(順序、範囲、出現など)を厳密に与える。Schemalet は、既存のスキーマ言語の置き換えを意図しているわけではない。

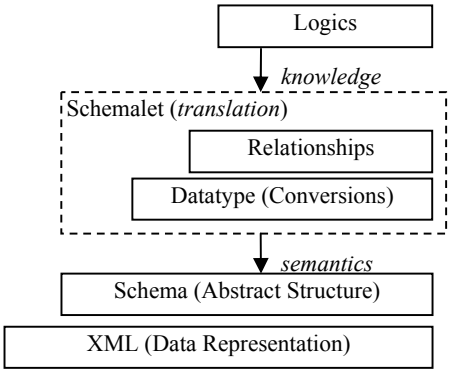

図 3 Schemalet の言語階層

図 3 は、Schemalet 周辺の仕様言語の階層を表している。この階層図は、セマンティック Web プロジェクトにおける"RDF+Ontology"に似ている。これは、Schemalet の目標が XML 上の構造ベースのデータ処理と RDF 上の知識・論理ベースのデータ処理の橋渡しを行うこと目指しているからである。ただし、本論文で議論するモデリング言語の名前が、schema-log(ic) でないことに注意して欲しい。我々は、Schemalet は論理学的な推論のシステムを導入せず、直感的なデータ変換関数の組み合わせで、スキーマ翻訳を目指している。

3. Schemalet データモデル

まず、データ表現とスキーマ翻訳の議論の土俵としてデータモデルをおく。ここでは我々は Schemalet モデルと呼ぶ。特徴は、ドメインタイプと相互関係の表現を分離する正規化が導入できる点である。XML データは、必ずしも Schemalet モデルで正規化できないが、非正規化データへの拡張は第 5 節で述べる。表 1 は、本論文を通して用いる表記法を示している。

{…}, […]	セットとタプル(シーケンス)
l_i, **L**	ラベルとその全ての有限集合, $l_i \in$ **L**
t_i, **T**	タイプ名とその全ての有限集合, $t_i \in$ **T**
v_i, **V**	データ値とその全ての有限集合, $v_i \in$ **T**
p_i, **P**	述語記号とその全ての有限集合, $p_i \in$ **P**
$t_i::v_j$	t_i で型付けされた値 (v_j)
x_i, **X**	変数とその全ての有限集合, $x_i \in$ **X**

表 1 基本的な表記

3.1 基本的な構造

データは、いくつかの値からなる構成である。Schemalet モデルは、そのような構成を自己記述的なオブジェクト形式で表す。

定義 3.1. *Schemalet オブジェクト*は、セマンティック要素 *semantic elements* の集合である。セマンティック要素は、タプル [*l*, *t*, *v*]からなる。ここで、

- *l* は *v* のラベルであり、
- *t* は *v* のタイプを識別する名前であり、
- *v* はセマンティック要素のデータ値である。

オブジェクト上の要素の順序には意味がないものとする。更に、オブジェクトは値の一種として、複合オブジェクトの表現を可能にする。次は、オブジェクトの例の定式表記である。

例 3.2. ある schemalet オブジェクト (非正規形)

{[Product, String, "aCD"],
[JPYPrice, Integer, 1200],
[USDPrice, Double, 9.80],
[Date, Object, {[Year, Integer, 2001], [Month, Integer, 12], [Day, Integer, 24]}]}

3.2 ドメインタイプと意味論

ドメインタイプは、単純な値(simple values)と複合な値(complex values)の種類を表す。次の通り定式化できる。

定義 3.3. ドメインタイプ *t* は、関数 td(*t*, *v*) → *v* が成り立つデータ値の集合 (all *v* ∈)Vt である。

第 2 節で簡単に導入したとおり、単純な値に対してドメインタイプを拡張し、より意味を捉えられるようにする。表 2 はドメインタイプの例である。また、エンコーディングはドメインタイプから独立し、*Encoding*/*DomainName* と別記することもある。

ドメイン名	エンコーディング	値の例	説明
Yen	Integer	1200	日本円
Usd	Double	9.80	US ドル

phone-ja	String	03-5841-2480	電話番号(国内)
Date	String	2001-12-24	日付
Date	Object	{[year, …], …}	オブジェクト

表 2: ドメインタイプの例

我々は、ドメインタイプの意味をよりはっきりさせるため、タイプ変換(datatype conversion)を拡張する。

定義 3.4 t_1 から t_2への変換関数(*conversion function*)は、tc(t_1, t_2, v_1) → v_2と表記する。ここで、td(t_1, v_1) → v_1 かつ td(t_2, v_2) → v_2である。

Schemalet モデルでは、タイプ変換を導入することで柔軟なタイプの扱いができる。例えば、従来のオブジェクト指向のタイプチェック機構と比較してみる。オブジェクト指向の値では、クラス-サブクラス関係からタイプチェックされる。例えば、yen を currency のサブタイプとすると、yen 型の値は currency 型に準拠するが、dollars とは一致しない。ところが、Schemalet の値は、型推論風にチェックされるため、yen と dollars が変換可能なら yen は dollars 型に準拠する。

3.3 概念ラベル

続いて、ラベルの意味、つまりデータ値の相互関係に話題を移す。第 2 節で述べたとおり、ラベルの意味は同義語や同音異義語があるための曖昧である。そのため、ラベルの意味をより明確に捉えるため、述語論理(predicate calculus)を導入する。

定義 3.5. 文脈的な意味、つまり値の相互関係は述語論理式で p(x_1, $x_{2, \ldots,}$$x_n$)で記述する。ここで、

- p は述語記号(p ∈ **P**)であり、
- x_1, $x_{2, \ldots,}$$x_n$ は、ラベルもしくはデータ値が入る。

我々は、ラベルの相互関係から、ラベルの同義語を定義することができる。

定義 3.6. ラベル l_1 と l_2 は、p(x, l_1) と p(x, l_2) が成り立つとき、シノニムである。

例えば、sell_for(x_1, x_2) を"x_1 sells for x_2"の相互関係を表す述語とする。例 3.2 のラベル JPNPrice と USDPrice が概念的には同じであることは、次のとおりになる。

sell_for(x_1, JPNPrice) ≅ sell_for(x_1, USDPrice)

いよいよ、データ表現に対する Schemalet 正規化形式を定義することができる。

定義 3.7 正規形データ(*normal object*)は、シノニムなラベルを含まない。言い換えれば、異なったタイプであっても同じ概念的な関係を持っている値なら、同じラベルが割り当てられる。

例 3.8 次は、例 3.2 の正規形としての書き直し版である。(複合オブジェクトは省略されている。)

{[Product, String, "aCD"],
[Price, Integer/yen, 1200],
[Price, Double/ usd, 9.80] }

正規形では、もはやラベルはオブジェクト上要素を識別する名前にはならない。逆に同じ概念の要素には同じラベルを与えることができる。我々はラベルを概念ラベル(*concept label*)と呼ぶ。

3.4 スキーマと Schemalet

我々は、Schemalet におけるデータ表現を定義してきた。まとめとして、その仕様となるスキーマと Schemalet を定義する。

スキーマとはいわゆる構造スキーマであり、一般的に抽象的な構造に様々な制約が与えられたものである。本論文では、順序や出現などの制約を省略して、次のように簡単に定義を与える。

定義 3.9 スキーマ(*schema*)は、schemalet オブジェクトの抽象的な構造であり、{[l_1, t_1], [l_2, t_2], .., [l_n, t_n]}と表記される。

これに対して、Schemalet は概念スキーマの一種であり、概念ラベルの意味を相互関係として記述する。

定義 3.10 *schemalet* は、ラベル付けされた相互関係の集合である。しばしば、Σ と表記し、Σ={$p_1(L_1)$, $p_2(L_2)$, …, $p_n(L_n)$}である。

例 3.11 次は、例 3.8 の Schemalet 仕様である。

{sell(merchant, product) , sell_for(merchant, price)}

ここで、ラベル merchant は、オブジェクト上に存在しない定数ラベルである。既存の述語をラベルで接続することにより、is_sold_for(product, price)を定義している。

本論文では、全てのオブジェクトとスキーマはその概念を表す schemalet が連想されていると想定する。この参照は、$\leftarrow_{ref}\Sigma$で表記する。

最後に、Schemalet のモデル論な解釈を簡単に導入する。ある Schemalet Σ 上の変数ラベル l を全て型付き値 t:v で置き換えられるとき、Σ は決定的に解釈可能である。変数ラベルが残るとき、非決定的な解釈である。つまり、例 3.11 の Schemalet は、例 3.8 のオブジェクトによって、決定的に解釈することができる。

sell(merchant, String::"aCD") ∧ sell_for(merchant, {yen:1200, usd::9.80})

また、スキーマは解釈が非決定的な Schemalet になる。我々は、我々は、決定的に解釈された Schemalet をファクト(fact)、非決定的な Schemalet をターゲット (target)と呼ぶこともある。

4. 翻訳システム

データ翻訳は、単なるデータ変換ではない。変換前と変換後のデータに対して、意味的な一貫性を保証する必要がある。正規化データ上で Schemalet を用いれば、2 種類の意味的なギャップが検出できる。この節では、それらに対し、個々の対応ルールを導入することで、最終的にデータ全体の翻訳ルールを導出する方法を述べる。

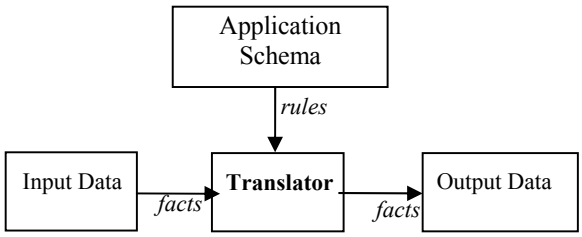

図 4. 翻訳器のシステム図

図 4 は、翻訳のアーキテクチャモデルを示している。最初に、アプリケーションやソフトウェアエージェントは、翻訳後のデータ表現を仕様化したターゲットスキーマ(アプリケーションスキーマ)を翻訳器にセットする。翻訳器は、与えられたデータとアプリケーションスキーマ間の対応ルールを逐次適用しながら翻訳を行う。本論文では、読みやすさのため、ボトムアップ的なプロセスで示すが、翻訳器をこのとおりに実装する必然性はない。

4.1 異種ディメンジョンの対応

ディメンジョンの相違は、同じ情報が違った方法で表現されたとき発生する。(例えば、3 メートルと 3 フィートの違いなどである。) Schemalet データでは、この種の違いはドメインタイプで識別可能であり、その対応ルールはタイプ変換として与えられる。

定義 4.1 (異種ディメンジョンの対応) Schemalet データ上では、ct(t_1, t_2, v_1) → v_2が成り立つとき、セマンティック要素 [l, t_1, v_1] は[l, t_2, v_2] に相当する。

タイプ変換では推移律(*transitive*)、つまり ct(t_2, t_3, ct(t_1, t_2, v_1)) = ct(t_1, t_3, v_1)が成り立つ。我々は新しい対応ルールを導出することも可能である。

例 4.2 翻訳器にアプリケーションスキーマ{[product, String], [price, yen]} とタイプ変換 ct(usd, yen, 9.80) → 1200 をセットする。このとき、入力データ{[product, String, “aCD”], [price, usd, 9.80]} は、次のとおり変換される。

{[product, String, “aCD”], [price, usd, 9.80]}

ct(usd, yen, 9.80) → 1200

{[product, String, “aCD”], [price, usd, 9.80], [price, yen, 1200]}

{[product, String], [price, yen]}

{[product, String, “aCD”], [price, yen, 1200]}

データ変換は、ときどきデータ衝突の問題を引き起こす。つまり、いくつかの変換ルールを適用できる場合、異なった複数の値、つまり同一オブジェクト上で 2 つの要素[l, t, v_1] 、[l, t, v_2] が存在し、$v_1 \neq v_2$ になる可能性がある。データ衝突の回避は、興味深い課題であるが、ここではタイプ変換にコストを導入し、コスト見積りで最適な解を選択するものとする。

4.2 同義なラベルの対応

同義なラベルは、同じ概念の情報に対して、入力データとアプリケーションスキーマが異なった概念ラベルを利用しているときに発生する。これは、お互いの Schemalet 仕様の相互関係を比較することで、概念ラベルの違いが検出でき、同義なラベルの付け直しとして導出できる。

定義 4.3 Σ と Σ’ を共通した相互関係 p を含む 2 つの異なった Schemalet とする。ラベル l と l' はそれぞれ Σ と Σ’上のラベルとすると、l は p(x, l)と p(x, l') が成り立つとき、l'に付け直すことができる。

Schemalet 間のラベルの付け直しは、次のようにデータ翻訳に適用することができる。

例 4.4 翻訳器にアプリケーションスキーマ {[product, String], [price, yen]} ← ref {sell(c_1, product), sell_for(c_1, price)} をセットする。ここで、入力データ{[goods, String, “aCD”], [cost, yen, 1200]} ← ref {sell(c_2, goods), sell_for(c_2, cost)} は、次のとおり翻訳できる。

(in){[goods, String, “aCD”], [cost, yen, 1200]}

← ref sell(c_2, good) ∧ sell_for(c_2, cost)

---

sell(c_2, String::”aCD”) ∧ sell_for(c_2, yen::1200)

{[product, String, x_1], [price, yen, x_2]} ← ref sell(c_1, x_1) ∧ sell_for(c_1, x_2)

{[product, String, “aCD”], [price, yen, 1200]}(out)

我々は、どのような場合、Schemalet 間の翻訳が成立するか調べておく。ここで、Σ_D と Σ_S をそれぞれ入力データとアプリケーションスキーマに対する Schemalet とする。例 4.4 は、$\Sigma_D = \Sigma_S$ の場合である。現実には、入力データとターゲットスキーマの概念が完全に一致するケースを稀である。次に、$\Sigma_D \neq \Sigma_S$ の状況で、どのような場合に翻訳可能かみていく。まず、明らかに 2 つの Schemalet 仕様が完全に不一致な場合(つまり $\Sigma_D \cap \Sigma_S = 0$)、翻訳不可能である。つまり、$\Sigma_D$ から Σ_S への翻訳のためには両者の間に類似性(部分的な包含関係)が必要となる。Schemalet では、$\Sigma_S \subset \Sigma_D$ のとき、入力データがターゲットスキーマより表現豊かであると判定できる。逆に、ターゲットスキーマの方が表現豊かな場合は、$\Sigma_D \subset \Sigma_S$ となる。そして、Σ_D が Σ_S へ完全に翻訳可能な場合は、$\Sigma_S \subseteq \Sigma_D$ が成り立つ場合のみである。これは、包含関係のある Schemalet 仕様、例えば Σ = $p_1(l_1, l_2)$ と Σ’ = $p_1(l'_1, l'_2)$ ∧ $p_2(l'_1, l'_3)$を想定することで容易に確認できる。

ただし、Σ_D と Σ_Sの間で完全な包含関係が成り立つことも稀であろう。最後により一般性の高い原理として、Schemalet 仕様間の全体の翻訳可能性から、ラベルの翻訳可能性としてまとめておく。まず、ラベル l 上の部分 Schemalet 仕様 Σ^l ($\subseteq \Sigma$)を導入し、Σ^lは Σ 上の l を引数にした全ての相互関係を持っているものとする。ここで、$\Sigma_D{}^l$

$\subseteq \Sigma_D$ と $\Sigma_S^{l'} \subseteq \Sigma_S$ を与えれば、定義 4.3 はより正確に書き直すことができる。

定義 4.3' Σ_D の概念ラベル l は、$\Sigma_S^{l'} \subseteq \Sigma_D^{l}$ が成り立つときに限り、l'に翻訳可能である。

4.3 データ表現とスキーマの翻訳

Schemalet では、データタイプと相互関係から派生するミスマッチをそれぞれ独立して対応付けることができる。これらの対応を結合させることで、データ表現自体の翻訳を導出することができる。具体的には、翻訳器は：

1. 入力データとターゲットスキーマの Schemalet 仕様 Σ_D と Σ_Sを比較する。Σ_D上の各ラベルに関して、$\Sigma_S^{l'} \subseteq \Sigma_D^{l}$が成り立つとき、$l'$に書き換える。
2. データタイプを変換する。翻訳されたラベルとターゲットスキーマ上のラベルが一致した場合、異ディメンジョンを検査する。ここで Schemalet データモデルでは、複合値は独自の Schemalet 仕様を持つこと注意する。複合値間データ変換の必要が発生した場合は、更なる Schemalet 翻訳が再開し、ステップ 1 に戻る。

本翻訳システムは、インターネット上で相互関係(述語)をオントロジーとして共有することで、相互関係間の対応ルールは必要がないという前提において成り立つ。ただし現在、あらかじめよく設計された相互関係であっても、理想どおり正規化するのが難しいケースが発見されている。その場合、Schemalet では、述語間の対応を関数として表現するが、本論文では紙面の都合上、詳しく述べない。

5. Schemalet 言語の設計と実装

Schemalet 言語は、データ表現やスキーマの意味としてドメインタイプと相互関係を付加するモデリング言語である。本節では、ドメインタイプの仕様書、XML 上での相互関係の定義、そして XML データ表現に対する Schemalet 言語の例を説明する。現在、Schemalet 翻訳エンジンは、Java 言語で開発中であり、近日公開する予定である。

5.1 タイプとタイプ変換の仕様

まず、定義 3.3 と 3.4 で定義したドメインタイプの仕様書から説明する。通常、XML データタイプのように、制約ベースの型定義は、宣言的な方法で仕様化することができる。一方、タイプ変換は必ずしも静的な対応ではなく、計算的な関数やリモートデータベースへの検索が含まれるため、宣言的な方法は直感的ではない。そこで Schemalet 翻訳エンジンには、プログラミング形式で仕様を提供することにする。幸い、Java プログラミング言語はこの種の仕様をインターネットで共有するために優れた特性を備えている。

注意したいのは、全ての仕様書をプログラミングコードで記述するわけでない点である。現在、ボキャブラリやオブジェクトに対する多くの XML 言語が存在する。これらのパーサを組込んで、Schemalet 翻訳エンジン上で再利用するのは特別なことではないだろう。

5.2 相互関係の仕様書

相互関係の仕様書は、XML 形式を採用し、述語記号とその変数を連想して定義する。例えば、述語 sells(x, y) は、次のように表現される。

```
<interrelation predicate="sells" arity="2" variables="x,y">
  <sentence xml:lang="en"> <var name="x"/> sells <var
name="y"/></sentence>
  <sentence xml:lang="ja"> <var name="x"/> は<var
    name="y"/> を売っている</sentence>
</interrelation>
```

Schemalet 言語は、<interrelation> 要素を用いて、predicate 属性で指定された述語記号と variables 属性で指定された変数名を定義する。ここで、述語記号はインターネット上においてユニークであると想定する。(実際は、名前空間と連想させる。) 更に、仕様書は変数間の相互関係の意味を表明するセンテンスを含んでいる。これは、意味の形式定義ではないが、人間がレポジトリーから相互関係を探すときのヒントとなる。

5.3 基本 Schemalet 言語

まず、第 3 節で定義した正規化データに対する Schemalet 言語から説明する。ここでは、正規化データを直接エンコーディング可能な X#と名付けた特別なデータ表現言語を用いる。

X#は、XML 要素を自然に Schamelet オブジェクトとして解釈できるように、最小限のコンストラクトを追加しているだけである。例えば、 [*label*, *type*, *value*] の 3 組は、XML 上で次のように表現できる。

<*label* xs:type="*type*"> *value* </*label*>

特長は、ドメインタイプを xs:type 属性で自己記述している点である。次は、例 3.8 の X# バージョンである。

```
<an Object xs:type="Object" xs:schemalet="Product">
  <product> aCD </product>
  <price xs:type="String/yen"> 1200 yen </price>
  <price xs:type="Number/usd"> 9.85 </price>
</anObject>
```

xs:schemalet 属性は、Schamelet 定義への参照である。Schemalet 上では、X#上のラベルと相互関係の述語変数が相互に連想される。X#上に存在しないラベルは、定数ラベルとして特別に宣言される。次は、例 3.11 の Schemalet 定義であり、名前 "Product"により XML 表現から参照されている。

```
<schemalet name="Product" from="normal">
  <constant name="merchant">
  <predicate id="sell">
    <label var="x" name="merchant" /> <label var="y"
name="product"/>
  </predicate>
   <predicate id="sell_for">
    <label var="x" name="merchant" /> <label var="y"
name="price"/>
  </predicate>
</schemalet>
```

5.4 拡張 Schemalet 言語

全てのデータが X#のような正規化された Schamelet データであれば、Schamelet 言語にとって理想的である。しかし、

残念ながら必ずしも正規化されていないことが多い。現実的な立場に立てば、一般の XML データに対して、Schamelet 仕様を適用する必要がある。例えば、次のようなデータを考えてみる。

```
<product name="CD">
  <price currency="usd"> 9.85 </price>
  <jpnprice> 1200 </jpnprice>
</product>
```

非正規データに対する拡張 Schamelet 言語では、まずドメインタイプの意味を分離する必要がある。続いて、概念関係を比べて、同じ概念の値には同じラベルをつける必要がある。XML 上では、この対応付けは、要素のフラグメントに対して、XPath [22]を用いて指定することができる。拡張 Schemalet 言語では、<map> 要素用いて、XPath とラベルとドメインタイプの組合せを対応付ける。

```
<schemalet name="Product2" from="xml">
  <map label="product" type="String" path="@name" />
   <map label="price" type="Number/usd"
path="price@curency='usd'" />
   <map label="price" type="Number/yen" path="jpnprice" />
   <constant name="merchant">
  <predicate id="sell">
    <label var="x" name="merchant" /> <label var="y"
name="product"/>
  </predicate>
   <predicate id="sell_for">
    <label var="x" name="merchant" /> <label var="y"
name="price"/>
  </predicate>
</schemalet>
```

Schemalet 翻訳エンジンは、<map>を用いて、XML データを正規化データとして読み込むことができる。逆に、翻訳されたデータ表現も XML データとして出力可能である。注意：XML データ表現に対して、XPath による対応付けは直感的であるが、複雑な構造にはあまり適さない。将来の課題といて、Schamelet オブジェクトと構造単位が近い、XSD の ComplexType などに正規化マップする方法が必要である。

6. 関連研究

異種データ間の意味的相互運用性は、マルチデータベースの分野で盛んに研究されてきた。 [11, 17]. これらの多くの優れた結果は、XML データの統合作業にも有用である。しかし XML データは伝統的なデータベースと異なる点も多い。 [2]. これは深遠な意味相互運用性の課題を単純化できる可能性を与えてくれる。特に、まずドメインタイプや相互関係をオントロジーとして共有した上で、既存の XML データ表現を再構成しなおすという立場を採っている。これは、データベースと XML データのライフサイクルの違いから妥当であると思う。

我々のアプローチのアイディアは、“the theory of semantic values” [16]から来ている。彼らは、Lisp 風のメタデータとして、データベース値に対して単位やスケール、状態などを付加し、リレーショナルデータベースの意味的統合を進めた。我々の意見では、状態は別の意味論として扱うべきであると思う。特に、XML データ上のように各要素が相互にメタデータとなりえる場合、Sciore の繊細な方法は直接適用するのは難しいだろう。

述語論理を相互関係を表現するために利用することは、知識表現の分野では非常に一般的である。[6, 7, 15] しかし、我々は演繹ルールの健全性に着目するより、翻訳における直感的な解釈を強調している。この解釈や翻訳モデルは、ER モデル、Bunge が開発したオントロジー公準系、更に自然言語の構成原理などの経験的な結果の影響を受けている。[4,9,23]. 結果として、翻訳はドメインタイプの変換とラベルの付け直しで構築することが可能になった。

異種データソースに対する変換や変形、翻訳に対する試みも少なくない。[6,5,11]. 多くは、直接データ間の対応を記述するルール言語をベースにしている。これらに対し、相互関係の組み合わせから構造の同義的な対応を抽出することができる。ラベル同士の対応を直接、ルール化する必要はないのである。我々の知る限り、この方法は独創的であり、仕様の作業を単純化し、更にデータ変換において意味的な一貫性を保証することができる。

最後に、現在の XML 変換技術と比較しておこう。schemalet と XSLT [22]の違いは、意味的な一貫性にある。XSLT を用いれば、例えば<price>9.85</price>は HTML の 9.85 dollars に変換できる。しかし、これは<price>と の意味が異なるため、翻訳とはいえない。

7. 結論

今日、異なった組織の間において XML データとそのスキーマのセマンティック相互運用性に対する必要性が高まっている。本論文では、セマンティック相互運用性に対する翻訳アプローチを報告する。Schemalet は、ドメインタイプと相互関係によって、XML データとスキーマを再構成するモデリング言語である。Schemalet を用いれば、情報システムは自動的にデータとアプリケーションスキーマの意味的な対応をチェックすることが可能になる。もしデータの意味がアプリケーションで必要な目的を含んでいた場合、翻訳は次の通り進む。まず Schemalet から、構造と名前に関する対応ルールを導出する。続いて、そのルールにドメインタイプの変換式を組み合せる。最終的にターゲットスキーマに対する翻訳式を自動生成する。Schemalet の貢献は、最悪の場合 m 対 n 対応になるスキーマ間の変換式の数を減らし、変換における意味的な非一貫性を防ぐ点にある。

本論文では、Schemalet 翻訳モデルの基本アイディアを示し、その言語のプロトタイプを論じた。しかし、将来に対してより多くの課題も残されている。ここでは学術的な課題のみ列挙する。ひとつは、セットや順序、その他のコンストラクトの意味を捉えるためモデルを洗練する必要がある。もうひとつは、Schamalet で仕様化する概念スキーマの再利用性を高めるため、述語の語彙分解と相互関係のクラス化と意味継承の仕組みを解明する必要がある。これらの課題とあわせて、現在の XML 技術との互換性を高めながら、実用化を進めていきたい。我々の願いは、インターネット上の産業に対して、意味相互運用性のための頑強な理論とシンプルで実用的なツールを提供することである。

謝辞

本研究は、文部科学省特定領域研究(情報学：課題番号 13224020)の研究費補助金の支援を受けて行っています。ま

た、自由な研究環境を提供して頂いている坂村健教授(東京大学)に深謝します。

参考文献

[1] S. Abiteboul, S. Cluet, T. Milo. Correspondence and Translation for Heterogeneous Data. *Theoretical Computer Science*, 2001.

[2] S. Abiteboul, P. Buneman, and D. Suciu, *Data on the Web: From relations to Semistructured Data and XML*. Morgan Kaufman, 1999.

[3] T. Berners-Lee, M. Fischetti and M. Dertouzos. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper San Francisco, 1999.

[4] P. Chen. The Entity Relationship Model – Toward a Unified View of Data. *ACM Trans. on Database Systems*, 1(1), 1976.

[5] S. Cluet, C. Delobel, J. Simeon, and K. Smaga. Your Mediators Need Data Conversion!. Proc. of ACM SIGMOD'98, pp. 177-188, 1998.

[6] S. B. Davidson and A. S. Kosky. WOL: A Language for Database Transformation and Constraints. In *Proc. of ICDE'97*. pp. 55-65, 1997.

[7] M Kifer, G. Lausen, and J. Wu. Logical Foundations of Object-Oriented and Frame-Based Language. Journal of ACM, 42(4), pp, 741-843, 1995.

[8] K. Kuramitsu and K. Sakamura. A Ubiquitous Data Model: Semistructured Object. IPSJ Trans. on Database, TOD12, 2001. (In Japanese)

[9] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. Prentice-Hall, 2000.

[10] B. Omelayenko. Integration of Product Ontologies for B2B Marketplaces: A Preview. SIGEcom Exchange, 2(1), pp 19-25, 2001.

[11] E. Pitoura, O. Bukhres, and A. Elmagrmid, Object Orientation in Multidatabase Systems. *ACM Computing Surveys*, 27(2), June 1995.

[12] O. Lassila and Ralph R. Swick. Resource Description Framework (RDF) Model and Syntax Specification, *W3C Recommendation*, 1999.

[13] J. Reagle. Eskimo Snow and Scottish Rain: Legal Considerations of Schema Design. *W3C Note, December-1999*. Available at http://www.w3.org/TR/md-policy-design

[14] M. Murata (ed). *Information Technology – Document Description and Processing Languages – Regular Language Description for XML (RELAX) – Part 1: RELAX Core*, ISO/IEC, DTR 22250-1, 2001.

[15] C. Rich (ed.) Special Issue on Implemented Knowledge Representation and Reasoning Systems. *SIGART Bulletin*, 2(3), 1991.

[16] E. Sciore, M. Siegel, and A. Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems, *ACM Trans. on Database Systems*, 19(2), pp.254-290, 1994.

[17] A. P. Sheth and James A. Larson. Federated Database Systems for Managing Distributed Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, 22(3), 1990.

[18] TREX – tree regular expression for XML, 2001.

[19] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (ed). XML Schema Part 1: Structures. *W3C Recommendation* 2001.

[20] P. V. Biron and A. Malhotra (ed). XML Schema Part 2: Datatypes. *W3C Recommendation*, 2001.

[21] J. Clark (ed.) XSL Transformations (XSLT) Version 1.0. *W3C Recommendation,* 1999.

[22] J. Clark and S. DeRose. XML Path Language (XPath) Version 1.0. *W3C- Recommendation,* 1999.

[23] Y. Wand, V. C. Storey, and R. Weber. An Ontological Analysis of the Relationship Construct in Conceptual Modeling, *ACM Trans. on Database Systems*, 24(4), pp. 494-528, 1999.