Vol.2014-DPS-158 No.1
Vol.2014-CSEC-64 No.1
2014/3/6

制約つきXML向けドメイン特化言語の提案

鈴木　源吾[1,a)]　榎本　俊文[1,b)]　小林　伸幸[1,c)]

概要：近年，XML を利用したシステムは一般的になったが，その開発には XML 特有の言語である XQuery 等を用いる必要がある．しかし，XML のデータ構造が非常に柔軟であるため，XQuery 等の XML 技術は一般化されすぎており使い勝手が悪く，開発生産性を低下させる一因となっている．そこで，完全な XML 処理には向いていないが，ある程度限定的な目的を実現する，使い勝手の良い ドメイン特化言語（DSL）を設計することを提案する．本研究ではそのドメインを HTML 等の汎用文書ではなく，構造がより単純で「データ的」であることを想定し（例：ソフトウェア設計情報），XML のキーを利用した一致判定や，複数の XML を一つにまとめる「名寄せ」演算を基礎とする演算体系と，その DSL 向けのプログラミング言語である Groovy による試験的な実装について述べる．

キーワード：XML，ドメイン特化言語，DSL，Algebra

1. はじめに

XML（eXtensible Markup Language）は，文書やデータの意味や構造を汎用的に記述できるのマークアップ言語であり，「タグ」を用いて情報の意味・構造・修飾等を埋め込んでいけるため，電子書籍・特許・ソフトウェア設計書等の電子的なドキュメントや，情報流通や設定ファイルのデータフォーマットとして広く活用されており，技術として定着した．しかし，近年は，以下の様な XML の欠点も広く認識されるようになってきた．

汎用的すぎて扱いづらい　XML のデータモデルは非常に汎用性が高い．これは，HTML 等の汎用のドキュメントを表現する（例：文字装飾）必要があるためであり，そのため，XML をプログラム上で扱う DOM（Document Object Model）は複雑であり，プログラマの頭を悩ませる種の一つとなっている．

プログラミング言語との親和性の低さ　XML はプログラミング言語からは DOM を通じてアクセスする必要がある．DOM はプログラミング言語と独立であるために，どの言語からも使えるメリットを持っているが，逆にプログラミング言語内の配列やオブジェクトとの親和性を欠いている．

開発の難しさ　上記の DOM の問題とも関連するが，XML を扱うためには，覚えるべき技術が多い．例えば，問い合わせ言語としては XQuery・XPath，スキーマ記述言語である XML Schema，XML の変換言語である XSLT，API としては，DOM，SAX がある．これらは，その出生の経緯から必ずしも一貫性のあるわかりやすい仕様にはなっておらず，開発のハードルは高く，技術者の調達にも影響する．特に問合せ言語である XQuery は，汎用的な XML 操作を可能とするために，オーバースペックとなることが多い．

上記のような XML に対する課題認識はあるが，それでも XML を積極的に利用したい状況は多い．

スキーマの存在　XML スキーマを利用することにより，XML の記述の正しさをチェックすることが可能になり，より安全なシステムの構築が可能となる．

標準との整合　XML を利用するケースとして，そもそも XML によってデータ形式の標準が規定されている場合がある．その場合，それを JSON 等にマッピングするよりは，そのまま XML によって扱いたいという要求がある．

そこで本研究では，上記のような XML の利点と，プログラミング言語に親和性の高いオブジェクトを融合させるような．ドメイン特化言語（Domain Specific Language, DSL）を構築することを提案する．対象ドメインを特定することによって，XML の加工パターンが限定される．それらの加工パターンをプログラミング言語内の DSL を扱う枠組みによって，プログラム言語との親和性が高い状態で XML を扱えるようにする．

[1] NTT ソフトウェアイノベーションセンタ
NTT, Minato-ku, Tokyo 108-0075, Japan
[a)] suzuki.gengo@lab.ntt.co.jp
[b)] enomoto.toshifumi@lab.ntt.co.jp
[c)] kobayasi.nobuyuki@lab.ntt.co.jp

2. 本研究における適用ドメイン：ソフトウェア設計情報 XML

XML 向け DSL の適用ドメインとして，ソフトウェアの設計情報の管理を取り上げる．ソフトウェアの設計書は，従来，Office 文書で管理されることが多かったが，Office 文書では，設計書間の整合性をチェックする等の，文書の構造や意味を利用することが困難だった．その問題の解決のために，UML 等のモデルを基本とした設計支援ツールも多く開発されているが [3]，大規模な開発における分割作業に向いていない等の理由で採用は進んでいない．最近は，Excel 等によって設計情報の構造をネストのある表構造によって記述することが多くなっている．そこで，本研究の前提として，ソフトウェアの設計情報は，Excel 等によって構造化されており，それを機械的に XML に変換できるという仮定を置くことにする．

このような設計情報に関する XML のデータ構造的な特徴は以下の通りである．

- 一般の文書（X-HTML）ほど複雑ではない．例えば，文字装飾の重要性は低い．
- 処理・画面・データ構造の単位で構成されることが多く，個々の XML のサイズはあまり大きくはなく，（特に大規模プロジェクトの場合）数が非常に多くなる．
- 構造は明確にあり，階層構造もある．例えば，データの構造や，機能の階層がある．
- 複数の設計情報で共通に参照される情報がある．例えば，データベースの情報は処理設計書とデータ設計書で記述・参照される．
- スキーマは，基本的には開発プロジェクト毎に決まっている．例えば，データ設計書であればそこに記述される項目は同じとなる．
- しかし，開発プロジェクトが異なれば，その流儀に合わせてスキーマが異なることが多く，スキーマが異なる設計情報を突き合わせたい場合も多い．

このような設計情報 XML に対して，以下のような要求が存在する．

設計情報の整合性のチェック　設計情報は複数の設計者・会社によって作成されるために，相互の矛盾が発生する場合がある．その設計情報間の整合性を機械的にチェックしたい要求がある．

設計書の相互関係の可視化　複数のサブシステム間をまたがる仕様や，データ仕様と処理仕様の関係等の確認のために，設計書間の相互関係を可視化したい要求がある．

設計情報の集計　設計情報を集計し，設計工程の進捗管理や，設計者の生産性等の評価指標を導きたい要求がある．

このような要求の中心となる処理は，複数の XML を突き合わせたり，その差異を求めたり，一つにまとめることとなる．そのような処理は，XQuery を利用しても可能ではあるが，このドメインに特化することによってさらなる利便性を得ることができる．例えば，複数の XML の一致性の判定は，構造を考慮する必要がある．データベースのカラムを特定するためには，カラム名だけでは不十分で，それが含まれるテーブルとデータベース両方を指定する必要がある．このような一致性判定は，構造のネストを再帰的に実行できることが望ましい．複数の XML の突き合わせにおいては，一致した複数の構造を再帰的に，一つにまとめるような処理も求められる．そして，これらは設計情報 XML の集合に対する和や差の操作となる．よって，本研究では再帰的な一致判定と，それを基とした再帰的な処理を含む集合操作によって構成される，構造化されたデータに対する集合演算をドメイン特化言語における操作として用意することを提案する．このような汎用的な集合操作は，XQuery と比較して，個別にタグを指定することをできるだけ最小限にし，XML を分解することなく，全体を指定した操作によって最終結果が得られるような操作としたい．このような操作は，すべての意味論を扱えない欠点はあるが，設計情報分野に特化すれば，個別毎のプログラムの作成を最小限に抑えることが期待できる．

3. 入れ子関係に対する再帰的代数とその XML 拡張の課題

3.1 入れ子関係に対する再帰的代数とその概要

前節のような，入れ子的なデータ構造に対して，再帰的な集合演算を行えるような既存技術の調査を行った．その結果，我々の求める集合演算に近い研究として，Colby による入れ子関係に対する再帰的代数 [4] の適用を検討した．古い研究（1989）であるが，入れ子関係の再帰的な処理が，本研究の問題意識（簡易・自動）に近い．まず，再帰的代数の例として，再帰的和のイメージを図 1 に示す．一般的に複数の集合の和とは，そのどれかに含まれる要素すべての集合となるが，再帰的和は何らかの基準で同じとみなされた（同一視された）レコードすべての集合であるが，その同じとみなされたときにその下位構造に対しても和を再帰的に適用する．例えば，設計書情報のドメインにおいて，複数人で書かれた設計書を一つにまとめるときに，大きな機能（図 1 の要素 A に相当）毎に，詳細機能（図 1 の要素 B に相当）をまとめたり，人毎に分類してまとめたり等の処理に利用できる．細かい指定なしで，設計情報をまとめあげる処理を実行できる．このような処理を XQuery で記述することは可能であるが，異なるタグに対して汎用に記述することは難しい．

3.2 再帰的代数の XML 拡張とその課題

[4] の再帰的代数は入れ子関係のレコードの集まりに対

Vol.2014-DPS-158 No.1
Vol.2014-CSEC-64 No.1
2014/3/6

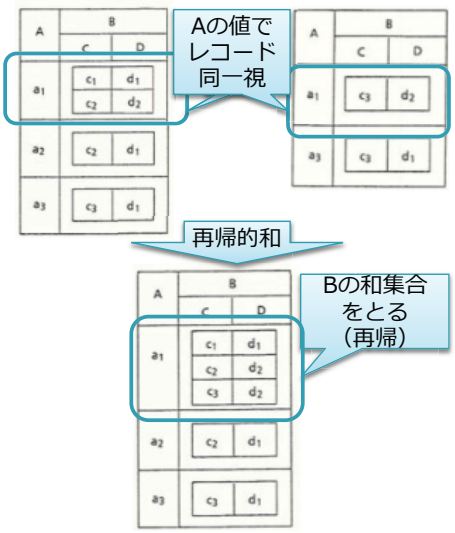

図 1 再帰的和の例

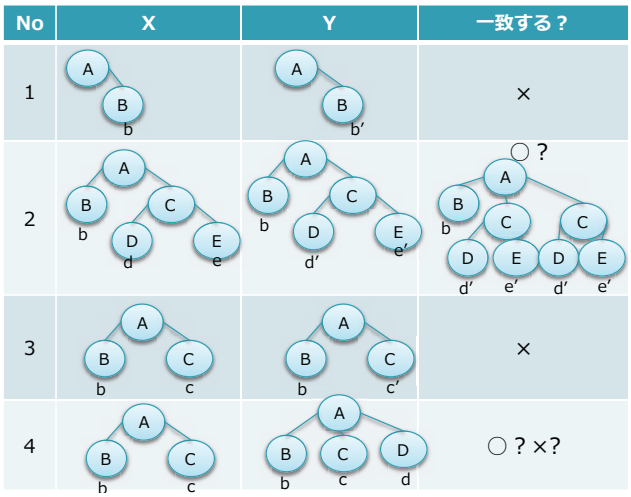

図 2 構造の異なる XML の同一視

して定義されていたが，その XML 拡張を考える．まず，XML に対する再帰的代数における演算は，その入力と出力は，XML 木の集まりの XML フォレストであるとする．以降，XML 木を，木のルートとなる要素と同一視し，入出力を要素の集まりと表現することがある．再帰的代数における演算は，レコードを同一視し，同一視されたレコードに対する下位階層のレコードで演算を再帰的に適用する演算であったから，

- 同一視の仕方
- 演算の結果の構造

の 2 つを XML 要素に対して決める必要がある．入れ子関係と XML の違いからすると，以下の 2 つの課題がある．

課題 1：構造の違いへの対応 入れ子関係は，構造が完全に一致することが前提とされているが，XML では構造に不揃いがありえる．例えば，設計書における XML では，同一システムを複数社で受け持つ場合に，マージすべき設計書の項目に違いがでることがある．

課題 2：繰り返しへの対応 入れ子関係でも繰り返しはあるが，必ず同じ構造のレコードが繰り返される．一方 XML では，繰り返しは同じ構造とは限らず起こりうる．

入れ子関係においては，下位に構造を持たないタプルの値によって同一性を判定していた．それをそのまま XML に適用するならば，子を持たない要素（Simple Type）の値によって同一視することになる．すると，図 2 の上の 3 つの例のように，同一視の有無が決定できるが，4 つめの例のように構造が異なる場合，共通する要素がすべて一致しているから同一視するのか，すべての要素が一致していないから同一視しないのか，と解釈が複数考えられる．

また，繰り返しのある XML を同一視する場合は，以下の 2 つの解釈が可能だが，データの意味や解釈（順番の有無，等）によって，どちらの案のほうが適切かは異なってくる．

- 案 1：位置で識別する．子要素の中で同じ位置にあるものは同一視すべきと考えて突き合わせする．子要素全体の中での位置で識別する方法と，同じ要素名の中の位置で識別する方法がある．
- 案 2：すべての組み合わせを突き合わせる．結果としては，左右の和集合が作成される．

4. 制約つき XML のための再帰的代数

本章では，前節において述べた課題に対する解決の方法について述べる．繰り返し等での解釈の任意性による複雑さを緩和するために，XML の表現能力に対して一定の制約を加え，構造の違いの同一視については，キーを利用した再帰的な同一視によって対処する．

4.1 制約つき XML

2 章で述べたソフトウェア設計情報 XML の構造的な特徴（構造は明確だが，スキーマの差異は存在．文字装飾のような不定的な構造はない等）と，3 章で述べた XML の突き合わせ・同一視を複雑にしない要求を考慮し，以下の制約を XML に対して仮定することとした．

- 繰り返し制約（R1）：ある要素の子要素に同じ名前が 2 つ以上出現する場合，すべての要素名は等しい．
- 順序性制約 1（O1）：同一階層の子要素どおしにおいて，要素名が異なる場合は順序性がない．
- 順序性制約 2（O2）：同一階層の子要素どおしにおいて，要素名がすべて同じ場合は順序性あり／なしを選択できる．

制約 R1 から，制約つき XML においては，すべての要素名が異なるパターン（パターン A）と同じ要素名が 2 つ以上となるパターン（パターン B）と，その組み合わせのみが許されることとなる（図 4）また，このようなパターンに限定するために，要素ノードとテキストノードの混在

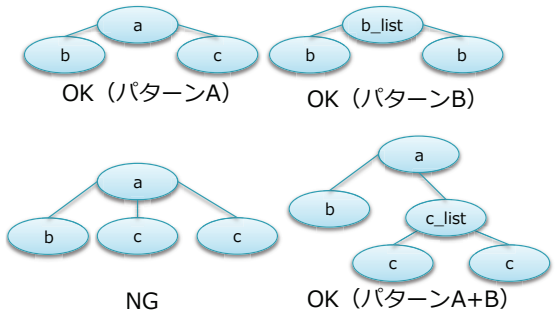

図 3 XML に対する制約

や，子要素として複数のテキストノードがあることは許されないとする．

この制約は，順序の考慮が限定的である点で，「データ的」であって「ドキュメント的」ではないが，パターン A における順序は，アプリケーション側で要素名を見て決定することができるし，パターン B を利用すれば，ドキュメント内で「章」が順序を持って繰り返される場合でも対応できるため，装飾のないソフトウェア設計書であれば，十分な仕様と言える．

この制約はスクリプト言語の持つデータ構造に近い．その用語に合わせ，制約 O1 の親要素と子要素の関係をマップと呼び，制約 O2 における親要素と子要素の関係で，順序ありの場合をリスト，順序なしの場合をセットと呼ぶこととする．

4.2 キーを利用した再帰的な一致判定

まず，キーを定義する前に，制約つき XML における基本的な要素の一致性について定義する．制約つき XML の（子要素を持つ）要素は，マップ・リスト・セットの 3 タイプに分類された．それぞれの一致性を以下のような自然な定義であるとする．キーの定義は，マップに対してのみ必要である．

- マップの一致：すべての子要素がそれぞれ等しいこと（両側にすべて存在）
- リストの一致：子要素の集まりがリストとして等しいこと（順序・数も一致）
- セットの一致：子要素の集まりが集合として等しい（順不同）

XML Schema においてキー定義は存在する．キー（xs:key）は要素に対して定義することができ，キーの名前，識別される要素であるセレクタ，識別するための要素であるフィールドを指定する．例えば，library という book の集まりからなる要素があり book を isbn で識別するときは，library の要素定義内にキー定義を作成し，セレクタが book，フィールドが isbn と指定すればよい．これは，XML 木の中のデータの制約であるためにこのような定義となっている．

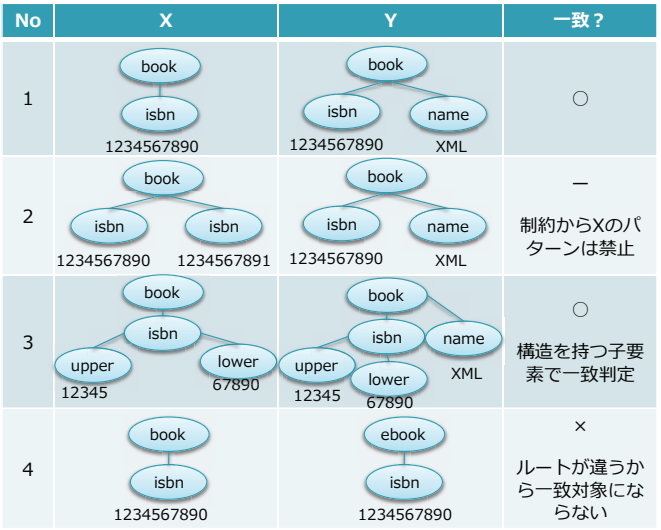

図 4 キーによる一致判定の例

それに対して，本研究におけるキーは，XML フォレストを対象とする演算において，その要素の同一性を識別することが目的となる．そこで「マップである同じ名前を持つ要素の集まりがあるとき，その要素を一意に識別できる子要素名前の集合をキーと呼ぶ．その一意性は再帰的に適用される」と定義する．

キーによる一致判定の例を図 5 に示す．この例では，book のキーが isbn で，isbn のキーが upper と lower であるとする．1 番目の例のように，isbn が等しければ book は一致すると判定される．2 番目の例は，XML 制約から，このようなパターンは出現しない．3 番目の例は book の一致を isbn で識別し，さらに isbn の一致を upper と lower で識別する再帰的な例である．4 番めのように要素の名前が異なる場合は一致対象にはならない．

4.3 単一化

演算結果の構造の決定という課題の解決のために，同一視された要素を一つにまとめる操作を「単一化」と呼ぶこととする．制約つき XML にための再帰的演算では，まず，集合演算するための要素の一致判定が行われ，一致された要素どおしを単一化によって一つにまとめ，最後に集合演算が適用される．単一化においては，設計書マージ等への応用を想定することにより，演算前の情報はできるだけ，演算後も保持し，再帰的に実行する方針とする．

単一化のルールを以下にまとめる．マップの単一化結果は，情報を保持する方針から，演算対象のすべての子要素を持つ要素を作ることとする．図 6-1 は B をキーとして単一化する例で，C・D 両方の子要素が単一化後も維持される．

セット（or リスト）の単一化は，キーによる判定ではなく，既にセット（or リスト）として同一であるから，結果は元と同じとなる（図 6-2）．

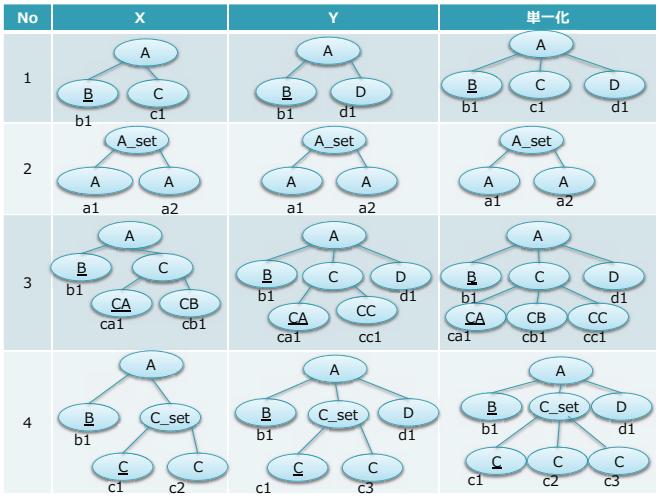

図 5 単一化

マップがさらに下位構造を含む場合は，下位構造がマップである場合，マップとしての単一化を行われる（図 6-3）．下位構造がセット（or リスト）である場合は，それらのセット（or リスト）をそれぞれ XML フォレストとみなして，元の XML フォレストに適用された演算を適用する．単一化操作に演算が影響するのは整理としてやや違和感はあるが，これは Corby 論文の再帰的演算の自然な拡張である．例えば，図 6-3 は再帰的和を行った場合であり，A がキー B により同一視されるときに，C_set は C からなる XML フォレストと解釈されて和演算が適用されている．

4.4 再帰的和と再帰的集合演算

前節までは演算を行う上での前提となる操作の話であった．本節から演算についての定義を述べる．再帰的な集合演算の代表として，再帰的和について説明する．その他の積・差等については同様に定義することができる．

再帰的和は，一般的な集合に対する和の直接な拡張であり，以下のように定義される．

- 2 つの入力の要素集合をそれぞれ突き合わせ，前に述べたキーを利用した再帰的な一致判定により，一致性を判定する．
- 一致した要素に対して，前に述べた単一化を行い，一つの要素へとまとめ，出力となる要素集合に追加する．
- 一致しなかった要素に対しては，そのまま出力となる要素集合に追加する．

よって，再帰的な演算の適用自体は，単一化の中に含まれており，この定義では表面にはでてきていない．再帰的和の例を図 7 に示す（わかりやすさのために表形式で示している）．キー isbn で同一視された要素内で review の和がとられている．

4.5 その他の再帰的演算：射影，選択，JOIN

Corby 論文においては，再帰的な射影演算・選択演算・

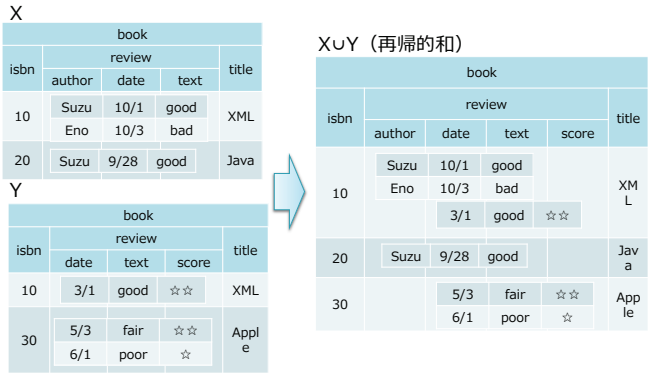

図 6 再帰的和

JOIN 演算も定義されている．しかし，これらは再帰的に演算が適用されるというよりは，入れ子構造に対する射影，選択と JOIN 演算になっており，入れ子がある場合に，どのような範囲で条件評価が行われ，どういう単位でレコードが残るのか，等といった定義である．これらを XML に直接拡張できるが，既存研究における XML の演算として議論されたものと類似しており，紙面の問題で記述は省略する．

5. Groovy を用いた制約つき XML 向け代数の実装とドメイン特化言語

5.1 Simple XML

これまで述べた制約つき XML に対する再帰的演算の試験的な実装を行った．実装の方針としては，最初に XML の課題として挙げた DOM の扱いにくさを低減するために，XML をできるだけスクリプト言語のデータ構造に近く扱えるようにしたいと考えた．制約つき XML はスクリプト言語のデータ構造に近い形で定義されていたし，スクリプト言語の DSL（ドメイン特化言語）的な機能を利用することによって，それは可能である．

実装イメージを図 8 に示す．この実装を Simple XML と呼ぶことにする．演算対象となる Simple XML のオブジェクトは，パース・XPath 等の XML 的な操作と，プログラミング言語のデータ構造としての操作を両方行うことができる．後者はスクリプト言語の DSL 機能を用いることによって可能となる．オブジェクトの内部には，XML 機能を持つオブジェクトとスクリプト言語のデータ構造の両者を持ち，適宜相互変換することによって，その内部は利用者から隠蔽する．

5.2 Groovy を用いたドメイン特化言語

Simple XML の実装言語として，Java と親和性の高いスクリプト言語として知られている Groovy を以下の理由で選択した．

- コレクション型が充実し，同等性の比較も用意される．
- XML を扱う機能が豊富に標準で用意されている．

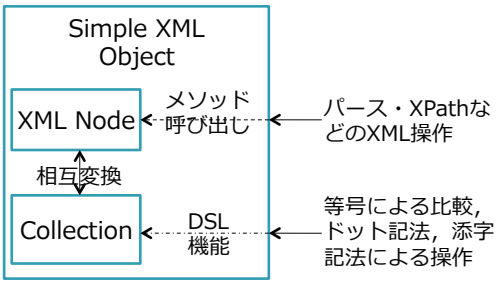

図 7 実装イメージ

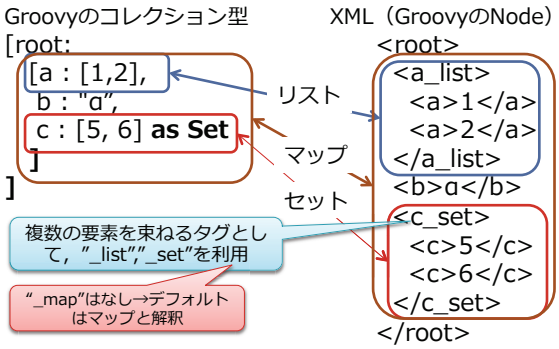

図 8 Simple XML モデルと Groovy オブジェクトの対応

- 言語としての能力を拡張できる DSL 機能が充実している．

Groovy のコレクション型は，リスト・セット・マップが用意されているので（それぞれ，[a, b, c]，[a, b, c] as Set，[a:1, b:2, c:3] と表現），制約つき XML の 3 つのパターンはそのままこれらにマッピングできる（図 9）．セットとリストの場合は，XML ではそのまとまりを表す要素（図中の a_list 等）を持つ．

Groovy においては，XML を便利に扱うために DSL 機能を有効に利用している．「.」や「」等のプログラミング言語における構成要素を用いて直感的に，XML の要素へアクセスすることができる．Groovy の XML 等のマークアップドキュメントを容易に作成するクラスである MarkupBuilder を用いると，例えば，そのクラスのオブジェクト doc に対して，doc.html {head {title(”links”)}} と指定することにより，ドキュメントを生成することができる．また，要素の取得にも「.」を利用できる．例えば，langs オブジェクトの子要素 language を langs. language として取得できる．得られた結果は each 等の言語としての機能を自然に利用して処理できる．

このような DSL 機能は，メソッドが適用されたときの振る舞いを変更することにより実現される．XML の例では，「子要素名と一致するメソッドが適用される場合，子要素を返却する」という振る舞いに変更している．

Simple XML では，その演算（union，intersection，diff 等）を制約つき XML の集合を表すクラスに対するメソッドで，制約つき XML の集合を返却するものとして実現している．よって，演算の組み合わせは Groovy におけるメソッドチェーンによって行う．

さらに，上記の XML 向け DSL と同様な扱いを利用し，要素名を「.」を利用して取得可能としている（演算と同じ名前の要素名が使用できない制約は生まれる）ため，演算とプログラミング言語における構成要素を組み合わせることによって，より直感的に演算とその結果の処理を行えるようにしている．

6. 考察

6.1 既存技術との関係：データ表現能力

制約つき XML は，マップ・リスト・セットの組み合わせでデータを表現するとした．これは，スクリプト言語におけるデータ構造に近いデータ表現能力である．これらは Groovy の場合，そのままマップ・リスト・セットに対応しており，JavaScript であればオブジェクトと配列に対応する（セットはない）．また，Google がデータ交換フォーマットとして提案している Protocol Buffers[5] の表現能力も近い（セットはない）．しかし，例えば，Groovy のコレクション型と制約つき XML は，完全には一致しない．制約つき XML では，[[1, 2], 3] のようなリストの入れ子や，[a:[1, 2]] のような値がリストとなることは除外されている．つまり，制約つき XML は，「入れ子構造に対して必ずタグ付けされているコレクション型オブジェクト」に対応している．これは XML の元々の思想である，データの意味をタグ（メタデータ）で明確に表現すべきということが反映されている．

6.2 既存技術との関係：データ交換フォーマットとスキーマ

現在最も普及しているデータ交換フォーマットは JSON（JavaScript Object Notation）であろう．JSON は，JavaScript のオブジェクトに直接マッピングできる点が優れており，本研究も影響を受けている．Protocol Buffers もデータ交換フォーマットとして普及しつつある．

制約つき XML は，XML を基礎とした技術であるので，XML スキーマを利用したスキーマが明確に定義できるメリットがある．一方，JSON Schema も提案されている [6] が，まだ，一般的に普及しているとはいえず，ツールの整備も XML ほどは進んでいない．Protocol Buffers はスキーマを明確に定義でき，それによる高速化のメリットが特徴である．しかし，標準との整合性という点で XML ベースの技術に劣っている．

6.3 既存技術との関係：XML 処理系

XML の DOM 操作の煩わしさを軽減させるための既存の提案が存在する．Groovy の XMLSluper は，XML に対してオブジェクト的なアクセスを実現するものである，

jQuery は，HTML の DOM が対象であり，ややアプローチが異なるが，DOM の煩わしい多段なループを軽減できる．

これらの技術に対して，Simple XML は制約を加えることとスクリプト言語の DSL 機能を利用することによって，よりスクリプト言語との親和性を増すことを目指していると言える．ドメイン特化の処理をライブラリとして用意する手段もあるが，ライブラリ開発は要求条件を事前にすべて把握する必要があり，カスタマイズの予知もパラメータのみであり大きくない．よって，適用範囲が限定的である場合には有効である．それに対して，DSL は，手続きも含めたカスタマイズの余地があることが特徴であり，適用範囲がやや広い場合に対して有効な手段と考えている．

6.4 一次評価

XML 化した設計書のチェック処理に対する適用可能性について，実際に利用された設計書の XML 例と，XML の整合性をチェックする XQuery 例を入手することによって検討した．実際に移植作業や，何%程度適用可能か等の網羅的な検討も行っておらず，ごく初期的な一次評価であるが，以下のような知見を得た．

- XML に対する制約の妥当性：直接的に適用できない例は存在したが，相互変換により制約つき XML を適用できる見込みがあった．適用できない代表的な例は，「< 設計書 >< 名前/>< 項目/>< 項目/>< 設計書/>」のように，同じ階層に異なる項目を含み（マップ）かつ繰り返し（セット）が発生するパターンである．しかし，その構造のセマンティクス（順序やまとまり）が，設計書管理のアプリケーションにおいて利用されていないため，相互変換によって適用可能と考えられた．
- 提案 DSL の機能的な充足性：設計書の整合性チェックにおける XQuery を簡易に記述できる見込みが得られた．例えば，ある設計書 XML のリストに，別の設計書 XML のリストが含まれるかを判定する処理が頻出するが，XML 演算における射影演算と差演算によって簡潔に置き換え可能であった．ただし，整合性チェックにおいては，再帰的演算の有効な事例は確認できず，設計書のマージ処理に期待できるために今後検証していきたい．

6.5 今後の課題

既存 XML 文書への円滑な適用と検証　XML に対して制約を与えているために，任意の制約のない既存 XML 文書に対して，そのままでは適用できない．例えば，セット・リストの要素名については，_set，_list を付加するルールになっていたが，既存 XML 文書の XML スキーマ等の型情報を利用して変換する方法が考えられる．そのためには，XML スキーマ等を基礎とした制約つき XML の表現能力の位置づけ等を理論的に整理する必要がある．

演算処理の性能向上　提案した演算を，大規模な設計書情報に適用するには，高速化の工夫が必要になる．元々再帰的演算を提案した Corby の論文では，演算の交換関係を利用した最適化により高速化を図っている．同様の最適化が本研究にもできるかを検討する余地がある．また，XPath 等の XML 系操作に演算を拡張が望ましいが，そのとき演算結果は Simple XML に閉じることが望ましく，その検討が必要である．また，より一致判定を柔軟にするためには，異なる階層間の同一視，等が必要である．

実装面の改善　現在の実装は，Groovy 言語の特性を十分活かしきれてはいない．動的なメソッド実行・クロージャを利用した一致判定のカスタマイズ等により柔軟性や使い勝手が向上すると思われる．また，他のより普及している Ruby，JavaScript 等への言語への移植も検討したい．

7. おわりに

データ処理向けの XML 開発における課題を整理し，XML の一般性を制約することにより，生産性の向上等を狙う，制約つき XML によるドメイン特化言語を提案した．プログラミング言語 Groovy を利用したドメイン特化言語の実装方式を提案し（Simple XML），再帰的和や差分抽出等の演算の実装を行った．今後は，実際の適用評価を進めつつ，性能改善等の課題を解決していく．

謝辞　本研究にさまざまなご教示をいただいた NTT データシステム技術の遠城秀和氏に深謝いたします．

参考文献

[1] M.Fowler: Domain-Specific Languages, Addison-Wesley (2010).
[2] F. Dearle: Groovy for Domain-Specific Languages, Packt Publishing (2010).
[3] モデリングツール Enterprise Architect, http://www.sparxsystems.jp/products/EA/ea.htm
[4] L.S. Colby: A Recursive Algebra and Query Optimization for Nested Relations, SIGMOD Conference (1989).
[5] Protocol Buffers, https://developers.google.com/protocol-buffers/
[6] JSON Schema, http://json-schema.org