項目間の対応関係を用いたXBRL財務報告書自動変換手法の提案

高尾 祐治
大阪大学大学院情報科学研究科
y-takao@ist.osaka-u.ac.jp

渡辺 貴史
大阪大学基礎工学部
t-watanb@ics.es.osaka-u.ac.jp

松下 誠
大阪大学大学院情報科学研究科
matusita@ist.osaka-u.ac.jp

井上 克郎
大阪大学大学院情報科学研究科
inoue@ist.osaka-u.ac.jp

湯浦 克彦
株式会社日立製作所
yuura@bisd.hitachi.co.jp

Abstract

近年, *XBRL* と呼ばれる *XML* を基盤とした財務報告書のための記述言語が提案され, 計算機上で財務情報を扱う標準的な枠組みに関する研究が進められている. 財務報告書は企業の活動結果を外部に報告する際, 一般に用いられる文書であり, 文書の *XBRL* 化によって財務情報の交換や分析作業のコストを削減することが可能となる. しかし, *XBRL* 文書の作成作業は依然として人手に依存する要素が多いため, 誤記や編集ミスによって財務報告書の信頼性が低下するという問題がある.

そこで本論文では, 文書作成の効率化と信頼性の向上を目標として, 既存の *XBRL* 文書から異なる形式の *XBRL* 文書へ自動的に変換する手法を提案する. 本手法では, *XBRL* 文書に記述されている項目の対応関係を定義することで *XBRL* 文書の自動変換を実現する. また, 各対応関係については, 条件節やデータ加工式を付加することが可能である. これによって, データ抽出や表の再構成といった柔軟な変換を行うことができる.

さらに, 本手法を用いた *XBRL* 文書間の自動変換ツールを試作し, 適用実験を行った. その結果, 本ツールを用いることにより, 信頼性の高い財務報告書作成を効率的に行うことが可能になることを確認した.

1. まえがき

企業の財務情報を広報する際には, 一般に財務報告書が用いられる. しかし, 財務情報を表現する統一されたデータ形式がなく, システム同士でデータを受け渡すことは非常に困難であった. このため, 社内の会計情報から社外に公表する財務報告書を円滑に作成できず企業の情報開示に非常に時間がかかっていた. このために, 投資家は古い情報を元に企業への投資判断をすることになるという大きなリスクが存在した.

この問題を解決するために, XBRL と呼ばれる XML を基盤とした財務報告書のための記述言語が提案され, 計算機上で財務情報を扱う標準的な枠組みに関する研究が進められている. 財務報告書は企業の活動結果を外部に報告する際, 一般に用いられる文書である. 財務報告書の XBRL 化によって電子的な財務情報の流通, 再利用を行うことができる. そのため, 分析作業のコストの削減と正確な財務情報の迅速な提供が可能となり, 企業や, 政府系機関での採用が進んでいる [8].

財務報告書の会計基準は国, 業種, 企業によって異なり, 内容や形式にも, 企業グループでの連結会計の導入, 国際関係基準への段階的移行などにより変更が加えられている. このように財務報告書は, 同じ内容を表す文書であっても多様な形態を持つ必要があるため, その作成作業は複雑なものとなる. しかし, XBRL 文書の作成作業は依然として人手に依存する要素が多いため, 誤記や編集ミスによって財務報告書の信頼性が低下するという問題がある.

そこで本論文では, 文書作成の効率化と信頼性の向上を目標として, XBRL 文書間の自動変換手法を提案する. 財務報告書には様々な形式が存在するが, 同じ財務活動についての報告であれば, その内容には同じ項目が含まれる. すなわち, 財務情報として等価な項目であれば, その値も等価か, あるいは定式による加工で相互に変換可能である. また, 必要なデータと不必要なデータを識別ができれば, 必要なデータの値のみをもつ文書を生成することができる. これらのことから, 項目間の対応関係, 必要なデータを識別するための条件, 値の変換式の 3 点についての情報を得ることにより自動変換を行う.

しかし, XBRL ではこれらの情報を表す方法は定義されていないため新たに対応関係定義, 条件文定義, データ加工文定義を導入する. これらの情報を適切に設定することで XBRL 財務報告書の自動変換を可能とする.

さらに本研究では, 提案する手法を用いて XBRL 文書間の自動変換を行うツールの試作を行い, 適用実験を行った.

以下, 2 節では XBRL について, 3 節では提案する自動変換手法で用いる対応定義について, 4 節ではアルゴリズムについて説明する. 5 節では試作した XBRL 財務報告書自動変換ツールと適用実験について述べ, 6 節では考察を述べる. 7 節ではまとめと今後の課題について述べる.

2. 財務報告書記述言語 XBRL

XBRL は, 各種財務報告書用の情報を作成・流通・利用できるように標準化された, XML をベースとする言語である. 財務情報を XBRL 文書化することにより, ソフトウェアやプラットフォームに依存しない電子的な財務情報の作成や流通・再利用が可能となる.

2.1. 財務報告書

財務報告書とは, 企業が定期的に公表する貸借対照表, 損益計算書など財務諸表を含む報告である. 例として, 商法による貸借対照表を表 1 に示す. 貸借対照表とは, その企業の資産・負債・資本の残高が記述された表であり, 企業の財政状況をわかりやすく表現している.

財務報告書の作成においては, 多くの会計データ, 企業組織等に関する説明文書, 監査人の報告などの様々な素材情報をもとに, いくつかの種類の報告を作成しなければならない. 財務報告書の会計基準は国, 業種, 企業によって異なり, 内容や形式にも, 企業グループでの連結会計の導入, 国際関係基準への段階的移行などにより変更が加えられている. さらに, 事業別, 地域別, セクション別の財務報告書や四半期ごとの報告を求められることもある. このように財務報告書は多様な形態を持つ必要があるため, その作成作業は複雑なものとなる. よって, 財務報告書の作成作業を効率よく行うことは非常に重要である.

表 1. 貸借対照表

会社名	貸借対照表 （平成×年×月×日現在）		(単位：○円)
(資産の部)		(負債の部)	××××
流動資産	××××	流動負債	××××
現金・預金	×××	支払手形	×××
受取手形	×××	買掛金	×××
売掛金	×××	短期借入金	×××
有価証券	×××	短期償還社債	×××
製品	×××	未払金・諸税金	×××
半製品・仕掛品	×××	前受金	×××
原材料・貯蔵品	×××	その他	×××
その他	×××	固定負債	××××
貸倒引当金	△×××	社債	×××
固定資産	××××	長期借入金	×××
有形固定資産	××××	その他	×××
建物・構築物	×××	(資本の部)	××××
機械・装置	×××	資本金	××××
工具・器具・備品	×××	法定準備金	××××
土地	×××	資本準備金	×××
建設仮勘定	×××	利益準備金	×××
無形固定資産	××××	剰余金	××××
工業所有権	×××	○○準備金	×××
その他	×××	○○積立金	×××
投資等	××××	別途積立金	×××
投資有価証券	×××	当期未処分利益 (損失)	×××
子会社株式・出資金	×××	〔当期利益 (損失)〕	×××
長期貸付金	×××		
その他	×××		
貸倒引当金	△×××		
繰延資産	××××		
開発費	×××		
合　計	××××	合　計	××××

2.2. 財務報告書の XBRL 化

アナリストや投資家は, 財務報告書を比較分析することで投資対象を決定するため, 鮮度がよく, 精度が高く, 扱いやすい財務情報の提供が重要となる. しかし, 財務情報は複雑である上に, その作成作業は人手に依存する部分が多い. そのため, 誤記や編集ミスが発生しやすい. また, 会計基準が異なる財務報告書を比較することも困難である.

これらの問題を解決するため, 2000 年 7 月に米国公認会計士協会によって XBRL が提案された [3]. XBRL は, XML をベースとする財務報告書記述のための標準化言語である. XBRL は異なる形式の財務報告書を同一の枠組みを用いて定義できるため, 財務情報を柔軟に記述することができる. 財務情報を XBRL 文書化することで, 財務報告書の作成を正確に, かつ効率的に行うことができる. さらに, [1] で紹介されているように, 財務データを共通の XBRL 文書に変換することにより, 財務報告書の比較を容易に行うことができる.

XBRL 文書はタクソノミ文書とインスタンス文書から成る. タクソノミ文書はタクソノミ本体 (以下, タクソノミ) とリンクベースから構成され, 財務報告書の文書形式を定義する. インスタンス文書はタクソノミ文書によって定義された形式で財務情報が記述された文書である [4]. 図 1 に, XBRL 文書の構成図を示す.

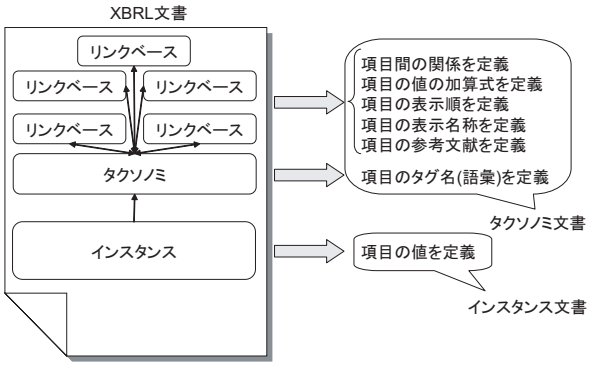

図 1. XBRL 文書の構成

2.3. タクソノミ

タクソノミは, インスタンス文書の語彙 (要素名, 属性など) を定義する XML Schema[6] である. 語彙の定義は次のように記述される.

```
<element
 name="assets"
 type="xbrli:decimalItemType"
 id="assets"
 xbrli:balance=""
 substitutionGroup="xbrli:item"/>
```

name 属性で要素名を, type 属性で要素の型を指定する. 要素の型は以下の 6 種類である.

- 数値型
  - xbrli:decimalItemType
  - xbrli:monetaryItemType
  - xbrli:sharesItemType

- 文字列型
  - xbrli:stringItemType
  - xbrli:uriItemType
  - xbrli:dateTimeItemType

また, id 属性でこの要素の ID を指定し, xbrli:balance 属性でその要素の表す財務情報が資本情報なのか負債情報なのかを指定する.

2.4. リンクベース

リンクベースは, タクソノミで定義された項目間の関係や, 各項目に対する追加情報を XLink の外部リンク機能を利用して定義した文書である. リンクベースには次の 5 種類がある.

定義リンク	項目間の関係を定義
計算リンク	項目の値の加算式を定義
プレゼンテーションリンク	項目の表示順を定義
ラベルリンク	項目の表示名称を定義
リファレンスリンク	項目の参考文献を定義

本手法では, 定義リンクと計算リンクのみを用いて自動変換を行うため, ここではこの 2 つについて説明する.

2.4.1. 定義リンク

定義リンクは要素を指定するためのロケータと, 抽象的な意味でどのように関連しているかを表す定義アーク要素により, 要素間の関係を定義するものである. ロケータは次のように記述される.

```
<loc
 xlink:type="locator"
 xlink:href="tax.xsd\#assets"
 xlink:label="tax_assets"
 xlink:title="assets"
 xlink:role=
  "http://www.xbrl.org/
   linkprops/locator/root"/>
```

- xlink:href 属性
  タクソノミで定義されている要素を参照するための属性である.
- xlink:label 属性
  xlink:href 属性で参照される要素に対するリンクベース内でのラベルを指定する. 要素間の関係定義はすべてこのラベルを用いて記述される.
- xlink:role 属性
  要素の役割を指定する. 上記の例の場合はこの要素が要素の木のルートであることを示している.

また, 定義アーク要素は次のように記述される.

```
<definitionArc
 xlink:type="arc"
 xlink:show="replace"
 xlink:actuate="onRequest"
 xlink:from="tax_assets"
 xlink:to="tax_currentAssets"
 xlink:title="Go Up to tax_currentAssets"
 xlink:arcrole=
  "http://www.xbrl.org/
   linkprops/arc/parent-child"/>
```

- xlink:from 属性, xlink:to 属性
  定義アーク要素によって関連付けられる要素を指定する.

- xlink:arcrole 属性
  要素間の関係 (親から子, 子から親) を定義する.

2.4.2. 計算リンク

計算リンクは, ロケータと, 子要素の値の計算法を表す計算アーク要素により計算に必要な情報を記述するものである. ロケータの記述は定義リンクと同様である. 計算アーク要素は定義アーク要素の属性に重みを表す weight 属性が加わる. 計算アーク要素は次のように記述される.

```
<calculationArc
 xlink:type="arc"
 xlink:show="replace"
 xlink:actuate="onRequest"
 xlink:from="tax_currentAssets"
 xlink:to="tax_assets"
 xlink:title="Go Up to tax_assets"
 xlink:arcrole=
  "http://www.xbrl.org/
   linkprops/arc/child-parent"
 weight="1"/>
```

weight 属性は子要素の値から親要素の値を計算するときの子の値の重みを指定するための属性である.

2.5. インスタンス文書

インスタンス文書は, タクソノミ文書で定義された要素で財務情報を記述した XML 文書である. 各要素はコンテキスト属性によってコンテキスト要素と関連付けられていなければならない. コンテキスト要素とはインスタンス文書内で要素に関する財務情報を記述したもので, 要素の値の単位や精度などの情報を付加する. 要素は, どのコンテキスト要素と関連付けられているかによって, 同じ名前の要素であっても識別が可能である.

2.6. 問題点

財務情報を XBRL 文書化することにより, データの比較・分析作業が効率化され, また, 情報の電子化とインターネット XML 基盤の利用により情報の提供速度が向上した. さらに XBRL 文書からの財務報告書の自動生成などにより効率と信頼性も向上した.

財務報告には様々な種類があり, 異なる形式の財務報告を作成するという作業が頻繁に行われる. しかし, XBRL 文書から別の XBRL 文書へと変換する作業は手作業の部分が多い. そのため, XBRL 文書の変換はミスが入りやすく効率の悪いものとなっている.

そこで, この問題を解決するために XBRL 文書間の自動変換を考える. 異なる形式の XBRL 文書を既存の XBRL 文書から自動で変換することができれば, 財務報告書の作成作業がより効率化され, 信頼性も一層向上すると考えられる.

3. 項目間の対応定義

財務報告書には様々な形式が存在するが, ある財務活動に関する異なる形式の報告には, 等価な項目が含まれる. 財務情報として等価な項目は, その値も等価かであるか, もしくは定式による加工で求めることができる. また, 必要なデータと不必要なデータの識別を行うことで必要なデータの値のみをもつ文書を生成することができる. これらのことから, 項目間の対応関係, 値の変換式, 必要なデータを識別するための条件の 3 点についての情報を使うことで XBRL 文書の自動変換を行うことができる. しかし, XBRL 文書には他の XBRL 文書の項目との対応に関する情報は記述されていない.

そこで本手法では, XBRL 文書中に記述される項目について, 対応定義, 条件文定義, データ加工文定義という 3 つの定義を新たに導入する. これら 3 個の定義を XML の要素として表現し, それらを記述した XML 文書である対応定義文書と, 変換元, および変換先のタクソノミ文書を使って既存のインスタンス文書を異なる形式のインスタンス文書へと変換する. 以下, 本手法で導入する定義について述べる.

3.1. 対応定義

対応定義では, 変換元 XBRL 文書で記述されている項目と, 変換先 XBRL 文書で記述すべき項目の対応を定義する. 項目間の対応はタクソノミで定義された要素を関連付けることで表し, ロケータとアーク要素に加えて, 条件指定を行うための要素を用いて記述する. ロケータは定義リンクや計算リンクと同様である. アーク要素は次のように記述する.

```
<equivalentArc
 xlink:actuate="onRequest"
 xlink:arcrole="element-element"
 xlink:from="tax_assets"
 xlink:show="replace"
 xlink:title="equivalent"
 xlink:to="tax2_assets2"
 xlink:type="arc"
 calculation="true"
 value="true"
 cond="cs1"/>
```

– xlink:arcrole **属性**
アークで対応付けられる要素の種類を表す. 属性値は次の 4 種類である.

element-element	: 要素と要素を対応付ける
element-attribute	: 要素と属性を対応付ける
attribute-element	: 属性と要素を対応付ける
attribute-attribute	: 属性と属性を対応付ける

– calculation **属性**
子要素の値を利用して親要素の値の計算を行うかを指定する. 値には true か false をとり, true であれば値を計算する. 既定値は false である. 変換元のインスタンス文書の要素値をそのまま記述できない場合にはこの属性により値を求める.

– value **属性**
変換先のインスタンス文書の要素に値が必要であるかを指定する. 値としては true か false をとり, true であれば値を記述する. 既定値は true である. 変換元のインスタンス文書の要素の値をそのまま記述できず, さらに計算によっても正しい値が求められない場合にはこの属性により値を記述しないように指定することができる.

– cond **属性**
対応が成立する条件がある場合にその条件を設定する. 条件自体と, 条件が成立した後の処理内容については, 後に述べる条件文定義によって記述する. ここでは, 定義された条件文につけられた ID をその値として持つ.

3.2. 条件文定義

条件文定義では, 変換元データの値によって要素を変換先 XBRL 文書に記述するかどうか, または, 値を加工するかどうかを定義する. 以下の 4 種類の要素を用いて条件文を定義する.

- conditionalStatement 要素
  condition 要素, formula 要素, str 要素を使い, 条件と値の加工を指定する. 加工には, 数値の場合は四則演算, 文字列の場合は結合を用いる.
- condition 要素
  条件の定義をする.
- formula 要素
  数値の加工に用いる定式を定義する.
- str 要素
  文字列の加工に用いる文字列を定義する.

以下, 各要素の説明を行う.

3.2.1. conditionalStatement 要素

アークによる対応付けの条件とその判定結果による値の加工を指定するための要素で, 以下のように記述する.

```
<conditionalStatement
 if="exp1"
 then="n * f1"
 else="n * 100"
 id="cs1"
 type="numeric"/>
```

– id **属性**
要素間の対応を定義するアーク要素からから参照される ID である.

– if **属性**
condition 要素を用いた条件式の指定を行う. 指定は後に説明する condition 要素の id 属性と and, or, (,) を用いて行う. 条件が真であれば対応関係が成立し, 偽であれば対応関係は不成立となる. 対応関係が不成立の場合, その要素を生成しない.

値を加工する場合には, 条件が真ならば then 属性で指定する式で, 偽ならば else 属性で指定する式で値を加工する. この属性は省略可能で, 省略された場合には条件が真であるとして then 属性で指定された値の加工を行う.

– then **属性**
条件判定が真である場合に用いられるデータ加工式を指定する. 指定は値が数値ならば formula 要素の id 属性と+, -, *, /, (,), n を用いて行う. また, 値が文字列ならば str 要素の id 属性と結合演算の+, 加工する前の値を表す n を用いて行う.

formula 要素と str 要素を用いずに指定することも可能だが, 文字列の加工で文字としての “+” や “n” を使用するには str 要素で定義する. この属性は省略可能で, 省略されて条件判定が真となった場合には加工は行わない.

– else **属性**
条件判定が偽の場合のデータ加工式を指定する. 指定方法は then 属性と同様である. この属性も省略可能である. しかし, 条件判定の結果が偽, かつ else 属性が省略された場合には, 対応関係が不成立となり対応する要素を生成しない.

条件処理の流れを図 2 に示す. 条件判定の結果と各属性の有無により最終的に得られる値は「加工しない」, 「then 属性の式で加工」, 「else 属性の式で加工」, 「対応する要素を生成しない」の 4 パターンとなる.

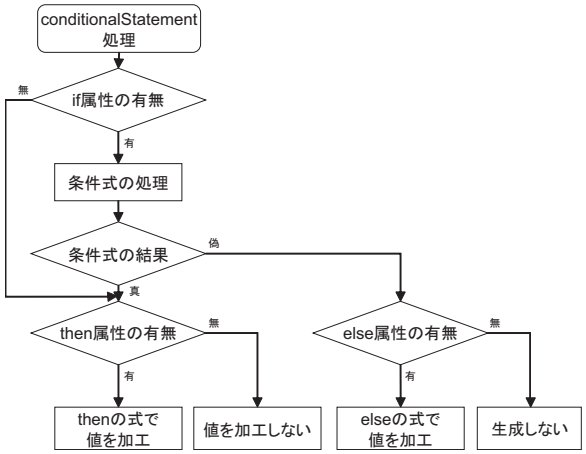

図 2. 条件処理の流れ

3.2.2. condition 要素

条件の具体的な内容を指定するための要素で, 以下のように記述する.

```
<condition
 id="exp1"
 xlink:href=
  "C:\test\testData\tax.xsd#assets"
 value="10"
 operator="greater than"
 type="element"/>
```

– id **属性**
conditionalStatement 要素から参照される ID を設定する.

– xlink:href **属性**
条件の対象となる要素あるいは属性を指定する. id 属性へ記述された ID が要素の場合にはこの属性は省略可能で, 省略されると条件が指定されたアークの xlink:to 属性の要素を指定したことになる.

– value **属性**
条件の判定基準となる値を設定する.

– operator **属性**
条件判定が真になるための, xlink:href 属性で指定された要素あるいは属性の値と value 属性値との関係を設定する. value の値に対する href で指定された要素 (属性) の関係として以下の 6 種類を定義した.

- greater than
- greater or equal
- less than
- less or equal
- equal
- not equal

– type **属性**
xlink:href 属性の参照が要素へのものならば element を, 属性へのものならば attribute を値として持つ. この属性は省略可能で, 既定値は element である.

上記の例では「tax.xsd というタクソノミで定義される要素 assets の値が 10 よりも大きい場合に真」という条件になる.

3.3. データ加工文定義

加工文定義では, 値を加工する定式を定義する. 数値の加工と, 文字列の加工の 2 種類を定義する.

3.3.1. formula 要素

数値の値を加工するときに利用する定式を定義するための要素で, 以下のように記述する.

```
<formula
 id="f1"
 exp="n + 10"/>
```

– id **属性**
conditionalStatement 要素から参照される ID である.

– exp **属性**
値の加工に使用する定式を設定する. 式には+, -, *, /, (,), n を使用できる. ここで, 変数 n はアークの対応付けによって取得した変換元のインスタンス文書の要素の値を表す. 上記の例の場合, “n + 10” は取得した値に 10 を加えるという意味になる.

3.3.2. str 要素

文字列の値を加工するときに利用できる文字列を定義するための要素で, 以下のように記述する.

```
<str
 id="s1"
 exp="START + n"/>
```

– id **属性** n
conditionalStatement 要素から参照するための ID である.

– exp **属性**
値の加工に使用できる文字列を設定する. formula 要素とは異なり, 属性値の文字列はそのままの形で使用される. 上記の例の場合, アークによって取得した値が “ABC” であったとしても str 要素が表す文字列は “START ABC” ではなく “START + n” になる.

4. XBRL 文書変換アルゴリズム

本アルゴリズムは, 変換元, 変換先のタクソノミ文書, 変換元インスタンス文書, 対応定義文書を入力として受け取り, 変換されたインスタンス文書を出力する. 変換は以下の手順で行う.

1. インスタンス文書情報の取得
2. 対応する要素の取得に利用するための, 変換元インスタンス文書の木を構築
3. 項目間の対応付け情報の取得
4. 以下の処理の再帰的な繰り返しによる, 変換先インスタンス文書の木の構築

   4-1. 対応関係を利用した対応する要素の値の取得
   4-2. 条件判定, および, 値の加工
   4-3. 子要素の追加
   4-4. 要素の追加
   4-5. 属性の処理

5. 構築した木をインスタンス文書として出力する

以下, 各処理について説明する.

4.1. インスタンス文書情報の取得

変換元のタクソノミ文書と, 変換先のタクソノミ文書からインスタンス文書の情報を取得する. 具体的には, 以下の 4 種の情報を取得する.

– **名前空間接頭辞の取得**
タクソノミと定義リンク, 計算リンクの名前空間に関する情報を取得する. 取得された名前空間接頭辞は要素や属性の値を取得する際に必要となる.

– **タクソノミからの情報の取得**
タクソノミで定義されている要素の情報を取得する.

– **定義リンクからの情報の取得**
定義リンクから各要素の関係情報を取得する. すべての定義アーク要素を調べ, アークの上書きがあればその情報によって上書きする. 同じ要素間で上書きの優先度を示す priority 属性の値が等しいアークが複数存在する場合には, 先に取得したアークの情報を優先する.

– **計算リンクからの重み情報の取得**
計算リンクから各要素の重みを取得する. すべての計算アーク要素を調べ, アークの上書きがあれば, その情報によって上書きする. 同じ要素間で priority 属性の値が等しいアークが複数存在する場合には, 先に取得したアークの情報を優先する. 最後に定義リンクで取得したアークに計算リンクから取得した重みの情報を追加する.

4.2. 変換元インスタンス文書の木の構築

取得した情報を元に, 変換元のインスタンス文書の木を構築する. この木は値の取得のために変換元のインスタンス文書の要素を探すときに, 要素間の関係を調べるのに利用する.

木の構築を行うために, まずルート要素を取得する. ここでいうルート要素とは, タクソノミやインスタンス文書全体のルート要素のことではなく, タクソノミで定義された要素で構成される木のルート要素のことである. 定義リンクでは要素間の親子関係が定義されているが, ルート要素は親要素を持たないため, 親を持たない要素をすべてルート要素として取得する.

定義リンクから得た情報で要素間の親子関係がわかるので, 子の要素の生成と追加をルート要素からの再帰処理により木を構築する.

4.3. 項目間の対応付け情報の取得

まず名前空間情報の取得を行い, 次に要素間の対応情報を取得する. 項目間の対応に指定する条件があればその情報も取得する.

4.4. 変換先インスタンス文書の木の構築

取得した情報を元に, 変換先のインスタンス文書の木を構築する. 処理は木のルート要素から開始する. 以下, 各処理について説明する.

1. **対応する値の取得**
処理する要素の名前を元に対応付け情報を調べ, その要素と対応付けられた変換元のインスタンス文書の要素の名前を取得する. 変換元のインスタンス文書から, 取得した名前で要素を探し, 値を得る.

2. **条件判定**, **および**, **値の加工**
条件付き対応の場合の条件の判定, 値の加工を行う. そ

れぞれの処理について説明する.

- 条件付き対応の場合の条件の判定
  アーク要素で条件が設定されていれば conditionStatement 要素の情報から条件式を取得し, 判定を行う.

- 条件式の評価による値の加工処理の決定
  条件式の判定結果により値の加工を行う.

- 値の加工
  conditinalStatement 要素の情報から加工する値の型と加工の式を取得して加工する.

3. **子要素の追加**
子要素は定義リンクの情報から得られるので, すべての子要素について処理を行う. また, 要素の値を子要素の値から計算しなければならない場合には, 子要素を追加する際に子要素が持つ値と重みを乗算する. この値をすべての子要素について計算して和をとり, 要素の値とする.

- 子要素の値を用いた親要素の値の計算
  図 3 は子要素の値を用いた親要素の値の計算の例である. assets, currentAssets, noncurrentAssets の値はそれぞれの子要素の値に重みをかけたものの和になる.

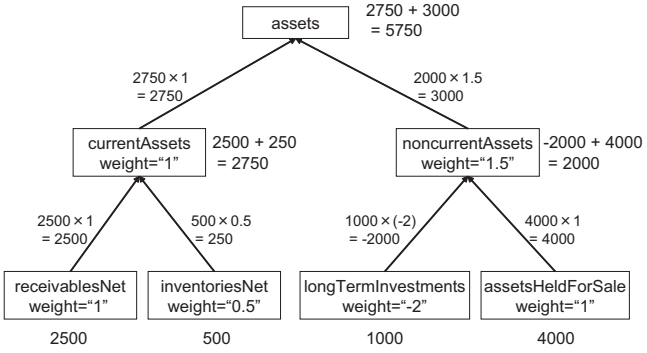

図 3. 計算処理の例

- 子要素の追加生成
  図 4 は子要素の生成処理において対応する要素が変換元のインスタンス文書に複数記述されている場合の処理の例である. balanceSheet2 要素の子要素である assets2 要素の生成処理において, 対応する assets 要素が 3 つ記述されている. しかし, これら 3 つの assets 要素は会計年度などが異なっており, それぞれ別々の情報を表している.

このような場合には, 変換先のインスタンス文書でも 3 つの要素を記述する. 変換元のインスタンス文書で記述されている数 (この場合は 3 つ) だけ子要素を追加生成し, それぞれ別々に値を取得する. この例の場合は 2 つの子要素を追加生成して処理をしている. そして処理が終了した子要素を追加する.

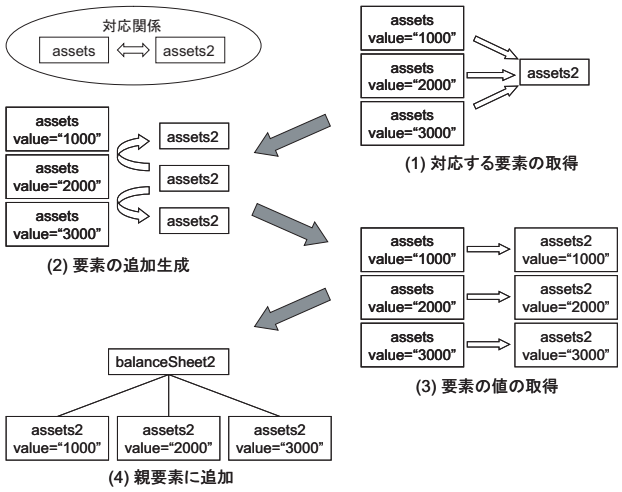

図 4. 子要素の追加生成の例

4. **要素の追加**
値の取得, あるいは子要素の値からの計算によって得られた値を要素に追加する. また, 属性の値は変換先のタクソノミでの要素定義でデフォルト値が設定されている. この属性値に付いても対応付けを定義すれば値の変更が可能である.

5. **属性の処理**
要素と同様の処理を行い, 属性値の取得, 設定を行う.

5. XBRL 財務報告書自動変換ツールの試作

前節のアルゴリズムを用いて, XBRL 文書の項目間の対応関係を利用して既存の財務報告書から別の財務報告書への自動変換を行うツールの試作を行った. 本節では, 試作した財務報告書自動変換ツールについて述べる.

5.1. ツールの概要

本ツールは変換元と変換先のタクソノミ文書, 変換元のインスタンス文書, 対応定義文書を基に, XBRL 文書を変換する. 本ツールの構成を図 5 に示す.

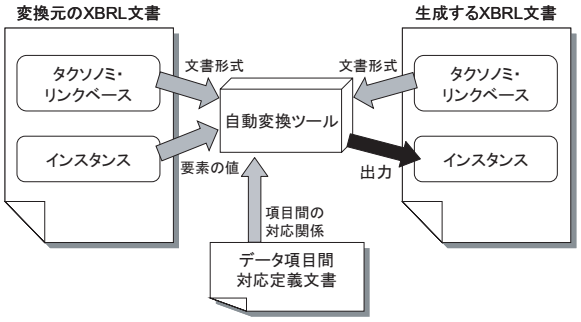

図 5. ツールの構成

本ツールの動作には変換元と変換先のタクソノミ文書, 変換元のインスタンス文書, 対応定義文書が必要である. タクソノミのパスはインスタンス文書から, リンクベースのパスはタクソノミから得られるので, 以下の 3 ファイルをツールの入力として与える.

- 変換元のインスタンス文書
- 変換先のタクソノミ
- 対応定義文書

対応定義文書は整形式 XML 文書でなければならない. 対応定義文書として整形式でない XML 文書が与えられた場合, 処理を中断して終了する. また, 対応定義文書中に誤りがあった場合, 誤りの含まれない対応のみ処理を行う.

5.2. 適用実験

本手法の有効性を確かめるために, 試作したツールを用いた実験を行った. まず, 表 2 の内容を持つ XBRL 文書と, 変換先の XBRL 文書のタクソノミ文書を用意し, 図 6 の対応関係を定義した. そして, 本ツールを用いて変換を行った結果, 表 3 の内容を持つ XBRL インスタンス文書を得た.

得られた XBRL インスタンス文書では, 対応関係に条件指定のない要素 (currentAssets と currentAssets2, liabilities と liabilities2 など) には変換元の文書の値をそのまま記述された. 条件判定および値の加工処理を行う noncurrentAssets は, 正しく処理され, 子要素からの値の計算処理を行う assets, liabilitiesAndStockholdersEquity, StockholdersEquity はそれぞれ正しく計算された.

以上の実験結果から, 本手法を実装したツールを用いることで正確で効率の良い XBRL 文書の変換を行うことができたといえる.

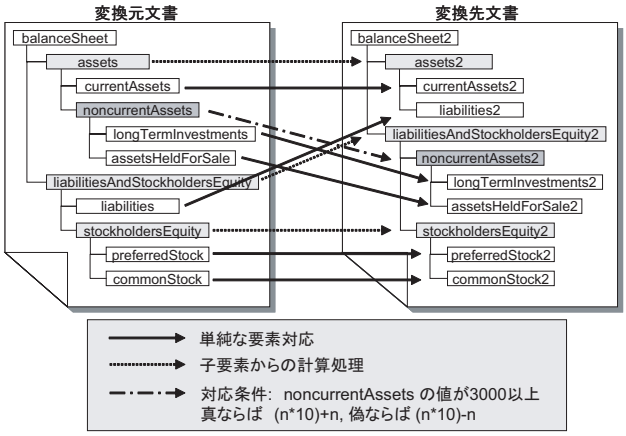

図 6. 項目間の対応

表 2. 変換元文書

balanceSheet	
assets	8000
currentAssets	1000
noncurrentAssets	5000
longTermInvestments	2000
assetsHeldForSale	3000
noncurrentAssets	2000
longTermInvestments	500
assetsHeldForSale	1500
liabilitiesAndStockholdersEquity	8000
liabilities	2000
stockholdersEquity	6000
preferredStock	3500
commonStock	2500

表 3. 変換先文書

balanceSheet2	
assets2	3000
currentAssets2	1000
liabilities2	2000
liabilitiesAndStockholdersEquity2	79000
noncurrentAssets2	55000
longTermInvestments2	2000
assetsHeldForSale2	3000
noncurrentAssets2	18000
longTermInvestments2	500
assetsHeldForSale2	1500
stockholdersEquity2	6000
preferredStock2	3500
commonStock2	2500

6. 考察

本手法を用いることでこれまで手作業で行われる部分の多かった XBRL 文書間の変換が自動的に行えることが示された. さらに, 本ツールは項目数 2000 程度の XBRL 文書間の変換は 2 分程度で完了することができ, 手作業で変換する場合に比べて効率が大幅に向上した. また, 値の計算などを自動的に行うことにより計算ミスや編集ミスがなくなり, 文書の信頼性を向上させることができた.

しかし, 本手法には, 変換元のインスタンス文書に記述されていない項目で値の計算や加工でも取得できない項目については, 変換先のインスタンス文書に記述することはできないという問題点がある. そのような未変換部分の問題の解決手段としては, 変換元の XBRL 文書にすべての財務活動のデータを記述してしまい, その文書から必要な XBRL 文書へと変換する方式が考えられる. あるいは, 未変換部分についてのリストを作成し, ユーザに通知するなどの機能をもたせることで修正作業の負担を軽減することができる.

また, 対応定義の記述方法には, 対応付けに設定する条件に細かい指定ができないことや, 値の加工で可能な処理が簡単なものに限られることなどの改善すべき点がある. 今回は XML 文書として記述した対応定義文書を読み込み変換を行うツールとして実装した. しかし, 細かく条件を指定し, 複雑な計算処理を行う際は, 要素間の対応を定義することができるプログラミング言語を設計し, インタプリタにより変換処理を行う方が有利である. インタプリタ変換方式の方が, 関数ライブラリの提供も容易に行うことができるため, 利用者にとっても使いやすいツールとなるであろう. XBRL 文書変換のためのプログラミング言語の設計と, インタプリタの実装は今後の課題とする.

本研究では XBRL を対象として文書変換手法の提案を行っている. しかし, XBRL は XML に基づいて定義されており, 本手法で用いている技術もまた XML の技術を応用したものであるため, 本手法は一般の XML インスタンスに対する変換手法として応用することが可能である. 具体的には, 既存の XML Schema とそのインスタンスから, 異なる XML Schema に適合するインスタンスを生成することが可能である.

7. まとめ

本研究では, XBRL 文書の自動変換を行うために, 項目間の対応関係の定義と, 変換アルゴリズムの提案を行った. また, 提案アルゴリズムを実装し, 財務報告書自動変換ツールの試作を行った. 自動変換により文書作成を容易に行うことができ, 財務報告書変換時の効率と信頼性の向上を確かめた.

今後の課題としては, XBRL 文書変換のためのプログラミング言語の設計と, インタプリタの実装を行い, 変換時により詳細な条件を指定できるようにすることや, 関数ライブラリを提供し, 値の加工でより複雑な処理を行えるようにすることが挙げられる.

参考文献

[1] ChuoAoyama
Audit Corp., e-Business TrendWatch XBRL 最新動向, http://www.chuoaoyama.or.jp/ebusiness/trend/

[2] XBRL International, XBRL, http://www.xbrl.org/

[3] XBRL International, XBRL Specifications, http://www.xbrl.org/resourcecenter/specifications.asp

[4] XBRL Japan マーケット・アンド・コミュニケーション (”マーコム ”) 委員会, “ XBRL FACT BOOK ”, XBRL Japan, 2002.

[5] 三分一信行,“ XBRL Specification 2.0 の概要 ”, 2001.

[6] チェルシー・ヴァレンタイン, ルシンダ・ダイクス, エド・ティテル, “ XML スキーマ詳解 ”, コンピュータ・エージ社, 2002.

[7] 中山幹敏, 奥井康弘, “ 改訂版 標準 XML 完全解説 ” 上下巻, 技術評論社, 2001.

[8] 野村総合研究所, 普及に向けて動きはじめた XBRL, http://www.nri.co.jp/report/it_solution/2002/n12_5.php

[9] 湯浦克彦, 竹内成明, 佐々木達也,“ xml.trend XBRL ”, 2002.

[10] ルーサー・ハムプトン, デヴィッド・ヴァンカノン, “ 拡張可能なビジネス報告言語 (XBRL) 仕様書 ”, 2001.