社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2005－DD－48　(4)
2005／1／28

XSLT スクリプトの対話的な生成

松本 大貴[†]　　塚本 享治[†]
東京工科大学[†]

XML の形式変換を行う技術として XSL 変換（XSLT）があるが、a) 変更元のソースを読み、操作したい情報を探さなければならない、b) テンプレート・ルールを使用するたびに、頭の中で動作を予想しなければならない、といった問題がある。これらの問題を解決するために、1) XPath 式記入によるノード選択、2) 文字列選択による XPath 式生成 3) XSL 雛形の自動生成 を考えた。
作成者が XPath 式を指定すると、システムは自動的に該当するノード集合の表示を行う(1)。表示方法として、Web ブラウザなどのように文書専用の表示を行う専用（ブラウザ）ビューや文書ソースの表示を行うソース・ビューなどがある。これらの表示画面において、マウスを用いて文字列（情報自体）を選択すれば、XPath 式（情報の位置）を生成できる(2)。このようにしてシステムに保存したノード集合や XPath 式をもとに、XSLT スクリプトの雛形を生成する(3)。
本方式を HTML 文書からの情報抽出を行う XSLT スクリプト作成に適応し、有効性を確認した。

Interactive generation of XSLT scripts

[†]Taiki Matsumoto and [†]Michiharu Tsukamoto
[†]Tokyo University of Technology

Extensible Stylesheet Language Transformation (XSLT) is a language for transforming the structure and content of XML documents. Descibing the stylesheets, we are confronted with identifying data of interest from the bulk XML documents and with constructing the template rules. In order to overcome these problems, we have developed an interactive script generator.
The users give indications to the generator using the following two ways;
1) Specified an XPath expression for the XML document by a keyboard, the system extracts and shows the corresponding portion of the document on the display.
2) Indicated a string on the display by a pointing device, the system constructs the XPath expression to extract the string.
By means of these interactive methods, the user continues to decide XPath expressions step by step. After all expressions are discovered, the final step is;
3) The system gathers the all discovered XPath expressions and generates the integrated XSLT scripting program..

１．はじめに

Web 上での文書交換において、XML を用いることは標準的に行われるようになった。XML はメタ言語であり、文書交換の効率性を高めるためにはスキーマの標準化を行う必要がある。しかし、標準フォーマットは仕様が莫大であり、改定にも時間がかかる。他の形式への変換が容易ならば、用途に合わせた簡潔な形式で文書を作成できる。形式変換を行う技術として XSL 変換（XSLT）[1]があるが、次のような問題がある。

(1) 変更元のソースを読み、操作したい情報を探す必要がある
(2) テンプレート・ルールを使用するたびに、頭の中で動作を予想する必要がある

例えば、HTML 文書から情報を抽出する XSLT スクリプトを作成する場合、欲しい情報は Web ブラウザ上で簡単に識別できるが、XSLT では HTML ソースを読んで情報の位置を特定しなければならない。また、XSLT ではテンプレート・ルールを頻繁に使用するが、その度に途中結果を出力して確認することになる。

本報告では、情報自体（文字列）から情報の位置を調べること、XSLT の途中経過を表示することに着目し、これらの問題を解決した。

第 2 節では XSLT に必要な処理を明らかにした上で、XSLT スクリプト作成の支援策を分類する。第 3 節ではその実現方法について、第 4 節では開発したツールの動作について述べる。第 5 節で本方式の考察を行い、第 6 節で結論を述べる。

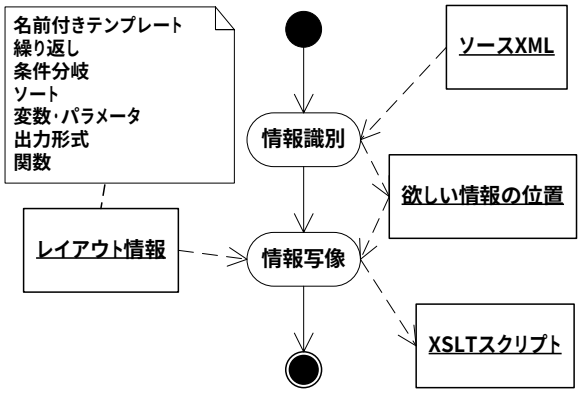

図 1　XSLT スクリプト作成フロー図

２．XSLT を用いた文書変換

図 1 は一般的な XSLT スクリプト作成のフロー図である。変換元の XML 文書を読み、必要な情報の位置を調べる（情報識別）。その後、変換後の形式を考慮しながら、識別した情報を再構成し、XSLT スクリプトを作成する（情報写像）。情報識別は XML 文書から必要な情報を選ぶ行為であり、それに対し情報写像では、結果となる文書のために情報を追加する。

２．１．情報識別

XSLT では情報の位置を XPath 式[2]で表現する。パターン（XPath 式）を用いてノード集合を指定し、その中でさらにパターンからの相対的な位置を XPath 式で識別する。このように XPath 式でノード集合を選択し、テンプレート・ルールを作成していく。

テンプレート・ルールは階層的に用いられるため、変換後の文書構造を把握することは困難である。テンプレート・ルールを作成するたびに途中経過を出力し、確認することがしばしば行われる。途中経過の確認作業を支援する機能が求められる（XPath 式記入によるノード選択）。

また、パターンを指定するためには、欲しい情報の位置を特定できている必要がある。欲しい情報自体（文字列）からその位置（XPath 式）を検索できれば、XSLT スクリプト作成者の負担が軽減される（文字列選択による XPath 式生成）。例えば HTML 文書を形式変換する場合、Web ブラウザ上で文字列をマウスで選択すれば、情報の位置を表す XPath 式を求めることができる。

２．２．情報写像

情報写像では、識別された情報をもとに、変換後の形式を考慮しながら情報を再構成していく。XSLT スクリプトはプレーン・テキストであるため、加工が容易である。前項の情報識別にて求められた情報を XSLT スクリプトの雛形として出力すれば、後からテキスト・エディタなどで容易に修正できる（XSL 雛形の自動生成）。また、XML 宣言や名前空間宣言などの決まりきった記述の自動化も行える。

３．システム概要

前節では、XSLT を用いた文書変換の支援する方法を 3 つに分類した。

(1)XPath 式に対応するテンプレート・ルールの表示を行う（XPath 式記入によるノード選択）

(2)XPath 式や文字列の一部から情報の位置を求める（文字列選択による XPath 式生成）

(3)XSLT スクリプトの雛形を生成する（XSL 雛形の自動生成）

本節ではこれらの実現方法について述べる。

３．１．XPath 式記入によるノード選択

この項では、XPath 式で指定されたノード集合をシステム上に保存し、利用者に解りやすく表示する方法について述べる。

図 2 は「XPath 式記入によるノード選択」のフロー図である。利用者がパターン（XPath 式）を指定すると、システムは自動的に該当する途中経過の表示を行う。

途中経過の表示方法は、全てに共通する表示形式として、文書のソースを表示するソース・ビュー、文書のツリー構造を表示するツリー・ビュー、テンプレート・ルールが持つ情報を XPath 式と値のリストであらわすリスト・ビューの 3 つを採用した。また、文書形式によっては、専用の表示を持つものもある。これを専用ビューと呼ぶことにする。本報告では Web からの情報抽出を題材としており、HTML 文書に特化したブラウザ・ビューを採用した。

システムが持つノード集合において、ルート要素がひとつとは限らない。また、専用ビュー（ブラウザ・ビュー）では、変換元の文書のスキーマ構文にしたがう必要がある。したがって、下図のように、「表示用へ整形」にて、専用ビューで表示可能な文書へとノード集合を整形する。ここでは JTidy[3]を用いてブラウザ・ビューのための整形を行った。

各テンプレート・ルールを表示するためには、元になるノード集合（図 2、ベースとするノード集合）、XPath 式により指定されたノード集合（図 2、ノード集合）、専用のビューのため文書（図 2、表示用文書）、の 3 種類の情報をシステムが持つ必要がある。ソース・ビュー、ツリー・ビュー、リスト・ビューはノード集合から途中経過の表示を行えるが、専用ビューはスキーマに従う必要があるため、表示用文書へと整形する必要がある。

システム上でのノード集合の扱い方については、次項とも関連するため、「システム上のノード集合構造」として後で述べることにする。

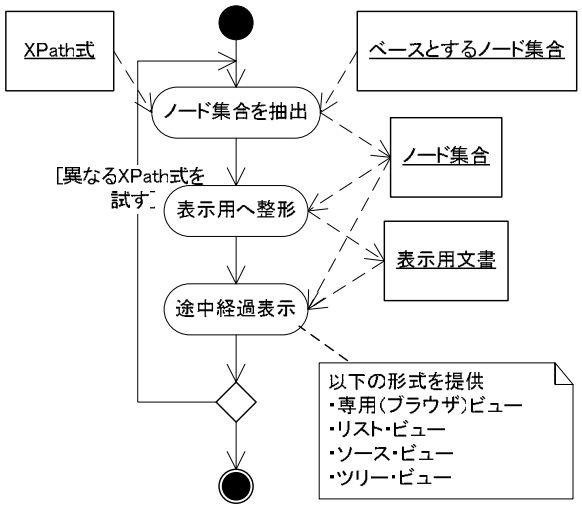

図 2　「XPath 式記入によるノード選択」フロー図

３．２．文字列選択による XPath 式生成

この項では、目的は前項、「XPath 式記入によるノード選択」によって制限されたノード集合から、必要な情報を直接選択する方法について述べる。「直接」とは、マウスを用いたドラッグ＆ドロップによる選択を想定している。これを実現するためには、欲しい情報自身（文字列）から情報の位置（XPath 式）を求める必要がある。一般的な XML 文書の検索においては、XPath 式からそれに対応する文字列を求めるが、ここではその逆、文字列から XPath 式を求める必要がある。XML 文書検索を高速化するために、XPath 式とその値（文字列）のリストを予め作成しておくことがしばしば行われる。このリストがあれば、XPath 式から文字列を求めることも、文字列から XPath 式を求めることもできる。本方式でも同様のインデクシング方式を採用する。また、インデクシングは、前項の「XPath 式記入によるノード選択」でのリスト・ビューは、このインデクシング結果と同一である。

図 3 は「文字列選択による XPath 式生成」のフロー図である。インデクシング結果と利用者が選択した任意の XPath 式または文字列のキーワードとを総当りで照合し、結果をシ

ステムに保存する。一連の作業を繰り返すことで、欲しい情報の位置を識別していく。

システム上のノード集合の扱い方については、前節の「XPath 式記入によるノード選択」と関連しているため、詳細は次項「システム上のノード集合構造」で述べる。

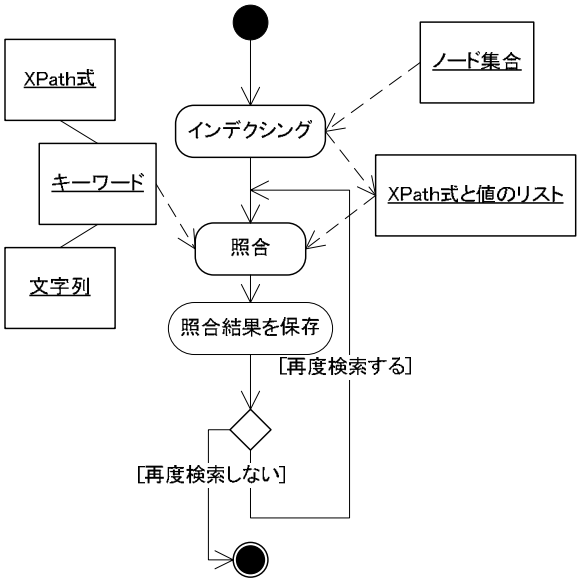

図 3　「文字列選択による XPath 式生成」フロー図

3．3．XSL 雛形の自動生成

「XPath 式記入によるノード選択（3．1）」と「文字列選択による XPath 式生成（3．2）」によって作成されたノード集合をもとに、XSLT スクリプトの雛形を生成する。その後、テキスト・エディタなど雛形を加工し、XSLT スクリプトを完成させる。各テンプレート・ルール内で選択された XPath 式情報は、ノードを表示するための一般的な要素である、「xsl:value-of」として生成する。また、XML 宣言や名前空間宣言などの決まり切った記述も自動的に行う。リスト 1 は自動生成された XSLT スクリプトのサンプルである。

3．4．システム上のノード集合構造

この項では、ここまで述べてきた「XPath 式記入によるノード選択」、「文字列選択による XPath 式生成」「XSL 雛形の自動生成」で共通して利用される、システムにおけるノード集合構造について述べる。

テンプレート・ルールでは、ベースとなるノード集合に対してパターン（XPath 式）を用いてノード集合を指定する。図 4 のノード集合クラス図のようにシステム上で扱うことにした。テンプレート・ルールを表すテンプレート・ルール・クラスでは、ベースとなるノード集合と XPath 式を保存する必要がある。テンプレート・ルール間の関係を表すクラスを別に用意する方法も考えられる。しかし、パターン（XPath 式）は頻繁に変更され、複数のクラスに影響が及ぶことになる。テンプレート・ルール・クラス自体が XPath 式情報とベースとなるノード集合の情報を属性として持つことにした。また、利用者にテンプレート・ルール・クラスの状態を表示するために、パターン（XPath 式）適応後のノード集合と、表示のための文書もテンプレート・ルール・クラスが属性に持つことにした。テンプレート・ルールは再帰構造であるため、テンプレート・ルール・クラスも再帰構造となる。

「文字列選択による XPath 式生成」のインデクシング結果は、リスト・クラスとして扱う。また、リスト・クラスとの照合において多重選択をしないようにするために、識別済みリスト・クラスも必要になる。

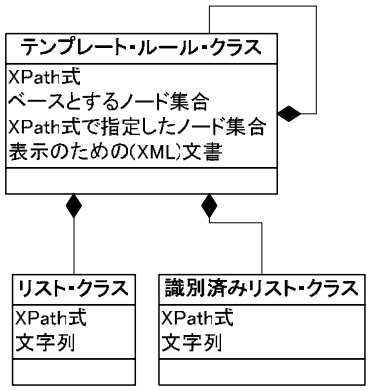

図 4　システム上のノード集合構造クラス図

4．システムの動作

この節では、Web 情報抽出のために実装した XSLT スクリプト作成ツールについて述べる。まず、一般的な Web ブラウザと同じように HTML 文書を表示する（図 5）。この例では Asahi.com[4]の記事ページを表示している。XPath 式：//table[tr/td/span/@class='aHeadlineText'] を指定すると、図 6 のように記事が書

かれた table 要素に制限されて表示が行われる（XPath 式記入によるノード選択）。表示には、ブラウザ・ビュー（図 6）、リスト・ビュー（図 7）、ソース・ビュー（図 8）、ツリー・ビューを用意している。ソース・ビュー（図 8）では table 要素がルート要素となっているように、XPath 式によって指定されたノード集合は必ずしも正しい HTML 文書ではない。ブラウザ・ビューでは、html 要素や body 要素を自動的に補って正しい HTML 文書へと整形してから表示を行う。各表示から欲しい情報を画面下部にドラッグ&ドロップすれば、文書中の情報の位置を識別できる（文字列選択 XPath 式生成）。このように情報を選択していき、最終的にはリスト 1 のような XSLT スクリプトが得られる（XSL 雛形の自動生成）。

リスト 1 の 1-5 行目と 23 行目は、XML 宣言やネームスペース宣言などの決まり切った記述であり、自動的に生成される。7-12 行目は最上位のテンプレート・ルールであり、図 5 の画面にて選択された情報が記述されている。14-21 行目はさきほどのテンプレート・ルールを親に持つテンプレート・ルールである。図 6-8 の画面から選択された情報が記述されている。8-10 行目と 15-16 行目は同一の情報（２月打ち上げ予定の...）を示している。それぞれは、記述されているテンプレート・ルールから相対的に指定される。17-18 行目は記事本文の一行目（来年２月に打ち上げ...）を示している。19-20 行目は記事本文の 2 行目を示している。

図 5　作成支援ツール

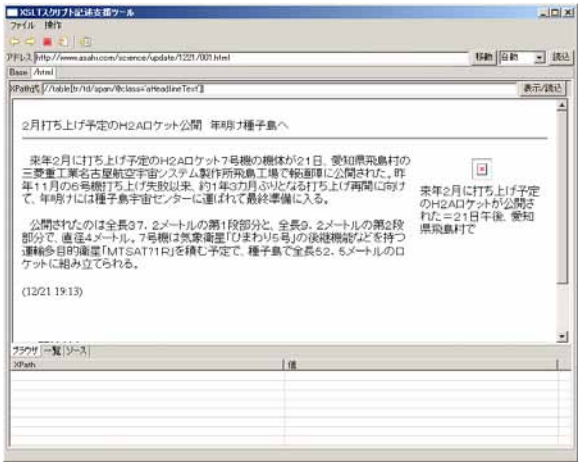

図 6　作成支援ツール – ブラウザ・ビュー

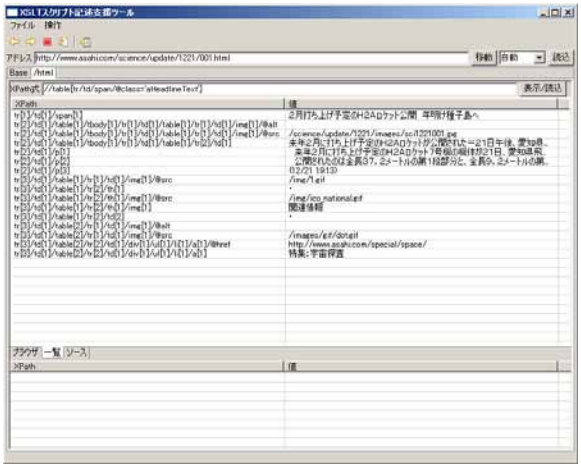

図 7　作成支援ツール - リスト・ビュー

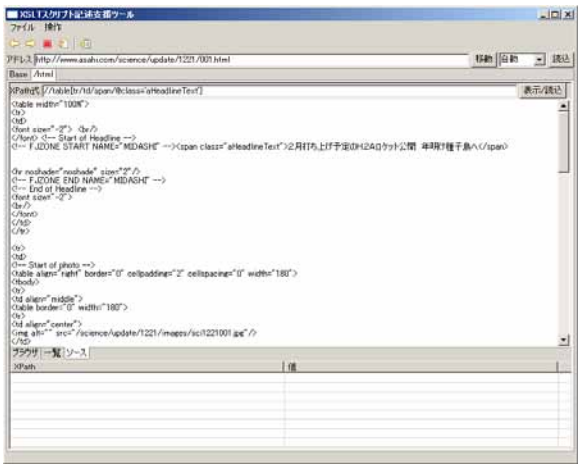

図 8　作成支援ツール - ソース・ビュー

リスト 1　生成される XSLT スクリプトの雛形

```
 1  <?xml version="1.0" encoding="Shift_JIS"?>
 2  <xsl:stylesheet version="1.0"
 3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 4  <xsl:output method="xml" encoding="Shift_JIS" indent="yes"/>
 5  <xsl:strip-space elements="*"/>
 6
 7  <xsl:template match="/html">
 8    <!-- ２月打ち上げ予定のＨ２Ａロケット公開　年明け種子島へ -->
 9    <xsl:value-of select="body[1]/table[1]/tr[3]/td[1]/table[6]/tr[1]/td[2]/
10      table[1]/tr[1]/td[1]/span[1]"/>
11    <xsl:apply-templates select="//table[tr/td/span/@class='aHeadlineText']" />
12  </xsl:template>
13
14  <xsl:template match="//table[tr/td/span/@class='aHeadlineText']">
15    <!-- ２月打ち上げ予定のＨ２Ａロケット公開　年明け種子島へ -->
16    <xsl:value-of select="tr[1]/td[1]/span[1]"/>
17    <!-- 来年２月に打ち上げ予定のＨ２Ａロケット７号機の機体が２１日 … -->
18  <xsl:value-of select="tr[2]/td[1]/p[1]"/>
19    <!-- 公開されたのは全長３７．２メートルの第１段部分と … -->
20    <xsl:value-of select="tr[2]/td[1]/p[2]"/>
21  </xsl:template>
22
23  </xsl:stylesheet>
```

５．考察

本方式の適応範囲、XSLT スクリプト作成の労力、本方式の適した分野、残された課題について述べる。

５．１．本方式の適応範囲

一般的な XSLT の記述形式は以下のように分類される。

・情報の位置の識別（XPath 式）
・テンプレート・ルール
・名前付きテンプレート
・繰り返し
・条件分岐
・ソート
・変数・パラメータ
・出力形式
・関数

このうち、全ての操作に関係し最も重要な処理は、「情報の位置の識別（XPath 式）」と「テンプレート・ルール」である。本方式はこれらに対応している。残りの処理は頻出の処理ではないこと、必要になったとしても再帰的に構造を展開しながら記述をする必要がないことから、本方式は XSLT に求められる重要な要素をカバーしているといえる。

５．２．XSLT スクリプト作成負担の軽減

リスト 2 は Asahi.com から情報を抽出するための XSLT スクリプトである。958 文字中 590 文字、約 61.6%が自動で出力された。また行にすると、全 32 行中（空行除く）、1-5、7、13、15、17、34、36 行目の計 11 行はそのまま使用でき、10、12、20、28-29 行目の計 5 行が少々手を加え、残り 16 行（8-9、11、14、18-19、21-22、24-27、30-33 行目）は新たに記述する必要があった。

自動出力に手を加えた内容は、属性 select に「normalize-space」や「last()」を付け加えたり、属性 xsl:for-each の中に情報を配置したことで属性 select を編集したりしたことである。新たに記述した内容は、形式変換後の文書の要素を書き加えたことや、条件分岐や繰り返し処理を記述するなどしたことである。

５．３．本方式の適した分野

本方式は情報の識別を特に支援する。そのため、第 4 節システムの動作で例に挙げたように、HTML のような複雑な文書からの情報抽出を簡単に行える。ここで扱った変換元の HTML 文書は、893 行、24683 文字あり、この中から欲しい情報を探していくことは大変な

労力がかかる。また、Web 上の有益な情報の大半は、データベース・システムから自動的に生成される。Web 情報抽出分野においても、抽出ラッパー（XSLT スクリプト）を容易に作成できる本方式が有効といえる。

５．４．残された課題

情報の識別において、「直感的な選択」で求められる XPath 式は自動的に求めている。第４節の例（リスト 1 の 16-19 行目）のように、ニュース記事が p 要素で一行ずつ分けて記述されている場合などには、p 要素全てを指定したいことがある。ノード集合を後から手で編集できるようにし、編集結果を表示させる機能を考える必要がある。これは前項での手を加えなければならなかった処理の改善にもなる。

６．おわりに

XSLT スクリプト作成時に頻出する処理、情報自体（文字列）からの位置の検索と、XSLT の途中経過表示を機械的に行えるようになった。本方式により、対話的な操作を実現し、XSLT スクリプト作成を強力に支援できるようになった。

参考文献

[1] XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/1999/REC-xslt-19991116
[2] XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/1999/REC-xpath-19991116
[3] JTidy. http://sourceforge.net/projects/jtidy
[4] Asahi.com. http://www.asahi.com

リスト 2　Asahi.com から情報を抽出するための XSLT スクリプト

```
 1  <?xml version="1.0" encoding="Shift_JIS"?>
 2  <xsl:stylesheet
 3    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 4  <xsl:output method="xml" encoding="Shift_JIS" indent="yes"/>
 5  <xsl:strip-space elements="*"/>
 6
 7  <xsl:template match="/html">
 8  <news>
 9    <headline>
10      <xsl:value-of select="normalize-space(//span[@class='aHeadlineText'])"/>
11    </headline>
12    <category><xsl:value-of select="//td[a='home']/a[last()]" /></category>
13    <xsl:apply-templates select="//table[tr/td/span/@class='aHeadlineText']" />
14  </news>
15  </xsl:template>
16
17  <xsl:template match="//table[tr/td/span/@class='aHeadlineText']">
18    <content>
19      <xsl:for-each select="tr/td/p">
20        <xsl:value-of select="normalize-space(.)" />
21      </xsl:for-each>
22    </content>
23
24    <xsl:if test="tr/td/table/tr/td/div/ul/li">
25    <relatedlinks>
26      <xsl:for-each select="tr/td/table/tr/td/div/ul/li">
27        <item>
28          <headline><xsl:value-of select="normalize-space(a)" /></headline>
29          <url><xsl:value-of select="a/@href" /></url>
30        </item>
31      </xsl:for-each>
32    </relatedlinks>
33    </xsl:if>
34  </xsl:template>
35
36  </xsl:stylesheet>
```