人工知能学会研究会資料
SIG-SWO-A501-03

なんでもRSS! – HTML文書からのRSS Feed自動生成

南野 朋之† 奥村 学‡

概 要

本稿では，HTML 文書中に含まれる時系列情報を自動的に発見し，RSS Feed を自動的に生成する手法を提案し，提案手法に基づいたアプリケーション “なんでも RSS!” について紹介する．このシステムを用いることで，RSS Feed を配信しない時系列を記述する Web ページを RSS Reader で閲覧することが可能になる．また，そのような Web ページの著者は，容易に “RSS-ready” な Web ページを作成することが可能になる．また本稿では，システム公開後の利用状況についても報告する．

Automatic Generation of RSS Feed based on HTML Document Structure Analysis

Tomoyuki NANNO† Manabu OKUMURA‡

Abstract

In this paper, we present a system to automatically generate RSS Feeds from HTML documents which include time-series information with date expressions (e.g., archives of weblogs, BBSs, chats, and mailing lists, update descriptions on a site page, announcements of events, and so on). Our system is based on extraction of date expressions, structure analysis of HTML documents, and title detection/generation from the contents.

1 はじめに

Web 上の情報は，人が目で見て理解する情報であったが，近年，計算機で直接扱うことができるメタデータを付加しようという動きが活発になってきている．ここ数年注目されているのが RSS[1, 2, 3] (RDF Site Summary / Really Simple Syndication) であり，現在爆発的に広がっている Weblog では，ほぼ標準で RSS によるメタデータ配信が行われるなど，広く利用されるようになった．RSS は，サイトの更新情報 (見出し，要約，本文，更新日時など) を配信するのに適したフォーマットであるため，多くの新聞社の Web サイトやニュースサイトなどでも利用されており，今後も RSS を配信するサイトの数は増え続けると考えられる．

RSS Feed には，サイトの更新に関する情報が共通の書式で含まれているため，例えば，RSS リーダーと呼ばれるアプリケーションを利用することで，容易にサイトの更新情報を閲覧することが可能になり，さらには複数の RSS Feed の情報をまとめて閲覧，利用することなどが容易に実現できる．また，サイトの更新情報を配信する以外にも，メタデータ公開のためのコンテナとしての可能性も秘めており，現在様々な形で利用されている [4].

しかしながら，RSS によりメタデータを配信しているサイトは，CMS(Content Management System) を利用しているような一部のサイトに限られているのが現状である．なぜなら，CMS を使えば，ユーザは HTML 文書を生成すると同時に，自動的に RSS を生成することができるが，人手で作成している Web ページなどでは，RSS Feed も人手で作成し，メンテナンスしなければならず，これにはコストがかかるためである．また

†東京工業大学大学院総合理工学研究科
Interdisciplinary Graduate School of Science and Engineering, Tokyo Institute of Technology
〒 226-8503 横浜市緑区長津田町 4259 R2-728
nanno@lr.pi.titech.ac.jp
‡東京工業大学精密工学研究所
Precision and Intelligence Laboratory, Tokyo Institute of Technology
〒 226-8503 横浜市緑区長津田町 4259 R2-720
oku@pi.titech.ac.jp

別の問題として，RSS Feed は一般的に，直近 n 回の更新に対するメタデータであるため，過去の情報を取得できないという問題がある [5].

そこで本論文では，HTML 文書中に含まれる時系列情報を自動的に発見し，RSS Feed を自動的に生成する手法を提案し，本手法に基づいたアプリケーション “なんでも RSS!” について紹介する．このシステムを用いることで，RSS Feed を配信しない時系列を記述する Web ページを RSS Reader で閲覧することが可能になる．また，そのような Web ページの著者は，容易に “RSS-Autodiscovery[6]” 対応の Web ページを作成することが可能になる．

2 RSS Feed 自動生成システムの概要

本節では，日付表現を伴って時系列を記述する Web ページから，RSS を生成する手法について概要を述べる．

RSS Feed は，channel 要素と item 要素の二種類の情報から構成される [1, 4]．channel 要素では，Feed のタイトル，URI，概要など，Feed 全体に関する情報, 及び，含んでいる記事の一覧などが記述される．item 要素では，channel 要素で一覧を示した各記事について，記事のタイトル，URI，概要などが記述される．

よって，Web ページ中に含まれる時系列情報から RSS Feed を自動生成するためには，適切な item 要素 (言い換えれば，“記事の範囲”) を抽出しなければならない．しかしながら，Web ページ中の時系列情報は，様々なスタイルで記述される．そこで，本研究で対象とする時系列情報は，明示的な日付表現を含む Web ページに限定する．これは，提案手法が繰り返し出現する日付表現を利用した構造解析手法に基づくためである．

図 1 に時系列情報を記述する際の典型例を示す．図 1(左) に示す日付ベースの時系列は，各記事に一つずつ日付表現が含まれる構造である．BBS や Web 日記，サイトの更新情報などを記述する時系列の多くが，このタイプに当てはまる．一方，Weblog やニュースのヘッドラインのページなどに多い，図 1(右) に示す記事ベースの時系列では，一つの日付表現に複数の記事が含まれる構造を示す．よって，このような Web ページから適切な RSS Feed を生成するためには，図 2 に示すような構造を抽出しなければならない．

図 3 に，本論文で提案する RSS Feed 自動生成システムのフローチャートを示す．システムはまず，日付表現を検出し，3.2 節で述べる手法を適用することで，日付ベースの item を抽出する．その後，日付ベースの item

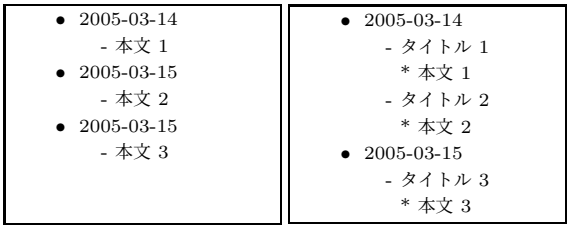

図 1: 日付ベース (左) と記事ベース (右) の時系列

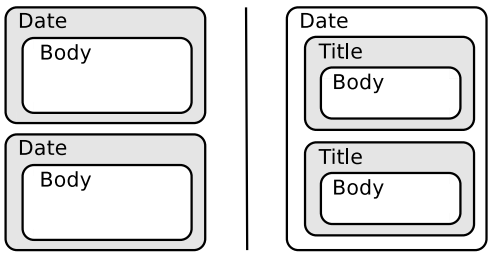

図 2: 抽出すべき構造 (日付ベースの構造 (左) と記事ベースの構造 (右))

がタイトル表現を含むかどうかを調べ，含まない場合は，適切なタイトルを自動生成することにより，日付ベースの RSS Feed を生成する．タイトル表現を含む場合は，さらにタイトル表現に基づいたセグメンテーションを行い，記事ベースの RSS Feed を生成する．

3 HTML 文書からの RSS Feed 自動生成

本節では，HTML 文書からの RSS Feed 自動生成手法について述べる．なお，本手法で対象とする時系列情報は，以下の性質をもつと仮定している．

1. 記事の書かれた日を示す日付表現は記事の上部に

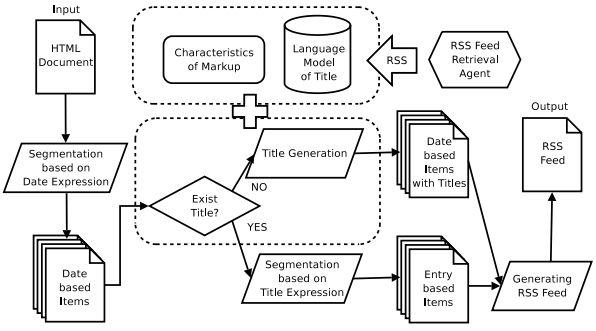

図 3: RSS Feed 自動生成システムのフローチャート

ある[1]

2. 一連の記事に対して，各日付表現に係るタグの種類は一定である
3. 一連の日付表現はフォーマットが一定である
   ex.) “2003/1/2” と “2-Jan-2003” は別のフォーマット

以下では，図 3 に従い，それぞれの処理について詳説する．

3.1 日付表現の抽出

システムはまず，HTML Tidy[7] により，well-formed な XML 文書に変換した HTML 文書から，日付表現を抽出する．日付表現の抽出には，人手により作成した正規表現ベースのルールを使用する．最上部に年が記載され，各記事の日付では年を省略し，月日のみを記載するようなケースや，年号の省略，西暦の上二桁の省略に対応するために，ヒューリスティックを用い，不足している情報の補完も行う．(日付表現の検出に関する詳細については，[8] を参照．)

3.2 日付ベースの記事の抽出

次に，システムは検出された日付表現を使用し，Web ページを日付ベースの記事に分割する．

Web ページには，複数の時系列情報が記述される可能性がある．そこで，システムはまず前節で発見された日付表現を “同じタイプの日付表現” 毎にグルーピングする．“同じタイプの日付表現” とは以下の条件をすべて満たす日付表現の集合を指す．

- タグの係り方が同じ
- 日付表現を囲む HTML タグ中において，タグの直後 (もしくは直前) から日付表現の直前 (もしくは直後) までの byte 数が同じ
  ex.) <h1> ☆ 2005.2.2　発表申込締切 </h1>
  <h1> ☆ 2005.4.8　論文投稿のページを追加 </h1>
  以上二つの日付表現は，タグの直後から日付表現の直前までの byte 数が同じ (どちらも 2byte) である．
- 日付表現のフォーマットが同じ
  ex.) “2004/1/1” と “1/2” は同じフォーマット
  “1-Jan-2004” と “2004/1/2” は別フォーマット

これら同じタイプの日付表現は，ブラウザによってレンダリングされた際に同じように表示されるため，これらを一連の日付表現と考え，システムは各タイプに対して，Web ページから日付ベースの記事を抽出することを試みる．

システムはまず，あるタイプに属する日付表現に対し，各日付表現によって修飾される記事の開始位置を，以下の手順で決定する．

1. 同じタイプに属する日付表現に対し，XPath[9] 表現を列挙する
   (同じタイプに属する日付表現は，タグの係り方は同一である)
2. それらの表現を根ノードから順に見ていき，すべての日付表現に対して共通でない最初のタグを記事の開始位置とする

図 4 の例を考える．この文書には，三つの日付表現 (“date 1”，“date 2”，“date 3”) が存在し，それらは同じタイプに属しているとする．これらの日付表現に対する XPath 表現は以下の通りである．

- date 1:　/body[1]/div[1]/h1[1]/date[1]
- date 2:　/body[1]/div[1]/h1[2]/date[1]
- date 3:　/body[1]/div[3]/h1[1]/date[1]

“/body[1]/div[3]” は，HTML 文書中の一つ目の body タグで囲まれた部分に含まれる div タグのうち，前から三番目の div タグで囲まれた範囲を示す．

これら三つの XPath 表現を見ていくと，“div” の直下にある “h1” で初めてすべての日付表現に対して別のタグが修飾している．(“body” は，三つの日付表現で共通．“div” は “date 1” と “date 2” で共通．) これは，深さ 3 の位置にある h1 タグでセグメンテーションを行えば，各セグメントにちょうど一つの日付表現が属することを示している．つまり，この深さ 3 の位置にある h1 タグは記事の開始位置を示すと考えることが出来る．

次にシステムは，記事の終了位置を決定する．システムはまず，以下のルールによって記事の終了位置の候補を抽出する．

(1) 次の記事の開始位置の一つ前の位置
(2) DOM ツリーにおいて，開始位置のタグよりも根ノードに近い HTML タグが表れる一つ前の位置

ただし，最後の記事については，上記 (1) の条件が使用できないため，以下の条件を追加する．

[1] フラットな構造でマークアップされる場合のみ，この性質を仮定する．マークアップにより日付と記事の対応が明らかである場合 (例えば，日付表現と記事を囲うようなタグが存在する場合) は，この限りではない．

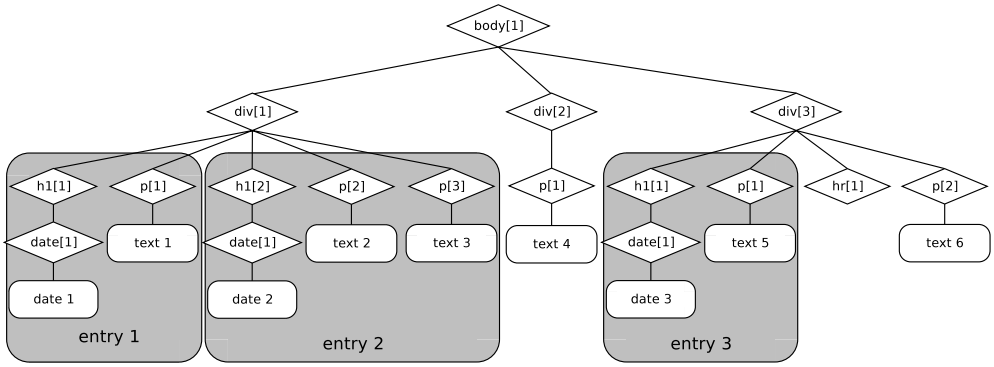

図 4: 日付ベースの記事の抽出例

(3) 開始位置のタグと DOM 中の深さが同じタグのうち，それ以前の entry で一度も出現していないタグが現れる一つ前の位置

次に，これらの候補位置の中からもっとも開始位置に近い終了位置を最終的な終了位置として選択する．

先ほど同様，例として図 4 を考える．図中の “date 1” に対する記事は，“text 1” を含む．なぜなら，この記事の終了位置はルール (1) によって決定されるためである．(二つ目の記事の開始位置は “date 2” を修飾する “h1” タグである．) “date 2” が示す二つ目の記事は，“text 2” と “text 3” を含むが，“text4” は含まない．なぜなら，この記事の終了位置はルール (2) によって決定されるためである．(“text 4” を修飾する “div” タグは開始位置の “h1” タグよりも根ノードに近い．) また，“date 3” が示す最後の記事は，“text 5” を含むが “text 6” は含まない．なぜなら，この記事の終了位置はルール (3) によって決定されるためである．(“hr” タグは，この記事より前に表れるどの記事にも含まれていない．)

以上のような処理により各日付表現が修飾する記事の開始位置と終了位置を決定することで，それぞれの日付表現のタイプに対して日付ベースの記事集合を決定することができる．

3.3 RSS Feed の収集

2 節で述べたように，適切な RSS Feed を生成するためには，タイトルの発見，タイトルの生成が必須のタスクとなる．本研究では，大量に収集した RSS Feed から得た統計情報を利用し，タイトルの発見，タイトルの生成を行う手法を提案する．

収集した RSS Feed は，weblogUpdates.ping[2] サーバ，

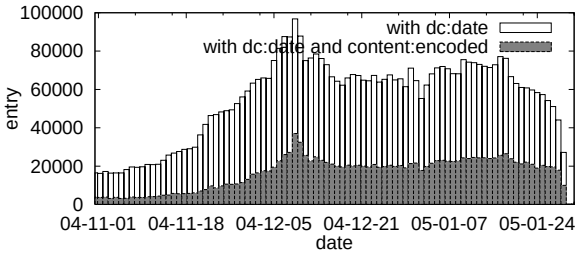

図 5: 収集した記事数の推移

ping.bloggers.jp[3] で公開されている changes.xml の情報を元に収集した，主に Weblog の RSS Feed である．RSS Feed には，記事のタイトルと本文 (の一部) が含まれる．

収集したデータは，一日あたり約 6 万〜7 万記事[4]．記事の日付分布は，図 5 のようになっている[5]．なお，図 5 は，記事の書かれた時間を示す要素 “dc:date [6] ” と，さらに，そのうち本文全体を含む要素 “content:encoded [7] ” を持つ記事数の分布を示している．

以降では，このように収集した RSS Feed から得られる統計情報を利用したタイトルの発見とタイトルの自動生成について述べる．

3.4 タイトルの検出

本節では，タイトルの検出について述べる．なお，この処理を適用する時点までに，システムは Web ページ

[2] http://www.xmlrpc.com/weblogsCom

[3] http://ping.bloggers.jp/

[4] 収集を開始したのが 12 月の初旬だったため，それ以前の日付を持ったデータは非常に少ない．

[5] 12 月 15 日以前の記事数と比較して，それ以降の記事数が減っている理由は，BBS などからの Ping を ping.bloggers.jp が受け付けなくなったため．詳しくは，http://uva.jp/dh/mt/archives/004616.html を参照のこと．

[6] Dublin Core Metadata Element Set, Version 1.1: Reference Description http://dublincore.org/documents/dces/

[7] RDF Site Summary 1.0 Modules: Content http://web.resource.org/rss/1.0/modules/content/

中の時系列情報を発見し，日付ベースの構造を抽出しているとする．また，以下の二つの仮定をおく．

1. タイトルは本文の前にある
2. すべてのタイトルは単一のタグ[8](属性を含む) でマークアップされている

二つ目の仮定については，多くのケースに当てはまるが，例外も存在する．例えば，タイトルだけをマークアップしているタグが存在せず，タイトルに付いている飾り文字など，タイトル以外の要素を含めてマークアップされているケースがある．しかし，本研究ではこれらがタイトルに混入することは，RSS Feed を作成する目的では，それほど問題にならないと考え，この仮定を導入した．

以上のような仮定を行うと，タイトルを検出する問題は，タイトルをマークアップしているタグを検出するという問題に置き換えて考えることが可能になる．そこでシステムはまず，以下の条件を満たす，タイトルをマークアップしている可能性のある候補タグを選択する．

1. すべての日付単位の記事に最低一度以上出現する
   (タイトルを含むのであれば，すべての記事にタイトルが付いているはず．)
2. そのタグでマークアップされている範囲に，text が存在する
   (タイトルは，text で存在する．)
3. そのタグでマークアップされている範囲に blockquote, center, div, h1..6, p, li, td, hr など改行効果のあるタグが出現しない
   (タイトル中に，これらの要素が含まれるのは不自然．)
4. 日付表現と，そのタグの中で最初に現れるタグ間に出現する text が 200byte 以下.
   (日付と最初のタイトルとの間に，長い text があるのは不適当．)

以上の条件により抽出された候補タグの中から，タイトルをマークアップしていると考えるタグを選択する．その際，HTML タグの特徴と，RSS Feed から収集した約 300 万のタイトルの平均文字長，及び品詞の 3-gram を使用し，以下の手順で選択する．

1, heading タグ (`h1..h6`) である
2, class 名に “title” や “head” などを含む

[8]記事の識別番号が，id 属性に付加されているケースがあったため，id 属性については，属性値が異なっている場合でも単一のタグでマークアップされていると考えることにした．

表 1: タイトルと本文の関係

ケース	数	割合
タイトルがそのまま本文に現れる	256,331	(25.6%)
タイトルの内容語がすべて本文に現れる	725,394	(72.5%)
タイトルの内容語が一つ以上本文に現れる	820,042	(82.0%)
タイトルの内容語が一つも本文に現れない	179,958	(18.0%)

3, タグがマークアップするすべての text について
- 平均文字長が 20 文字以内
- 収集したタイトルの品詞の 3-gram に適合

以上の特徴を満たすタグがあった場合，それらの中で最も前方に出現するタグをタイトルを示すタグとして選択する．ただし，上記 1,2. の性質を満たすタグがあった場合，それを優先する．また，タイトルが無いと判断された場合は，日付ベースの時系列情報と考える．

3.5 日付ベースの RSS Feed の生成

3.4 節で述べたタイトル検出処理により，タイトルが含まれていないと判定された場合，日付ベースの RSS Feed を生成する．

その際，タイトルの生成を行う必要がある．なぜなら，RSS の item 要素のタイトルは，必須項目であるだけでなく，ユーザが本文を読むかどうかの重要な指標となるためである．

タイトルの生成に関しては，本研究では，extraction ベースの手法でタイトルの付加を行う．この理由は，実際にサービスにする際の処理のスピードを考えてということもあるが，RSS Feed より収集した content:encoded を持つ 100 万件のタイトルと本文のペアを調査した結果，表 1 に示すように，多くの場合でタイトルに必要な語が本文に現れるという特徴を持っていたためである．

よって，本研究では，

- 複数の一連の記事に同時にタイトルを生成
- 記事の書かれた時期に特徴的に現れる単語を重要視
- 大量に収集した RSS Feed から得られる統計情報を利用

する，extraction ベースのタイトルの生成を提案する．タイトルの生成は，“タイトル候補の抽出”，“適切なタイトルの選択” の二段階手法で行う．次節以降で，それぞれの段階について詳説する．

3.5.1 タイトル候補の抽出

本文からタイトルとして適切な範囲を選択する．使用する素性は，以下の二つである．

1. 文字長 n のタイトルが選択される確率

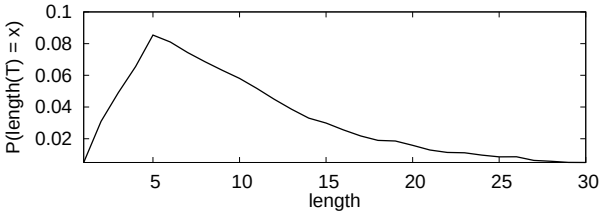

図 6: タイトルの文字長

2. タイトルの先頭，末尾を考慮した品詞 (活用形を含む) の 3-gram に基づく，タイトルの生成確率

これらの確率は，RSS Feed から収集したタイトル 3,281,080 から計算した．図 6 にタイトルの文字長の分布を示す．また，タイトルの品詞 3-gram については，3,177,896 のタイトルから計算を行った．

以上の統計情報を元に，以下の手順で本文中からタイトルの候補を抽出する．

1. 本文を HTML タグで分割し，形態素解析を行う．
2. 名詞，未知語の連続している部分は一つの形態素と考え，品詞は最後の形態素の品詞とする．
3. 形態素解析した文の任意の範囲の形態素列において，以下の条件を満たす形態素列を抽出
   - その形態素中に含まれる，先頭，末尾形態素を考慮したあらゆる 3-gram が，収集したタイトルに出現する
   - 括弧を含む場合は，括弧の対応がとれている

以上の手順によって得られた候補に対して，式 1 によってスコアを計算する．

$$\begin{aligned} score(T) &= log(P(length_{char}(T) = n)) \\ &+ log((\prod_{i=0}^{n+2} P(p_i|p_{i-1}, p_{i-2}))^{\frac{1}{n+3}}) \quad (1) \end{aligned}$$

T は，タイトルの bag-of-words を，右辺の第一項は，文字長 n のタイトルが選択される確率を示す．右辺の第二項で，3-gram の生成確率の相乗平均を計算している理由は，タイトルの長さが長くなるにつれ，この品詞列の生成確率が低下してしまうのを防ぐためである．最終的には，スコアが閾値以上の候補を最終的なタイトル候補として出力する．

3.5.2 適切なタイトルの選択

以上の処理によって抽出された候補の中から，最も適切なタイトルを選択する．

本研究では，以下の三つの要素を考慮し，適切なタイトルを選択する．

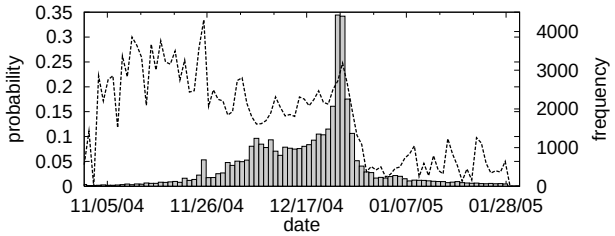

図 7: 「クリスマス」を本文に含む記事数の推移とタイトルへの出現確率の推移

1. $tf(w_i)$: 本文中の形態素 w_i の頻度
2. $idf(w_i)$: 同時にタイトルを付ける複数の本文において，その形態素 w_i を含み，かつタイトルを付けようとしている文書より前に書かれた文書の数の逆数 (タイトルにふさわしい語だったとしても，すべての記事に同じタイトルが付くのは不自然)
3. $P_{Date}(w_i \in T_{Date}|w_i \in D_{Date})$: その単語がある時期に本文に現れた際，それがタイトルに含まれる確率 (例えば，1 月に「クリスマス」という単語が本文に登場しても，タイトルにはなりにくい)

なお，w_i は，タイトル中の i 番目の形態素を示し，T, D は，それぞれタイトル，本文の bag-of-words である．

$P_{Date}(w_i \in T_{Date}|w_i \in D_{Date})$ に関しては，記事の書かれた日付に対して，以下の四つの確率を考える．

- P_{day}: 記事を書いた日付と直前二日間に書かれた記事から計算
- P_{month}: 同じ月に書かれた記事から計算
- P_{year}: 同じ年に書かれた記事から計算
- P_{all}: 収集されたすべての記事から計算

P_{Date} は，P_{day} が定義されていればそれを，定義されていなければ，P_{month} を，さらにそれも定義されていなければ，P_{year} をといったように，順に適用する．なお，これらの確率は，収集している RSS から動的に計算され，1 時間毎に更新される．

図 7 に，2004 年 11 月 1 日から 2004 年 12 月 31 日までの期間において，“クリスマス” を本文に含む記事数の推移と，本文に “クリスマス” が登場したときに，タイトルに “クリスマス” が含まれる確率の推移を示す．図 7 から以下のことが観察できる．

- 12 月中では，“クリスマス” を含む記事数は増えており，12 月 25 日付近で最大である
- クリスマス以降では，記事数は 11 月と同等程度であるにも関わらず，タイトルに含まれる確率は非常に低くなる

- 2005-01-28
  明日はいよいよカザフスタン戦。日本代表の今年の初戦ですね。ドイツワールドカップの最終予選に向けて，良い結果を期待します！
- 2005-01-29
  カザフスタン戦、勝ちました！圧勝でしたね！雨の中，横浜国際総合競技場まで行って良かった！この調子で最終予選も突破して欲しいですね。
- 2005-01-30
  昨日のカザフスタン戦で興奮したせいか、昨日はあんまり眠れませんでした。おかけで大遅刻。

図 8: 記事の例 (1)

表 2: 生成されたタイトル

記事	タイトル	スコア (log)
2005-01-28	カザフスタン戦	-4.09332785321058
2005-01-29	横浜国際総合競技場	-4.48821544478775
2005-01-30	おかけで大遅刻	-5.01575871021318

これらの統計情報を元に，各タイトル候補に対して，式 2 で score を計算し，reranking を行う．

$$score(w_1, ..., w_n) = P_{length}(n) \quad (2)$$
$$* \quad \sum_{i=1}^{n} tf * idf(w_i)^2 * P(w_i \in T_{Date} | w_i \in D_{Date})$$

なお，w_i はタイトル候補中の i 番目の形態素を示す．(ただし，助詞，助動詞，名詞-非自立，動詞-非自立，形容詞-非自立，名詞-代名詞，記号は除いて計算する．)

最終的に score が最も高かったタイトル候補を，その記事に対するタイトルとする．

3.5.3 適用例

図 8 の 3 つの記事について，以上の処理を適用したときの結果を表 2 に示す．この例では，一つめの記事のタイトルとして選ばれた「カザフスタン戦」がすべての記事に現れるが，二つめ，三つ目の記事では，本文に出現したときにタイトルに出現する確率が高いにもかかわらず，タイトルとして選択されていない．

また，図 9 の例において，日付を変化させたときのタイトル変化を表 3 に示す．このように，記事の日付に対して，適切なタイトルが生成されている．

3.6 記事ベースの RSS Feed の生成

3.4 節で述べたタイトル検出処理により，タイトルが含まれていると判定された場合，さらに構造解析を行い，記事ベースの RSS Feed を生成する．

再度，図 2 を考える．図 2 に示す，日付ベースと記事ベースの構造を比較すると，日付ベースの構造における日付表現が，記事ベースの構造におけるタイトル表現と対応していることがわかる．言い換えれば，3.2 節で述べた，日付表現に対して適用した構造解析手法を，日付表現により分割した各記事に含まれるタイトル表現に対して，再び適用することで，記事ベースの構造を抽出することが可能である．

以上のような手続きにより，記事ベースの RSS Feed を生成する．

お正月と言えば、「あけましておめでとう！」。クリスマスと言えば、「メリークリスマス！」。今年の成人式は、1 月 10 日。

図 9: 記事の例 (2)

表 3: 日付を変化させたときのタイトルの変化

日付	タイトル
2004-12-25	クリスマス
2005-01-01	あけましておめでとう
2005-01-10	今年の成人式

4 なんでも RSS!

4.1 概要

本節では，本論文で提案した手法を使用したアプリケーション，“なんでも RSS![9]” について述べる．(図 10)

図 11 は，JSAI2005[10]のスクリーンショットの一部である．このサイトには，“最新情報” セクションには，サイトの変更箇所が日付表現を伴って記述されている．

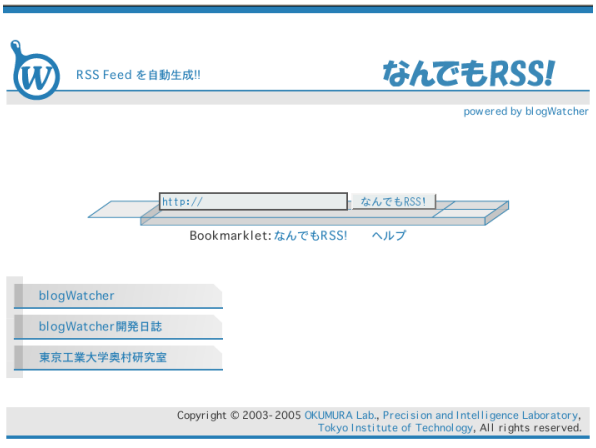

図 10: なんでも RSS!

[9] http://blogwatcher.pi.titech.ac.jp/nandemorss/
[10] http://www.jaist.ac.jp/jsai2005/

図 11: JSAI2005 サイトのスクリーンショット

図 12: なんでも RSS!の出力結果

図 12 は，“なんでも RSS!” が出力した RSS Feed の一つである．

このように，本システムは，Web ページ中のタイトル表現を検出し，それぞれの日付表現が修飾する適切な Web ページの一部を切り出し，RSS Feed を生成する．また，Web ページが複数の時系列を含む場合は，すべての可能性のある Feed を作成し，ユーザはそれらの中から自分の欲しい Feed を選択することが可能である．

このシステムと RSS Reader のようなアプリケーションを組み合わせることで，ユーザは，このサイトの変更を容易に追跡することが可能になる．また，別の利用方法としては，このシステムが出力する RSS Feed の URL を，この Web ページの header に指定することで，追加のメンテナンスコスト無しで，“RSS-Autodiscovery” 対応の Web ページを作成することが可能になる．

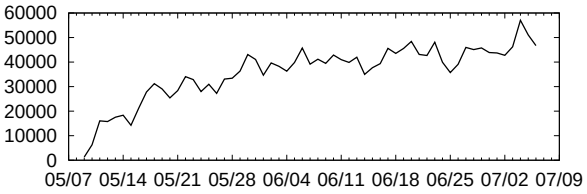

図 13: アクセス数の推移

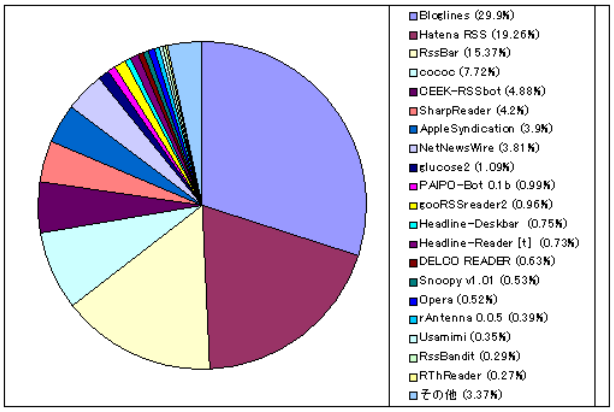

図 14: User-Agent の内訳

4.2 利用状況

図 13 に “なんでも RSS!” が配信する RSS Feed へのアクセス数の推移を示す．2005 年 5 月 9 日の公開以来，約 2ヶ月で累計で 212 万以上のアクセスがあった．(2005 年 7 月 6 日現在) 最近では一日あたり，延べ 4 万 5 千以上，異なりで 6 千以上の RSS Feed を配信している．

表 4 に，“なんでも RSS!” が配信する RSS Feed へのアクセスランキングを示す．ほとんどのサイトが RSS Feed を持っていないサイトであるが，“小心者の杖日記[11]” では，“なんでも RSS!” を利用し，RSS-Autodiscovery Ready なサイトを実現している．また，上位ランキングには含まれないが，2ch[12]など掲示板への適用も多く見られる．

また，図 14 に，“なんでも RSS!” へアクセスする User-Agent のランキングを示す．(一部のブラウザは除く．) これら RSS リーダーによると思われるアクセスが，全体の 9 割以上を占めている．

これらの結果から，RSS Feed を配信しないサイトに対しても，Feed を提供して欲しいというユーザの強いニーズが伺える．OS による RSS の標準サポートが発

[11] http://www.outdex.net/diary/
[12] http://www.2ch.net/

表 4: アクセスランキング (上位 20 位まで)

順位	RSS Feed	タイトル	アクセス数
1	×	[堕] 自堕落記 / Diary “Jidarakuki - My falling days”[13]	16,123
2	×	ITmedia ニュース top[14]	12,617
3	×	IT 総合情報サイト「ITmedia」Home[15]	11,524
4	○	極弐[16]	9,793
5	×	Yahoo!ニュース - トピックス[17]	9,288
6	×	やじうま Watch[18]	9,079
7	×	窓の杜[19]	8,678
8	×	DIARY[20]	7,924
9	×	妖精現実 フェアリアル[21]	7,883
10	×	カトゆー家断絶[22]	7,618
11	○	小心者の杖日記[23]	7,540
12	×	@nifty：デイリーポータル Z[24]	7,309
13	×	黒板ほ[25]	7,194
14	×	パソコン遊戯[26]	6,889
15	×	05.7 K.Moriyama’s diary[27]	6,213
16	×	TECHSIDE[28]	6,009
17	×	バーチャルネットアイドル・ちゆ１２歳[29]	5,659
18	×	hermitage akihabara -the voices- アキバショップの生の声！[30]	5,648
19	×	気まぐれ日記[31]	5,565
20	×	Beltorchicca[32]	5,562

表される など，RSS の普及も加速しているため，今後も RSS Feed が無い場合でも RSS リーダーで更新情報を閲覧したいという要求は増すと考えられる．

4.3 エラー分析

前節で，異なりで 6 千以上の RSS Feed を配信していると述べたが，HTML 文書の構造解析のみが試みられ，RSS Feed が取得されていないケースが，約 1 万 2 千ページ存在する．これらのページには，RSS Feed の自動生成が失敗しているケースが含まれると考えられる．

本節では，これら 1 万 2 千のサイトの内，構造解析が試みられている回数の多い上位 100 サイトについて，実際処理を適用し，エラーが発生しているかどうか，また発生している場合の原因について分析を試みた．なお，100 位のサイトのアクセス回数は，約 2ヶ月間で 28 回である．

表 5: エラー分析の結果

原因	ページ数
正常に処理	39
時系列情報なし	41
日付の認識ミス	12
構造解析ミス	4
フレームページ	2
日付の誤認識	1
パターン発見不可	1
合計	100

表 5 に分析の結果を示す．

100 ページ中 39 ページで，正しく RSS Feed を自動生成出来ていた．本システムでは，対象ページが RSS Feed を提供している場合，自動生成した Feed と共に，サイトオリジナルの RSS Feed を提示している．正しく解析されているページの多くが，RSS Feed を提供しているページだったため，オリジナルの Feed を購読しているユーザが多いと考えられる．

また，100 ページ中 41 ページには，時系列情報が含まれなかった．中には，漸進的に記述されるようなエントリを含むページもあったが，明示的な日付表現を伴わないものは本研究の対象外であるため，時系列情

[13] http://www.tom.sfc.keio.ac.jp/~takot/diary/
[14] http://www.itmedia.co.jp/news/
[15] http://www.itmedia.co.jp/
[16] http://www.goku2.com/
[17] http://dailynews.yahoo.co.jp/fc/
[18] http://internet.watch.impress.co.jp/static/yajiuma/
[19] http://www.forest.impress.co.jp/
[20] http://aaa.parfait.ne.jp/metal/diary3/nicky.html
[21] http://www.faireal.net/
[22] http://www6.ocn.ne.jp/~katoyuu/
[23] http://www.outdex.net/diary/
[24] http://portal.nifty.com/
[25] http://kiwofusi.at.infoseek.co.jp/
[26] http://pasokon-yugi.cool.ne.jp/
[27] http://www.moriyama.com/diary/2005/diary.htm
[28] http://techside.net/bbsf.html
[29] http://tiyu.to/title.html
[30] http://www.gdm.or.jp/voices.html
[31] http://cclub.cc.tut.ac.jp/~hachi/cgi-bin/diary/diary.cgi
[32] http://www14.big.or.jp/~onmars/

報ではないとし，この項目にカウントしている．

また 100 ページ中 12 ページでは，日付表現を認識できず，RSS Feed の自動生成に失敗していた．“050719” など，明確に日付であると判定できないケースについては，本システムでは日付と認識しないような方針で処理を行っている．また，構造解析の計算量を減少させるため，本文中に含まれる日付を出来るだけ認識しないような方針で日付の認識を行っている．この 12 ページでは，これらの制約により日付が認識されず，結果として RSS Feed の自動生成に失敗している．今後，認識可能な日付表現の網羅性を高めると同時に，本文中の日付かどうか判定する処理のチューニングが必要であると考えられる．

100 ページ中 4 ページで構造解析ミスが発生していた．構造解析ミスには，以下の 3 種類があった．

- 抽出すべきエントリの一部しか抽出できていない
- 日付と記事の対応が正しくない (日付が本来の記事ではなく，次の記事の日付として解析されている)
- 本文とタイトルが同じタグでマークアップされているため，タイトルの認識に失敗し，記事単位のエントリが抽出できていない

今後，このようなケースにも対応できる構造解析手法の開発が必要であると考えられる．

“フレームページ” のエラーは，フレームページに対する処理を行ってしまっており，子フレームの中身が処理されていないケースである．子フレームの URL を指定すれば，どちらも正しく解析できるケースであった．フレームの URL が指定された場合，子フレームを自動処理するような，ユーザビリティの向上が必要である．

“日付の誤認識” は，“Volume 4/1, May 2005” という例に対し，日付部分は，“1, May 2005” であるのに対し，“4/1” を日付として認識してしまっていた．また，“パターン発見不可” は，ページ中に一つのエントリしかないため，パターン解析が出来ず，RSS Feed を生成できなかったケースであった．

5 おわりに

本論文では，時系列情報を記述する Web ページ中の日付表現とタイトルを検出することにより，RSS Feed を自動的に生成する手法を提案した．

RSS によるメタデータ配信が盛んに行われるようになってきたが，RSS Feed を出力するサイトは，CMS を使う一部のサイトに限られているのが現状である．なぜなら，人手で記述されるような Web サイトでは，追加のメンテナンスコストが必要になるためである．

本手法に基づいて構築したシステム “なんでも RSS!” は，2005 年 5 月 9 日から `http://blogwatcher.pi.titech.ac.jp/nandemorss/` で公開中であり，広く利用されている．また，初期の分析では，日付表現を伴い記述される時系列情報を含むページの多くで，自動的な RSS Feed の生成に成功している．

今後の課題は，提案手法の評価である．評価は，(1) 構造解析に関する評価，(2) タイトル生成に関する評価の二点を行う予定である．また，さらなる課題として，生成された Feed のタイプ (掲示板，weblog，新着情報，イベント案内など) の自動的な判定や，取得した item の内容に基づく分類などが考えられる．また，今後より多くのページで正しく RSS Feed を自動生成できるよう，システムをチューニングしていく必要があると考えている．

参考文献

[1] RSS-DEV Working Group. Rdf site summary (rss) 1.0. `http://web.resource.org/rss/1.0/spec`, 2000.

[2] Dan Libby. Rss 0.91 spec, revision 3. `http://my.netscape.com/publish/formats/rss-spec-0.91.html`, 1999.

[3] RSS at Harvard Law. Rss 2.0 specification. `http://blogs.law.harvard.edu/tech/rss`, 2002.

[4] 神崎正英. Rss – サイト情報の要約と公開. `http://www.kanzaki.com/docs/sw/rss.html`, 2001.

[5] David R. Karger and Dennis Quan. What would it mean to blog on the semantic web? In *Proc. Third International Semantic Web Conference*, pp. 214–228, 2004.

[6] Mark Pilgrim. dive into mark - rss autodiscovery. `http://diveintomark.org/archives/2002/05/30/rss_autodiscovery`, 2002.

[7] Dave Raggett. Clean up your web pages with html tidy. `http://www.w3.org/People/Raggett/tidy/`, 1994.

[8] 南野朋之, 鈴木泰裕, 藤木稔明, 奥村学. blog の自動収集と監視. 人工知能学会論文誌, Vol. 19, No. 6, pp. 511–520, 2004.

[9] James Clark and Steve DeRose. Xml path language (xpath) version 1.0. `http://www.w3.org/TR/xpath`, 1999.