エンジニアリングデザイン

学内情報配信システム

2010SE273 山下健太　2010SE237 田中秀明　2010SE199 新海由侑
2010SE162 小川詩織　2010SE175 大沢沙希

1. はじめに

南山大学のサイト(http://www.ic.nanzan-u.ac.jp/)は図 1 のような構成になっている. 貸与 PC で設定を変更せずにインターネットにアクセスすると，ホームページとして SETO/gakusei.html（瀬戸キャンパス学生情報）が表示される．このページには News & Topics があるが, 情報理工学部や総合政策学部に関係なく情報を表示しているため, よく文章を読まなければどれが自分に適している情報かが分かりにくい．また, SETO/gakusei.html の前のページにあたる KYOMU/INFO/kyomu_index.htm (瀬戸キャンパスホーム）には，検索エンジンがあるが，SETO/gakusei.html にはないため, 知りたい情報をすぐに見つけることが出来ない．今回の研究では，情報理工学部生を対象に，News & Topics に掲載される情報を取り扱いやすくする．今回とりあげられた主な南山大学サイト内の構造は図 1 のようになっている．また，SETO/gakusei.html は図 2 のように 3 つのフレームで構成されている.

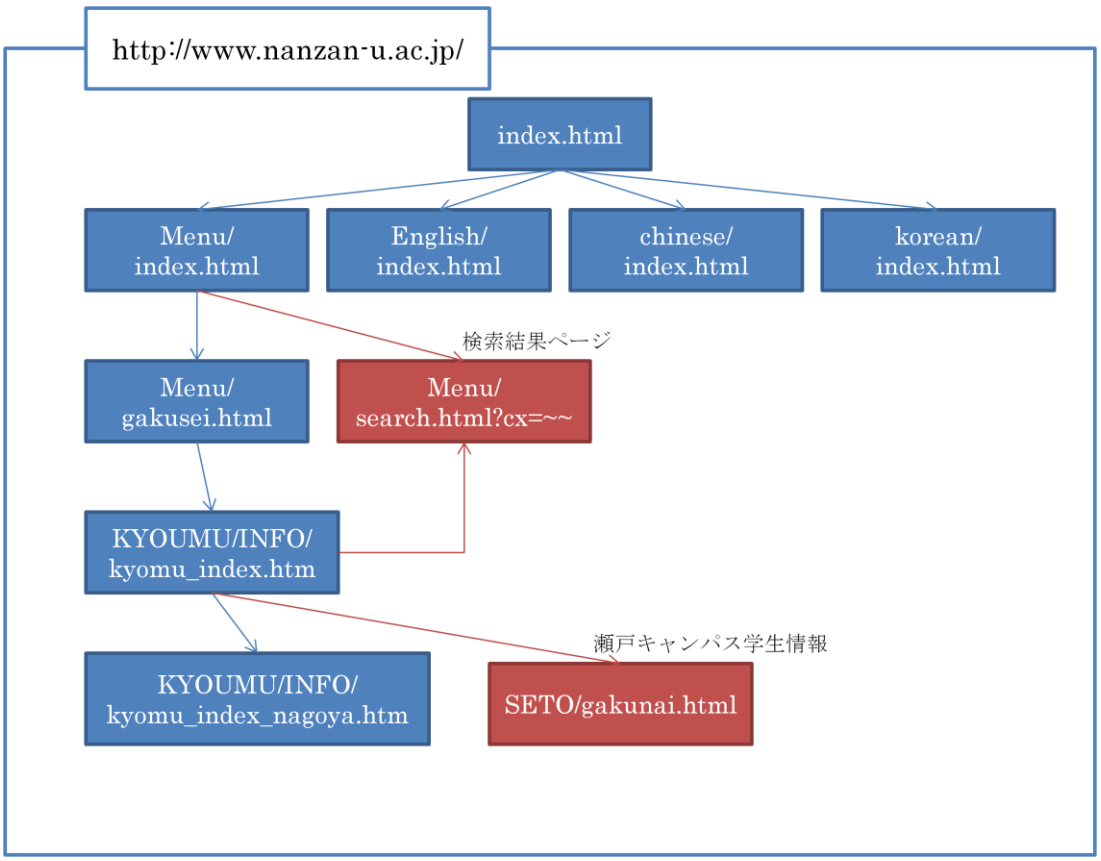

図 1

図 2

2. 問題と解法

現状：

SETO/gakusei.html の News & Topics では，情報理工学部や総合政策学部に関係なく情報を表示している．また，News & Topics 内には「重要」，「注意」，「NEW」などのタグがついているが，これらの情報は更新順に並べられているため，どの情報がどのくらい重要で期限が迫っているものかどうかが分からない．したがって，よく文章を読まなければならない．また，SETO/gakusei.html には検索エンジンがないため，そのページ内では検索をすることができない．

ターゲットにする問題：

情報理工学部生が，更新される学生情報から必要とする情報を取得する．

代替案：

- ✧ スマートフォンやパソコンなどのブラウザを用いて，自分で定期的に SETO/gakunai.html を確認する
  - ➢ 現在ほとんどの人が取っている方法であり，利便性は低い

✧ 定期的に確認を行う人が，手作業でメール配信をする
  - 手元に情報を保持することができるため，非常に便利ではあるが，配信者の負担が大きく，実現性は低い

✧ News & Topics の情報を，日付，重要度，学部などの内容で並べ替えできるようにする
  - 「日付」,「学部」などの既存ではないタグをつける手間がかかる．また，ソートで並べ替えるなど，専門的すぎて時間内に作成することができないと考えられる

✧ 1 週間程度のカレンダーを作成し，そこに期限が迫っているなどの情報を表示する
  - 1 週間の情報を 1 度に見ることができるため，便利ではあるが，作成する人の負担が大きく，実現性は低い

✧ タグをつけるなど，現在使われているページ全体の根本的な見直し
  - 非常に時間がかかるため，作成できない

✧ 学生課が，ページを更新すると同時に学生にメールを配信する
  - 学生課に頼む必要があり，現実的でない上，我々の仕事ではなくなる

✧ SETO/gakunai.html に Menu/index.html のものと同等な検索エンジンを設置
  - 実現性があり，利用者にとって活用しやすい方法であると考えられたため，最初の解法候補であり，実装まで行った

瀬戸キャンパスの生徒は主に SETO/gakunai.html をホームページとして使用するため，ここに検索エンジンを設置することで，より検索を容易にできるようになると考えられる．目の付きやすさ，Menu/index.html との統一性，スクロールバーの有無を考慮し，①(SETO/gakunai_head.html)のフレームに検索エンジンを追加する．実際に SETO/gakunai_head.html に検索エンジンを追加したものが図 3 である．今回実装したものは，学外ページ全体からの検索を行うものである．しかし，検索エンジンの下部に「全体」「学生情報」「学内」など検索条件を選択できるラジオボタンを設置することにより，利便性が向上すると考えられる．検索エンジンの実装にあたって，google カスタム検索エンジンを使用した．以下に追加，変更した部分のソースを示す．

```
<td rowspan="2">
<FORM method="GET" action="http://www.google.co.jp/search">
<INPUT type="text" name="q" size=31 maxlength=255 value="">
<INPUT type="submit" name="btnG" value="検索">
<INPUT type="hidden" name="hl" value="ja">
<INPUT type="hidden" name="sitesearch" value="ic.nanzan-u.ac.jp/SETO/">
<INPUT type="hidden" name="domains" value="ic.nanzan-u.ac.jp/SETO/">
<INPUT type="hidden" name="ie" value="Shift_JIS">
<INPUT type="hidden" name="oe" value="Shift_JIS">
</FORM>
</tr>
```

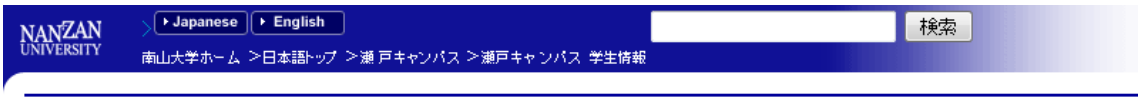

図 3

解法：

RSS 配信

選択理由：

自動配信することによって，配信者の必要性をなくし，人が作業を行う負担をなくすことができる．それにより，利用者がシステムを意識せずに利用できるので利便性が高い．また，配信されたデータであればオフラインでも情報を確認できるため，インターネットに繋がらない環境下でも内容の確認が可能である．この点で検索エンジンよりも優れていると考えた．配信されたデータについては，取捨選択することができ，自分の意思でタグ付けもできるため，情報の管理が容易である．スマートフォン上でもアプリを用いて，RSS 配信サービスを利用できる．したがって，スマートフォンが普及している現在では，高い利用者数が獲得できる．

3. 技術的な解析

初期段階の構想は以下のとおり

① ダウンロード
   I. HTML ファイルをダウンロード
② 変換
   I. RSS 配信
③ アップロード
   I. RDF ファイルをアップロード
   II. ログを残す
④ リーダで読む

これらの解析について図示すると図 4 のようになる.

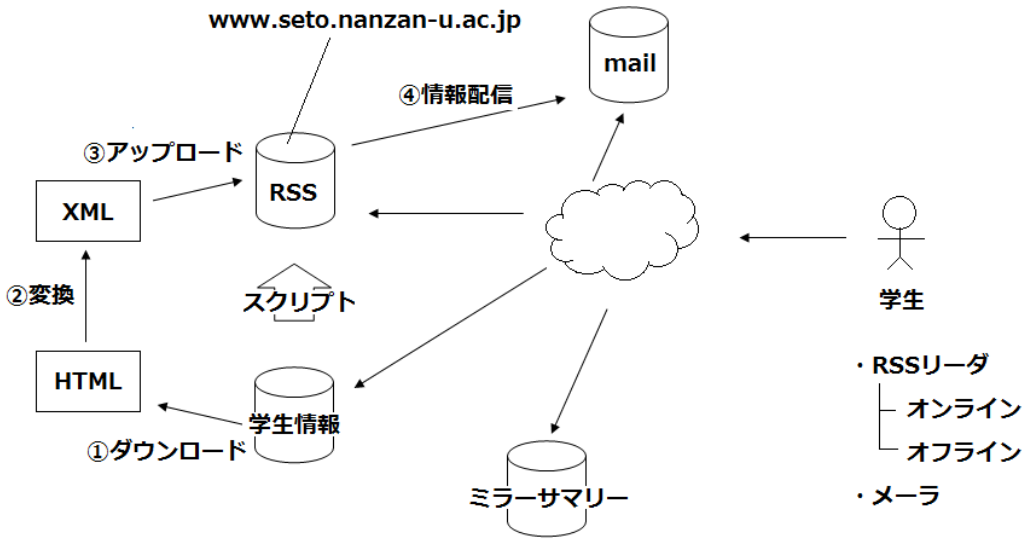

図 4

4. 設計の結果･期待される効果

本システムは次の 5 つの機能で実現することとなった.

① HTML ファイルをダウンロード
② HTML ファイルを RDF ファイルに変換
③ RDF ファイルをアップロードする
④ 処理のログを残す
⑤ 以上の処理を，自動的に実行する

これらをコンテキスト図にまとめると，次のようになる.

図 5

コンテキスト図を細分化し，機能間のデータの流れを表すと次のようになる.

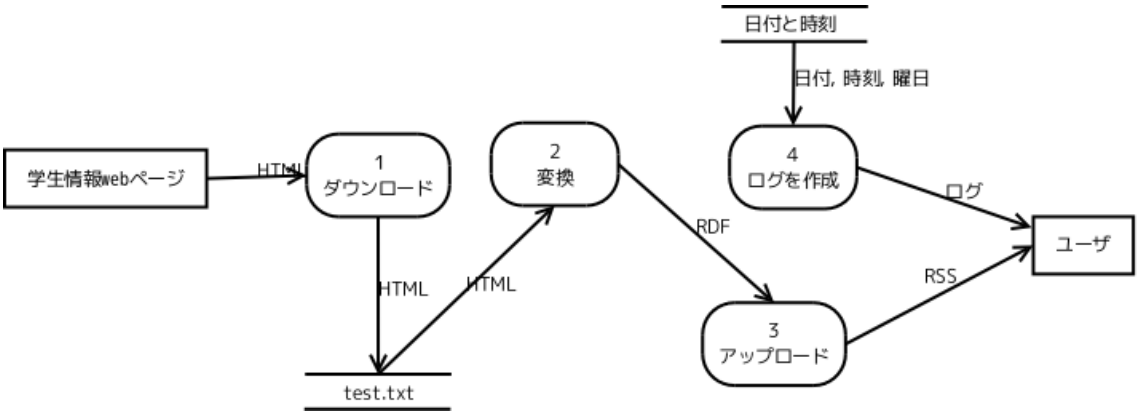

図 6

以上 5 つの機能の詳細は次のとおりである.

① ダウンロード

SETO/gakunai.html の HTML ファイルをダウンロードし，サーバのディレクトリに保存する．ダウンロードは以下の手順で行う．プログラムを用いて行う必要がある場合のみ，説明の後に部分的なコードを記す.

I. 保存用ファイルを開く

HTML ファイルの保存先として，サーバのルートディレクトリにあるファイルtest.txt を開く.

```
# ファイルを開く
open(FILE, ">test.txt") or die;
```

II. HTML コードを取得

wget 関数を用い，目的のページのHTML コードを取得する.

```
# 配列にHTML コードを格納する
$url = 'http://www.ic.nanzan-u.ac.jp/SETO/pub/topics.html';
my @array = `wget -o /dev/null -O - ¥"$url¥"`;
```

III. ファイルtest.txt にHTML コードを書き込む.

print 関数を用い，取得したHTML コードをファイルtest.txt に書き込む．なお，本来ならばコードをこの様にtxt ファイルに保存する必要はないが，今回のシステム製作ではダウンロード機能と変換機能の担当者が異なるため，データの受け渡しとしてこの様な手段を取った.

```
# HTML コードをファイルに書き込む
print FILE @array;
```

IV. test.txt ファイルを閉じる

ファイルtest.txt を閉じる．なお，本来ならばここでファイルを閉じる必要はないが，ダウンロード機能単体でもテストを行えるように，この段階で一旦ファイルを閉じる．

```
# ファイルを閉じる
close(FILE);
```

② 変換

RSS配信に関する実装はperlを用いて行った．今回の実装では，HTMLファイルから記事のタイトルと，その内容のURLのみを取得し，XMLの形式に変換した．

今回の変換は，test.txt において一定の規則が成り立っているため，その規則を利用して解析を行う．この際に 2 つの変換パターンがある．1 つ目は，更新日時から差分だけを変換する方法．2 つ目は，毎回全ての項目を変換する方法である．

今回の学生情報を変換する手順は，以下のとおり．

I. test.txt を解析（目印「<!--」のカウント）

ここでは，test.txt 中の目印「<!--」をカウントする．この作業によって，この後の手順Ⅱ～Ⅳを行う際に，変換の正確性を高めることができる．

II. test.txt を解析（アドレス抽出）

test.txt 中の目印「<A href=”」に該当する行を抽出する．

III. test.txt を解析（アドレス部分を XML 形式に変換）

手順Ⅱで抽出した文字列の変換を行う．まず，文字列先頭の「<A href=”」を消去し，次に，文字列の末尾の「“ >」を消去する．これによって，純粋なアドレスのみを抽出することができる．さらに，純粋なアドレスにも「http:～」から始まるものと「/」から始まるものがある．後者のアドレスは，外部からリンクできないため，「http://www.ic.nanzan-u.ac.jp」を付け足すことによって，リンクできるようにする．抽出したアドレスを配列 address に追加する．

IV. test.txt を解析（タイトル抽出）

手順Ⅱと同様にして,「</A>”」 があれば，その前の文字列を抽出する．そして，タイトルを配列 title に追加する．

V. ex.txt に書き出し

ここでは，ex.txt に上書きで情報を書き込んでいく．手順ⅢとⅣで作成した配列 address と配列 title を rdf の規則にしたがって記述していく．

VI. top.rdf に書き出し（index.txt を書き出す）

ここでは, top.rdf に上書きで情報を書き込んでいく. rdf の規則にしたがって，最初に示すヘッダーの情報を index.txt から書き込む.

VII. top.txt に書き出し（文字コード変換）

ここでは，ex.txt に記されたものを文字コード変換して，top.txt に上書きで書き込んでいく．

VIII. top.rdf に追加書き出し

ここでは, 手順Ⅵでつくった top.rdf に追加として top.txt の情報を書き込んでいく．全て書き込み終えたら，rdf の規則にしたがって,「</rdf:RDF>」をファイルの最後に書き込む.

IX. 変換完了（HTML を出力）

ここでは，変換が完了したことを画面に表示するために，html を出力する．

以上が変換に必要な手順である．なお，これらの作業は rdfmaker.cgi を用いて行われている．これを含めた RSS 配信に必要なファイルを以下に記述する．

◆ test.txt

図 2 の SETO/gakunai.html フレーム③の HTML 文書を書き込んだファイル．text.txt に書き込まれる学生情報のソースは以下のような形式．必ずこの形式で書かれているため,「<!-- が含まれる文章をカウント」や,「<A href=”を削除」などの操作によって XML 文書への変換を行っている．変換後，変換についてのソースは以下を参照．記事によって異なる部分については，太文字で表記してある．

```
<!-- YYYYMMDD -->
<div>
<img src="タグの画像 URL " alt="タグ">
        <span>
```

<A href="**記事の URL**" >**記事タイトル**</A> (**YYYY**/**MM**/**DD**)

◆ ex.txt

test.txt から URL とタイトルのみを抽出し，これらに関して XML の形式にしたファイル．変換後の学生情報のソースは以下のとおり．記事によって異なる部分については，太文字で表記してある．

<item rdf:about="**記事の URL** ">
<title>**記事タイトル**</title>
<link>**記事の URL**</link>
</item>

◆ index.txt

XML のヘッダを記述したファイル．ex.txt には記事情報のみを，新着情報が来る度に 1 行目から書き込んでいくため，ヘッダは別で用意している．

◆ top.rdf

最終的な XML ファイル, これを用いて RSS 配信を行う. ヘッダである index.txt と記事情報である ex.txt を合わせたものを保存している．

◆ rdfmaker.cgi

HTML の取得，解析，変換を行うファイル．以下に全ソースコードを記述する．

```
#!/usr/bin/perl

# 更新履歴を残す
# time関数で得られた数値をlocaltime関数で整形し各変数に入れる
($sec,$min,$hour,$mday,$mon,$year,$wno) = localtime(time);

# 曜日は 0~6 の値が返されるので配列で扱う
@wdays = ('SUN','MON','TUE','WED','THU','FRI','SAT');

# 返された値を更にsprintf関数で整形し，書式を読みやすいようにする
$nitizi =
sprintf("%04d/%02d/%02d(%s) %02d:%02d",$year+1900,$mon+1,$mda
y,$wdays[$wno],$hour,$min);

# 準備
$fname = 'log.txt';
$cufld = '.';
$dbfld = '/w_test/';
```

```
$path = $ENV{'PATH_INFO'};

# 動作環境セット
if (($i = index($path, $dbfld)) != -1) {
        $cufld = '..' . substr($path, $i, $i + length($dbfld) - 1);
}

# ファイル書き込み
open(FILE, ">>$cufld/$fname") or die;
print FILE "$nitizi";
print FILE "¥n";
close(FILE);

print @nitizi;

# HTMLを解析・変換
$fname = 'test.txt';
$cufld = '.';
$dbfld = '/w_test/';
$path  = $ENV{'PATH_INFO'};

if (($i = index($path, $dbfld)) != -1) {
        $cufld = '..' . substr($path, $i, $i + length($dbfld) - 1);
}

# 学生情報の右側のフレームからHTMLソースを持ってくる
$url = 'http://www.ic.nanzan-u.ac.jp/SETO/pub/topics.html';
my @array = `wget -o /dev/null -O - ¥"$url¥"`;

# ファイル書き込み
open(FILE, ">test.txt") or die;
print FILE @array;
close(FILE);

open(IN,"test.txt");
$count = 0;
@xx = <IN>;
foreach $yy (@xx) {
        if($yy =~ /<!-- /) {
        #   <!-- があったら
                $count++;
        }
}
close(IN);

open(IN, "test.txt");
$first = "http://www.ic.nanzan-u.ac.jp";
$num = 0;
@xx = <IN>;
foreach $yy (@xx) {
```

```
		if($yy =~ /<A href="/ && $num < $count) {
		#<A href=" があったら(URL抽出)
			$yy = substr($yy,10);
			#文字列先頭の <A href=" を消去(XML用に整形その1)
			if($yy =~ /" target="_blank">/){
				$yy = $`;
			} else {
				substr($yy,-5,5) = "";
				#文字列の末尾の " > を消去(XML用に整形その2)
			}
			if($yy =~ /^¥//){
				$yy = $first.$yy;
			}
			$address[$num] = $yy;
			$num++;
		}
}
close(IN);

open(IN, "test.txt");
$num = 0;
@xx = <IN>;
foreach $yy (@xx) {
		if($yy =~ /<¥/A>/ && $num < $count) {
		# </A>" があったら(タイトル抽出)
		$title[$num] = $`;
		$num++;
		}
}
close(IN);

open(OUT,"> ex.txt");
$num = 0;
for(;$num < $count;$num++){
		print OUT"¥n";
		print OUT'<item rdf:about="';print OUT$address[$num];print
OUT ' "> ';
		print OUT"¥n";
		print OUT"<title>";print OUT$title[$num];
		print OUT"</title> ¥n";
		print OUT"<link>";
		print OUT$address[$num];
		print OUT"</link> ¥n";
		print OUT"</item> ¥n";
}
close(OUT);

open(OUT,"> top.rdf");
open(IN, "< index.txt");
while(<IN>){
```

```
        chomp;
        print OUT "$_¥n";
}

close(OUT);
close(IN);

open my $r, '<:encoding(cp932)', 'ex.txt';
open my $w, '>:encoding(UTF-8)', 'top.txt';

while(my $d = <$r>) {
        print $w $d;
}

close $w;
close $r;
open(OUT,">> top.rdf");
open(IN, "< top.txt");
while(<IN>){
        chomp;
        print OUT "$_¥n";
}
print OUT "¥n</rdf:RDF>";
close(OUT);
close(IN);

#HTMLを出力する
print "Content-type: text/html¥n¥n";
print "<HTML>¥n";
print "<P>Hello World</P>¥n";
print "<P>test.cgiの後にrdfmaker.cgiを実行した</P>¥n";
print "<A HREF=test.txt>test.txt</A><BR>¥n";
print "</HTML>¥n";

exit(0);
```

③ アップロード

生成したRDF ファイルをアップロードし, RSS リーダなどを用いて閲覧できる状態にする．ここまででRDF ファイル作成してサーバのルートディレクトリに保存した.

④ ログを残す

time 関数を利用し，このシステムが実行された日付と時刻を記録する.

```
#time 関数で得られた数値をlocaltime 関数で整形し各変数に入れる
($sec,$min,$hour,$mday,$mon,$year,$wno) = localtime(time);
```

```
# 曜日は0～6 の値が返されるので配列で扱う
@wdays = (’SUN’,’MON’,’TUE’,’WED’,’THU’,’FRI’,’SAT’);
# 返された値をさらにsprintf 関数で整形し，書式を読みやすいようにする．
8 $nitizi =
sprintf("%04d/%02d/%02d(%s)%02d:%02d",$year+1900,$mon+1,$mday
,$wdays[
$wno],$hour,$min);
#0ログファイルに日付と時刻を書き込む
open(FILE, ">>$cufld/$fname") or die;
print FILE "$nitizi";
13 print FILE "¥n";
close(FILE);
```

⑤　自動的に処理を実行する

perl 言語はサーバ上の cgi ファイルへのアクセスがあったときのみ処理が行われるため，perl プログラム内 の処理によって自動的に処理を開始することはできない．そこで，指定した URL に定期的にアクセスを行う web サービスを利用し，プログラムを定期的に実行させることにした．

ここでは，次の web サービスを利用した．

Free Web Cron Service

http://www.mywebcron.com/

Free Web Cron Service による定期的な実行を開始したあと，ある 1 日のログを確認すると次のような記録が 残っていた．

2012/10/14(SUN) 00:07
2012/10/14(SUN) 00:24
2012/10/14(SUN) 01:08
2012/10/14(SUN) 01:25
（中略）
2012/10/14(SUN) 22:07
2012/10/14(SUN) 22:24
2012/10/14(SUN) 23:07

これにより，処理の自動化に成功した事が確認できた．

以上の実装により，次のようなシステムが完成した．

http://ed-cgi-test.toypark.in/(図7)

このシステムは実際に稼働し，学生情報の　RSS を配信している．

図 7　http://ed-cgi-test.toypark.in/

実際にブラウザ上のgoogle Reader[2]と端末(iPod touch 4g)のアプリ[3]でRSS購読を行った図7と図8を以下に示す.

図 8

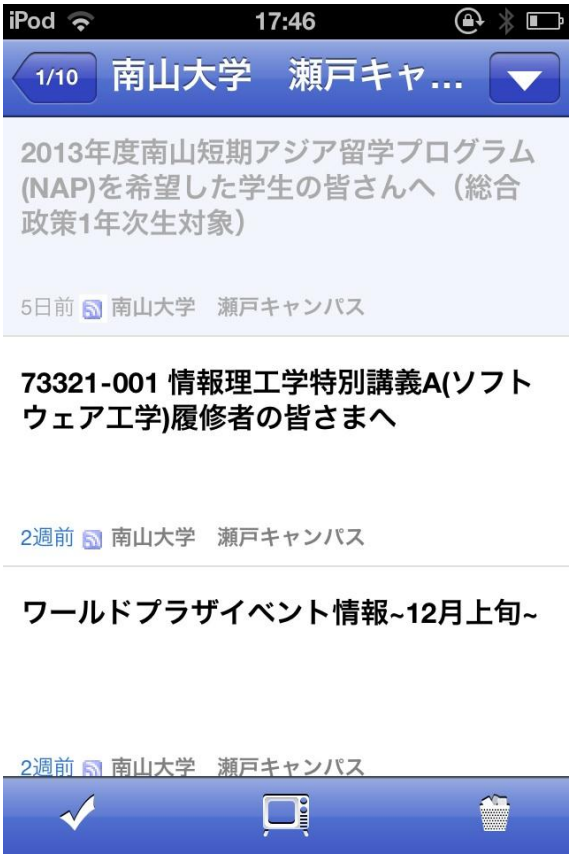

図 9

5. 結論

学生情報に掲載される情報をいつでも確認することができる．取捨選択をして自分に必要な情報のみをすぐに発見できるという点では，今回実装を行ったRSS配信は効果的なものだと考えられる．しかし，test.txtのサンプルを見て分かる通り，元々のファイルには日付情報やタグなども付けられている．よって，これらの情報も取り込み，XMLファイルに書き込むことで，更に便利な配信が可能になると考えられる．タグ付けを用いて，情報理工学部生が対象のものは配信し，総合政策学部対象は配信しないなどの拡張をすることができれば，取捨選択の必要性が減ることになると思われる．

今回作成したものは，タグの特性を生かして情報を抽出，変換している．このため，例えば他の大学の学生情報についても，何か目印として使えるものを見付けだしてコードを書き換えることによって，同様にRSS配信が可能であると考えられる．しかし，これでは人による作業が入り込み，仮に目印となるものがない場合に応用することができない．よって，今回のようにタグの中の目印を探すのではない方法でHTMLをXMLに変換することができれば，更に簡潔にRSS配信が可能なのではないかと思わ

れる．

また，大切な情報であり，かつ掲載されている場所が分かりにくいものは学内専用ページに多く，RSS配信はこれらに対して対応できていない．よって，今回実装に至った検索エンジンを学内専用まで範囲を広げ，RSS配信と同時に用いると良いと思われる．これによって「自分向けの情報の持ち運び」と「必要な情報の検索」が可能となり，現在の学生向け情報発信方法よりも利便性が向上すると考える．

6. コスト解析

山下：①ダウンロード
　　③アップロード
　　④ログを残す(8 時間)
田中・新海：②変換
　　レポート(12 時間)
小川・大沢：レポート作成(8 時間)

7. 参考文献

[1] google：カスタム検索エンジン，入手先〈http://www.google.co.jp/cse/?hl=ja〉（参照 2012-12-06）．

[2] google：Google リーダー，入手先〈http://www.google.co.jp/reader/view/〉（参照 2012-12-10）．

[3] itunes.apple.com：iTunes App Store で見つかる iPhone、iPod touch、iPad 対応 iReadG Free:，Perkin Tang,
入手先〈https://itunes.apple.com/jp/app/ireadg-free/id415451262?〉（参照 2012-12-10）．