X-treeDiff+ 기반의 한글 문서에 대한 변화 탐지†

(Change Detection of Hangul Documents Based on X-treeDiff+)

이 석 균*
(Suk Kyoon Lee)

요 약 XML 문서에 대한 변화탐지는 중요한 연구 분야이다. 그러나 한글 문서 파일 형식으로 XML이 지원되고 있음에도 한글 문서의 특성을 반영한 변화 탐지 연구는 아직 미비하다. 한글 문서는 일반적인 XML 문서와는 달리 서식 정보의 비중이 매우 커서 X-treeDiff+와 같은 일반적인 XML 문서의 변화탐지 알고리즘을 그대로 적용하기에는 적합하지 않다. 본 논문에서는 한글 문서에 대한 변화탐지를 위해 새로운 내용 기반의 대응 알고리즘을 제안하고 이를 X-treeDiff+에 구현하였다. 실험을 통해 제안된 알고리즘은 대부분의 편집과정의 문서에 대해 우수한 성능을 보이고 있음을 제시했다.

핵심주제어 : 한글, X-treeDiff+, 변화탐지, XML

Abstract The change detection of XML documents is a major research area. However, though XML becomes a file format for Hangul documents, research on change detection of Hangul documents based on the characteristics of Hangul documents is rather scarce. Since format data in Hangul documents are very large, which is different from ordinary XML documents, it is not proper to apply general XML change detection algorithms such as X-treeDiff+ to Hangul documents without any change. In this paper, we propose new contents-based matching algorithm and implement it in X-treeDiff+. The result of our testing shows better performance for most documents in editing process.

Key Words : Hangul, X-treeDiff+ Change Detection, XML

1. 서 론

문서 작업 과정에서 다양한 버전의 동일 명칭의 문서들이 여러 컴퓨터에 또는 여러 폴더에 존재할 때, 원본 문서의 확인 또는 문서 간의 편집내용의 식별은 쉽지 않다. 특히 다수 사용자에 의한 편집 작업이 여러 파일에 동시에 이루어질 때 수정 내용을 식별하고 통합하는 일을 더욱 쉽지 않다. 최근 오피스 문서들의 변화 탐지, 병합, 그리고 버전 관리는 중요한 문제로 등장한다[1].

텍스트 문서의 변화(차이)를 탐지하는 diff는 리눅스의 주요 도구로 문서의 버전관리, 병합의 핵심 기능을 담당한다. XML의 사용이 보편화됨에 따라 XML 문서의 변화탐지도 중요한 연구 분야로 관심을 받고 있다[1]. 한편, XML은 오피스 문서의 파일 형식으로 사용

† 이 연구는 2010년도 단국대학교 대학연구비 지원으로 연구되었음.
* 단국대학교 공과대학 컴퓨터학부

되기 시작하는데, 오피스 문서는 데이터 중심의 XML 문서에 비해 서식 정보가 큰 비중을 차지하고 있어 기존의 XML 문서의 변화탐지 기법을 적용하기에는 적절치 않다. 본 논문에서는 XML 문서의 변화탐지 알고리즘인 X-treeDiff+를 기반으로 한 한글 문서의 변화 탐지에 대한 연구 결과를 소개한다.

트리구조 문서에 대한 변화탐지는 70년대부터 연구가 진행되었다[2,3,4,5]. 초기에는 다양한 문제들에 대한 최적(optimal) 알고리즘에 대한 연구가 이루어졌으나 이는 NP-Hard 문제[6]로 실시간 처리에 적당하지 않았다. 최근에는 실시간 처리를 위한 알고리즘들이 연구되었는데[7,8,9,10,11,12,13,14,15], 이들 중 XML의 적용에 적합한 순서 트리(ordered tree) 모델의 대표적인 알고리즘들로는 XyDiff[7], X-treeDiff+[10], XMDiff[12], DiffXML[13], DeltaXML[14,15] 등이 있다. 이들 알고리즘은 삽입, 삭제, 갱신의 기본 연산들을 제공한다.

XyDiff는 XML 문서들의 웨어하우스 구축을 위한 알고리즘으로 위의 기본연산들 이외에 이동 연산을 지원하고 실행시간이 트리의 노드 수 n에 대해 O(nlogn)의 성능을 보이며 DiffXML도 이동연산을 지원하나 삽입, 삭제가 노드 단위로만 이루어지고 e를 삽입, 삭제, 이동된 노드 수라 할 때 실행시간이 O(n*e + e2)의 비용을 갖는다. 한편, X-treeDiff+는 해킹에 의한 웹 페이지의 변조 공격을 실시간 탐지하기 위한 시스템에 사용되었던 X-treeDiff[16]을 확장한 알고리즘으로 편집스크립트 생성 기능을 갖으며 이동연산은 물론 복사연산까지 지원하며 실행시간이 O(n)의 성능을 보인다.

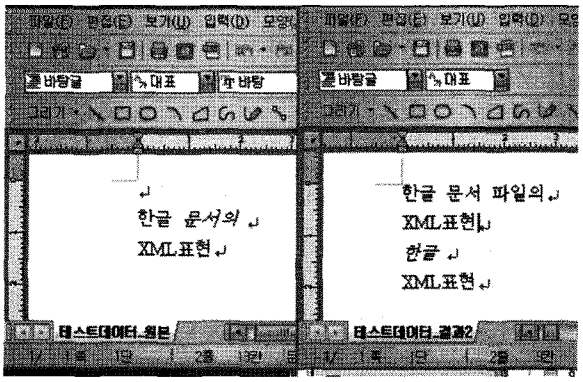
본 논문에서는 X-treeDiff+를 기반으로 한글 문서들에 대한 변화탐지를 수행한다. 우선 한글 문서의 XML 파일 형식의 특징을 분석하여 X-treeDiff+을 한글 문서의 특징을 고려한 내용기반 대응 알고리즘을 제안하고 실험을 통해 성능 분석을 수행한다. 논문의 구조는 다음과 같다. 2절에서는 한글의 XML 형식의 문서구조와 X-treeDiff+의 알고리즘 개요를 소개하고, 3절에서는 내용기반 대응 알고리즘을 제시하고 4절에서는 실험결과에 대한 분석 결과를 제시한다.

2. 연구 배경 및 기존 연구 소개

2.1 한글의 XML 형식의 문서구조

한글의 XML 파일 형식은 HWPML로 표현되는데 이는 W3C XML 기반의 개방형 마크업 언어이다[17]. 한글 워드 프로세서는 이를 사용하여 내용을 XML 문서로 나타내며 저장 시 확장자로 .hml과 .xml을 사용한다. .hml 파일 형식은 문서 내용 및 서식 내용이 하나의 파일에 저장한다. .xml 파일 형식은 문서의 구조와 내용만을 저장하고 대부분의 서식 정보 및 스타일 시트 등의 내용은 .xsl 파일에 저장한다. 본 논문에서는 .xml과 .xsl 파일 형식을 전제로 설명한다.

<그림 1> (a)의 원본문서를, <그림 2>에서는 이를 .xml 파일 형식으로 표현된 내용을 보인다. HWPML의 문서 구조는 다음과 같다. 루트 엘리먼트로 <HWPML>이 있고 <HWPML>의 자식 엘리먼트들로는 <HEAD>, <BODY>, <TAIL>이 존재한다.



(a) 원본문서 (b) 결과문서

<그림 1> 예제 한글 문서

HEAD 엘리먼트는 문서요약정보와 글꼴, 글자속성, 문단속성 등 서식에 관한 문서의 전반적인 정보를, 그리고 TAIL 엘리먼트는 바이너리 데이터 저장, 스크립트 코드 등의 내용으로 구성된다. 본문을 의미하는 BODY 엘리먼트는 자식 엘리먼트로 <SECTION>이 있다. SECTION 엘리먼트는 문단을 의미하는 <P>를 자식 엘리먼트로 포함하며, P 엘리먼트는 자식 엘리먼트로 <TEXT>를 가지는데 이는 텍스트 문자열을 위한 엘리먼트로 테이블, 그림, 수식 등 다양한 자식 엘리먼트들을 가진다. TEXT 엘리먼트의 대표적인 자식으로는 <CHAR>가 있는데 이는 글자를 나타내는 엘리먼트로 문서 내의 대부분의 글자 부분을 표현할 때

사용된다. 상세한 HWPML의 문법과 시맨틱은 한글 문서 파일 구조[17]과 한글과 컴퓨터의 HWPML의 DTD[18]에 제시된다.

HWPML의 엘리먼트들의 속성들은 매우 다양하다. <그림 2>의 예의 경우 P 엘리먼트의 속성 중에 ColumnBreak와 PageBreak는 각각 단 나눔과 쪽나눔 여부를 나타내는 속성이고, ParaShape와 Style은 각각 PARASHAPE 엘리먼트의 id속성 값과 STYLE 엘리먼트의 id 속성 값을 참조하는 속성들이고, PARASHAPE과 STYLE은 문단의 서식과 스타일 정보를 나타내는 엘리먼트이다. TEXT 엘리먼트의 CharShape 속성도 글자의 서식을 나타내는 CHARSHAPE 엘리먼트의 Id 속성값을 참조한다.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<?xml-stylesheet type="text/xsl" href="XML테스트.xsl"?>
<HWPML Style="export" SubVersion="7.0.0.0" Version="2.7">
 <BODY>
  <SECTION Id="0">
    <P ColumnBreak="false" PageBreak="false"
       ParaShape="1" Style="0">
      <TEXT CharShape="1"><CHAR/>
    </P>
    <P ParaShape="1" Style="0">
      <TEXT CharShape="1"><CHAR>한글 </CHAR>
      </TEXT>
      <TEXT CharShape="5"><CHAR>문서의 </CHAR>
      </TEXT>
    </P>
    <P ParaShape="1" Style="0">
      <TEXT CharShape="1"><CHAR>XML표현</CHAR>
      </TEXT>
    </P>
  </SECTION>
 </BODY>
</HWPML>
```

<그림 2> 원본문서에 대한 XML 형식 문서

위의 예제와 설명에서 알 수 있듯이 한글 문서에는 다양한 형태의 많은 서식 정보가 포함되고 있어 실제로 간단한 텍스트로 구성된 한글 문서라 해도 서식 정보의 비중이 훨씬 크다.

2.2 X-treeDiff+의 개요

X-treeDiff+[10,16]는 우선 두 XML 문서를 X-tree로 변환하고 두 X-tree의 노드들 또는 서브트리들 사이의 대응을 결정한 후 이로부터 편집스크립트를 추출한다. 본 논문에서는 두 문서를 편의상 원본문서와 결과문서 또는 원본트리와 결과트리로 호칭한다.

2.2.1 X-tree 정의 및 X-treeDiff+의 대응과정

X-tree는 X-tree 노드들로 구성되는데 X-tree 노드는 트리구조 유지를 위한 필드들과 Type, Value, AttMap, M_Node, M_Type, Op, Opord 등의 필드들로 구성된다. Type은 노드가 엘리먼트 또는 텍스트 노드인지를 구별하고, Value는 엘리먼트 명 또는 텍스트 값을 저장한다. AttMap은 엘리먼트 노드의 경우, 이에 속한 속성들을 속성 명을 키, 그리고 속성 값을 값으로 저장하는 해시 맵이다. M_Node는 대응된 노드의 포인터를, M_Type은 대응유형을, Op는 편집유형을, Opord는 편집연산의 실행순서를 저장한다.

<그림 2>에서 첫 번째 <P>는 빈 문장의 문단, 두 번째 <P>는 "한글 문서의 "의 문자열의 문단, 세 번째는 "XML 표현"의 문단이 표현되어있다.

X-tree에서 노드의 식별을 위해서는 노드의 XPath를 사용하며 X-tree에서는 이를 nID(노드 식별자)라 한다. <그림 2>에서 두 번째 <P> 노드는 /HWPML[0]/BODY[0]/SECTION[0]/P[1]로 표현된다. "XML 표현"은 X-tree의 텍스트 노드로 표시되는데, 이는 /HWPML[0]/BODY[0]/SECTION[0]/P[2]/TEXT[0]/CHAR[0]/text()[0]로 나타낸다. 본 논문에서 간결한 표현을 위해 HWPML, BODY, SECTION, TEXT, CHAR, text() 대신 H, B, S, T, C, t()를 사용한다. 즉 두 번째 <P>는 /H[0]/B[0]/S[0]/P[1]로 표현한다.

한편, 효율적인 대응 과정을 위해 X-tree 노드에 nMD와 tMD 필드들을 포함하는데, nMD 필드에는 노드의 데이터의 해시 값이 저장되어 노드 간의 빠른 비교를 가능하다. tMD 필드는 서브트리 간의 비교를 위해 사용되며 현 노드를 루트로 하는 서브트리에 속한 데이터의 해시 값이 저장된다. 따라서 두 노드의 동일여부는 nMD 값의 비교로, 두 서브트리의 동일여부는 tMD 값의 비교로 확인한다.

원본과 결과문서들을 파싱하여 원본과 결과트리를 생성한 후 5단계의 대응과정이 시작된다. 1단계에서는 두 X-tree의 각 노드들에 대해 tMD 값의 비교를 통해 1-1 대응되는 동일 서브트리의 쌍을 찾아 대응시킨다. 이때 1-n, n-1, 또는 n-n의 대응관계에 있는 노

드들은 추후로 대응을 연기한다. 2단계에서는 1단계에서 대응된 서브트리의 루트로부터 이들의 대응을 상향 확장한다. 이때 대응된 노드 쌍의 부모 노드들을 확인해서 엘리먼트 명이 같으면 부모 노드도 대응시키고 서로 다르면 대응과정을 중단한다. 이 과정을 모든 대응된 노드에 대해 반복한다. 3단계에서는 X-tree의 루트로부터 깊이우선탐색을 수행하면서 아직 대응이 결정되지 않은 노드들에 대해 대응 결정을 한다. 현 방문 노드 n이 대응이 되었으면 대응이 되지 않는 자식 노드들에 대해 우선 tMD 값이 같은 노드가 n의 대응노드의 자식노드들 중에 있는지 확인한다. 발견되면 이들을 대응시킨다. 남은 자식노드들에 대해 nMD 값에 대해서도 동일한 시도를, 그리고 그 후 엘리먼트 명에 대해서도 동일한 대응 시도를 한다. 4단계에서는 대응된 노드 쌍에 대해 대응의 튜닝 작업을 수행하고 5단계에서는 아직 대응되지 않은 노드들 중 tMD 값이 같은 노드들에 대해 대응을 시킨다.

2.2.2 X-treeDiff+의 편집연산

두 X-tree들의 대응 결과로부터 발견된 차이(변화)는 일련의 편집연산들, 즉 편집스크립트를 통해 표현된다. X-treeDiff+는 노드에 대한 편집연산으로 삽입(INSERT), 삭제(DELETE), 갱신(UPDATE_ TEXT), 이동(MOVE), 복사(COPY)를, 속성 편집 연산으로는 INSERT_ATTR, DELETE_ATTR, UPDATE_ATTR을 지원한다.

<그림 1>의 원본문서에 "한글 " 대신 "영어 "로 변경하고 "추가된 문장"을 끝에 추가하는 작업은 다음의 편집연산들로 표현된다.

```
<UPDATE_TEXT
 snid="/H[0]/B[0]/S[0]/P[1]/T[0]/C[0]/t()[0]" tv="영어
"/>
<INSERT      tnid="/H[0]/B[0]/S[0]"      tpos="3"
opord="1">
   <P ParaShape="1" Style="0">
     <TEXT CharShape="1">
       <CHAR>추가된 문장</CHAR>
     </TEXT>
   </P>
</INSERT>
```

위의 UPDATE_TEXT에서 snid는 원본 문서의 편집 대상 노드의 nid로 갱신, 이동, 복사 연산 시 원본 노드를 의미한다. INSERT에서 tnid는 삽입될 장소를 나타내며 tpos는 tnid가 나타내는 노드의 몇 번째 자식노드로 삽입할 지를 표현한다. tnid와 tpos는 이동, 복사연산에서도 유사한 의미로 사용된다. 다음에서는 <그림 1>의 원본문서와 결과문서에 대해 X-treeDiff+의 실행결과로 생성된 편집스크립트를 제시한다.

```
<dblab:es xmlns:dblab=".. 생략" dc="2010-10-24">
 <UPDATE_ATTR snid="/H[0]/B[0]/S[0]/P[1]/T[0]"
    attrName="CharShape" attrValue="5"/>
 <DELETE snid="/H[0]/B[0]/S[0]/P[1]/T[1]"/>
 <MOVE snid="/H[0]/B[0]/S[0]/P[1]"
    tnid="/H[0]/B[0]/S[0]"      tpos="2" opord="2"/>
 <INSERT tnid="/H[0]/B[0]/S[0]/P[0]/T[0]/C[0]"
    tpos="0" opord="1">한글 문서 파일의</INSERT>
 <COPY snid="/H[0]/B[0]/S[0]/P[2]"
tnid="/H[0]/B[0]/S[0]"  tpos="3" opord="3"/>
</dblab:es>
```

INSERT_ATTR과 DELETE_ATTR의 예는 제시하지 않았지만 UPDATE_ATTR과 유사한 형식을 갖는다. 위의 편집스크립트의 내용을 <그림 1>의 원본문서에 대해 실행하면 <그림 2>의 결과문서가 생성된다.

3. 효과적인 혼글 문서의 변화 탐지 기법

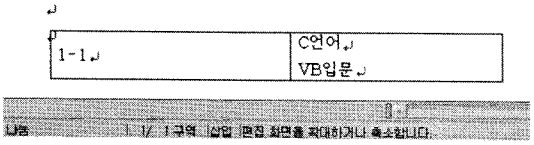
3.1 혼글의 특징과 기존 X-treeDiff+의 한계

HWPML은 오피스 문서 작업을 위한 목적으로 일반적인 XML 문서와는 다른 특징들이 있다. 이는 매우 많은 엘리먼트들과 속성들로 구성되어 있어 모두에 대한 특징 분석은 어렵다. 본 논문에서는 문단(<P>), 테이블(<TABLE>)과 관련 엘리먼트들을 변화 탐지의 관점에서 분석한다. <P>는 사용빈도가 매우 높고 <TABLE>은 복잡한 구조를 가지고 있기 때문이다.

혼글에서는 텍스트, 테이블, 차트, 그림 등의 대부분의 개념들이 <P> 내에 표현되고 있어 <P>의 사용은 매우 빈번하다. 따라서 <P> 간의 대응 처리방법이 매

우 중요하다. <P>에서의 변화는 서식의 변화와 텍스트의 변화로 나눌 수 있는데, 서식정보는 ParaShape과 Style 속성을 통해, 그리고 <P>에 속한 <TEXT>의 글자<CHAR>의 서식정보는 CharShape 속성을 통해 표현되는데, 이들 속성들은 각각 서식 정보를 나타내는 엘리먼트들을 참조한다. <TEXT>는 사용자가 문단 내의 텍스트 중 일부 내용에 대해 서식을 변경시키면 이 부분은 기존 텍스트에서 분리되어 다른 <TEXT> 엘리먼트로 표현된다. <그림 2>의 H[0]/S[0]/P[1]/T[1]는 이러한 경우로 '문서의 '를 이탤릭으로 변경한 예이다.

<TABLE>과 같은 구조적인 객체는 트리구조의 엘리먼트들과 이들 엘리먼트들에 속한 속성들을 통해 그 객체의 구조를 나타낸다. <그림 4>는 <그림 3>의 원본문서2의 테이블에 대한 XML 표현이다. <그림 4>에서처럼 <TABLE>은 <P>의 자식노드인 <TEXT>에 속해있는데 <ROW>와 <CELL>을 통해 테이블 구조를 표현한다. 또한 <TABLE>은 ColCount(열의 수)와 RowCount(행의 수)를 포함하여 여섯 개의 속성들로, <CELL>은 ColAddr(열의 위치) 등을 포함해서 열두 개의 속성들로 구성된다. <CELL>의 하위 엘리먼트로 <PARALIST>는 테이블의 셀에 입력될 문단 리스트를 나타낸다.



<그림 3> 원본문서2

앞에서 설명했듯이 혼글에서는 서식 정보의 비중이 매우 크고 <P> 엘리먼트가 매우 빈번하게 발생하며 테이블과 같은 복잡한 트리 구조의 엘리먼트들이 존재한다. 이러한 특징들은 기존의 X-treeDiff+를 통한 변화탐지 작업 시 일반적인 XML 문서의 변화탐지에서는 없었던 문제점들을 발생시킨다. 이들을 정리하면 다음과 같다.

(1) 내용기반의 대응이 이루어지지 못함: 두 문서의 대응과정에서 텍스트 콘텐츠의 변화와 서식변화가 동일하게 처리되므로 생성되는 편집연산들이 직관적으로 이해하기 쉽지 않다. 이는 일반적으로 문서 작업 시, 대개 텍스트를 먼저 변경하고 이에 대한 서식을

```
<P ParaShape="1" Style="0">
  <TEXT CharShape="1">
    <TABLE BorderFill="2" CellSpacing="0" ColCount="2" PageBreak="
      <SHAPEOBJECT InstId="1518098851"/>
      <ROW>
        <CELL BorderFill="2" ColAddr="0" ColSpan="1" Dirty="false"
          <PARALIST LineWrap="Break" LinkListID="0" LinkListIDNext
            <P ParaShape="2" Style="0">
              <TEXT CharShape="1">
                <CHAR>1-1</CHAR>
              </TEXT>
            </P>
          </PARALIST>
        </CELL>
        <CELL BorderFill="2" ColAddr="1" ColSpan="1" Dirty="false"
          <PARALIST LineWrap="Break" LinkListID="0" LinkListIDNext
            <P ParaShape="1" Style="0">
              <TEXT CharShape="1">
                <CHAR>C언어</CHAR>
              </TEXT>
            </P>
            <P ParaShape="1" Style="0">
              <TEXT CharShape="1">
                <CHAR>VB입문</CHAR>
              </TEXT>
            </P>
          </PARALIST>
        </CELL>
      </ROW>
    </TABLE>
```

<그림 4> XML로 표현된 원본문서2의 테이블 내용

변경하나, X-treeDiff+ 등 대부분의 변화탐지 알고리즘들은 텍스트 콘텐츠의 변화와 서식 변화를 구별하지 않고 대응과정에 동시에 고려하기 때문이다.

(2) <P>에 대한 대응이 비효율적임: <P>는 텍스트 이외에 테이블, 차트 등의 다양한 객체를 포함하여 <P>의 발생빈도는 매우 높다. 또한 대부분의 <P>의 서브트리 구조들이 유사하여 <P>에 대한 대응 처리가 전체적인 대응의 효율성에 끼치는 영향은 매우 크다. 기존의 X-treeDiff+에서 이에 대한 처리는 대응을 트리의 아래쪽으로 확산시키는 3단계에서 이루어지는데 <P>의 상위 엘리먼트에서 <P>로의 확산 시 tMD, nMD와 자식 수, nMD, Value와 자식수, Value 등의 정보를 사용해서 대응을 결정한다. 이때 대부분의 <P>는 Value 값, 즉 엘리먼트 명이 동일하기 때문에 비효율적인 대응이 생성되곤 한다. 특히 대부분의 문서에는 문단 내에 동일한 내용 특히 빈 문자열이 포함되는 경우가 빈번한데 이때 기존의 대응 알고리즘은 문단 전체에 대한 문맥을 고려하지 않고 대응을 처리하기 때문이다.

(3) 복잡한 구조의 엘리먼트에 대한 대응의 비효율성: <TABLE>과 같이 복잡한 구조의 엘리먼트의 경우, 약간의 서식 변경이 대응을 왜곡시켜 효율적이지 않은 대응을 생성하곤 한다.

3.2 X-treeDiff+ 기반의 효과적인 혼글 문서의 변화 탐지 기법

본 절에서는 혼글 문서의 효과적인 변화탐지를 위해 불필요한 서식 정보를 배제한 내용기반의 대응 기법을 소개한다. 이를 위해 cMD와 tcMD 개념들을 제안하고 이에 기반한 X-treeDiff+의 개선 방법을 설명한다.

3.2.1 cMD와 tcMD 개념

앞 절에서 소개한 문제점들은 부분적으로 대응과정에 텍스트 내용과 서식정보가 사용되고 있는데 기인한다. 따라서 이의 해결을 위해서는 반드시 필요한 서식정보를 포함한 내용 기반의 대응 기법이 필요하다. 이를 위해 core_attributes, cMD와 tcMD 개념들을 제안한다. 이들 개념들의 정의는 다음과 같다.

정의1. core_attributes는 XML 문서에 속한 엘리먼트들의 속성들 중에서 단순 서식 정보가 아니라 내용 또는 구조 정보를 담고 있어 대응과정에 노드의 대응 여부의 결정에 필요한 속성들의 리스트로 정의한다.

본 논문에서 core_attributes의 예로는 텍스트를 다루는 <P> 관련 엘리먼트들과 <TABLE> 관련 엘리먼트들에 대한 속성들만 제시한다. 텍스트를 다루는 문단의 경우에는 모든 속성들이 서식 정보를 나타내고 있어 본 논문에서는 테이블 구조에 관한 속성들을 core_attributes로 정하는데 이들은 <TABLE>의 RowCount와 ColCount, <CELLZONELIST>의 Count, <CELL>의 ColAddr, RowAddr, ColSpan, RowSpan과 Header로 구성된다. 물론 core_attributes에 설정할 속성들은 사용 목적에 따라 다양하게 결정될 수 있다.

정의2. cMD는 텍스트 노드의 경우에는 노드의 텍스트의 해시 값으로 정의하고, 엘리먼트 노드의 경우에는 엘리먼트 명과 core_attributes에 포함된 속성 및 속성 값을 문자열 통합한 결과 문자열에 대한 해시 값으로 정의한다.

정의3. 텍스트 노드에 대한 tcMD는 그 노드의 cMD로 정의하고, 엘리먼트 노드의 tcMD는 자신 노드의 cMD, 모든 자식 노드들에 대한 tcMD들에 대한 문자열 통합한 결과 문자열에 대한 해시 값으로 정의한다.

위의 cMD개념은 nMD와 유사하며 tMD가 nMD를 기반으로 계산되는 것처럼 tcMD는 cMD를 기초로 정의된다. 단, cMD와 tcMD는 해시 값 계산에 텍스트 노드의 텍스트, 엘리먼트 명과 core_attributes에 속한 속성들만을 포함하게 된다. cMD와 tcMD 기반의 대응은 텍스트 노드와 엘리먼트 노드들을 중심으로 대응과정을 진행하며 이때 엘리먼트의 속성들 중 core_attributes에 속한 속성들만 고려하는데, 이때 대응과정에서 주요하지 않은 서식정보를 배제한다는 의미에서 '내용(contents)' 기반의 대응이라 한다.

3.2.2 X-treeDiff+의 수정사항

먼저 내용 기반의 대응에 필요한 X-treeDiff+의 알고리즘에 대한 수정 사항을 설명한다. X-tree의 노드에 cMD필드와 tcMD필드가 추가하고 X-tree의 생성 시 각 노드의 cMD와 tcMD값을 계산 후 저장하며, 대응 과정의 각 단계의 세부 알고리즘에서 nMD와 tMD 대신 cMD와 tcMD를 사용한다. 이를 다음에서 단계별로 설명한다.

기존의 대응 1단계에서는 원본문서의 모든 노드(엘리먼트와 텍스트)들과 결과문서의 모든 노드들에 대해 동일한 tMD 값을 갖는 노드의 쌍들 중 1-1 대응이 되는 노드 쌍(즉 두 서브트리의 루트 노드의 쌍)을 구했다. 이 과정에서 각 문서에 대해 유일한 tMD 값의 노드를 구하기 위해 노드의 tMD 값과 주소를 엔트리로 하는 해시 테이블들을 사용하는데 이들 테이블의 엔트리들에 대해 순차 방문을 하면서 1-1 대응의 노드 쌍을 구해 이들을 대응시켰다. 본 논문의 제안 방법에서는 tMD 대신 tcMD를 기준으로 동일한 해시 테이블들을 사용하여 1-1 대응되는 노드의 쌍(두 서브트리의 루트 노드의 쌍)을 구한다. 이때 해시테이블의 순차탐색이 아니라 원본 문서를 깊이우선탐색 순으로 탐색하면서 1-1 대응된 노드의 쌍을 구한다. 해시테이블의 순차 방문이 아니라 원본문서를 깊이우선으로 방문하면서 처리하는 이유는 가장 큰 단위의 서브트리의 1-1 대응을 우선 구하고자함에 있다. 이때 발견된 1-1 대응된 모든 노드들의 쌍(서브트리의 루트뿐 아니라 자식 노드들을 포함)을 해시테이블들에서 제거하면, 추가적인 1-1 대응이 가능한 노드 쌍을 구할 수 있음에 착안하여 이 과정을 반복한다. 더 이상 추가적인 1-1 대응의 노드 쌍이 발견되지 않으면 종료한다.

2단계는 기존과 동일하다. 3단계에서는 원본 X-tree를 깊이우선탐색 순서로 접근하면서 대응이 이루어진 노드의 자식 노드들 중 아직 대응이 결정되지 않은

자식 노드들에 대해 대응 시도를 한다. 현재 방문 중인 대응된 노드를 n, 그리고 n의 대응 노드를 M_node(n)으로 나타낼 때, n의 자식노드들 중 미대응(대응되지 않은) 노드들과 M_node(n)의 자식노드들 중 미대응 노드들에 대해 우선 tcMD값이 같은 쌍을 찾아 대응시키고, 나머지 미대응 노드들에 cMD 값이 동일하면서 서브트리의 노드 수도 동일한 노드의 쌍을 찾아 대응시킨다. 끝으로 Value값이 같으면서 자손 노드들의 수가 동일한 노드의 쌍을 대응시킨다. 기존의 3단계에서는 tMD, nMD, Value의 값만으로 비교했으나 이제는 구조적인 비교를 수행하여 대응의 효율을 높인다. 4단계는 대응의 튜닝 단계로 5단계는 대응되지 않은 나머지들에 대한 노드들의 처리로 기존과 동일하다.

편집스크립트 생성은 cMD와 tcMD를 기반으로 생성된 대응을 기반으로 편집스크립트를 생성하는데 이때 두 가지 방법이 가능하다. 우선 대응된 노드들에 대해 텍스트 컨텐츠와 core_attributes의 차이에 대해서 편집스크립트를 생성할 수 있는데 이를 '주요 속성 기반의 편집스크립트'라고 하고 대응된 모든 노드들에 대해 다시 nMD 값의 차이의 여부를 분석해서 텍스트 컨텐츠와 모든 속성들에 대해 편집스크립트를 생성할 수 있다. 이를 '모든 속성 기반의 편집스크립트'라고 한다.

4. 실험 결과 분석

본 절에서는 내용 기반의 대응 기법으로 수정된 X-treeDiff+의 실험 결과를 제시한다. 대응 기법의 성능 비교는 생성된 편집스크립트가 적정한가를 평가해야 하는데 이는 쉬운 작업이 아니다. 최근 XML문서의 변화탐지 알고리즘 분석에 편집연산의 수가 평가기준으로 사용되었는데[1,16], 본 논문에서도 이를 평가지표로 하여 실험한다. X-treeDiff+에 기존 대응 알고리즘과 내용 기반의 대응 알고리즘을 둘 다 구현하여 최근 논문 문서 15쌍에 대해 실험하였다. 이때 기존의 대응 알고리즘에 의한 편집스크립트와 제안된 방법에 의한 편집스크립트를 비교하기 위해 기존 방법에 의한 편집스크립트에서도 core_attributes 속성들만을 포함한 편집연산 수와 모든 속성들을 포함한 편집연산 수를 계산해서 제안된 방법의 결과와 비교한다.

<표 1>에서 문서의 파일 크기는 한글 문서를 XML 형식으로 저장할 때 생성되는 .xml 파일과 .xsl 파일의 크기를 kb단위로 표시한다. 대개 그림 파일 등을 따로 저장하기 때문에 한글 문서의 크기보다는 대부분 작다. 가령 1번 실험의 원본의 한글파일은 548kb이나 xml 파일은 32kb, xsl 파일은 89kb로 구성된다. 실험 대상인 30개 파일들은 81kb - 1,837kb의 크기의 문서들이다.

제안된 알고리즘의 대응 성능의 개선 효과는<표 1>의 편집연산 수의 증감율을 통해 제시된다. 주요 속성 기반의 편집스크립트의 경우 15번째 쌍의 경우를 제외하고는 모든 경우에서 편집연산 수가 감소했고 10번째와 11번째 쌍의 경우에는 -74%, -80%의 감소효과가 있다. 모든 속성 기반의 편집스크립트의 경우는 다소 편차가 있으나 세 경우를 제외하고는 모두 개선효과가 있다.

<표 1> 실험결과

No	파일 크기 (xml, xsl)		편집 연산 수					
			주요 속성 기반			모든 속성 기반		
	원본	결과	기존	제안	증감율	기존	제안	증감율
1	39, 89	22, 91	126	122	-3%	310	302	-3%
2	39, 89	19, 140	185	148	-20%	312	270	-13%
3	22, 91	32, 96	142	124	-13%	152	139	-9%
4	46, 222	43, 192	137	127	-7%	412	408	-1%
5	46, 222	58, 133	277	224	-19%	430	370	-14%
6	43, 192	44, 197	142	112	-21%	200	225	+12%
7	85, 133	96, 178	276	105	-62%	662	291	-56%
8	79, 201	96, 178	356	279	-24%	685	578	-16%
9	46, 77	52, 118	523	259	-50%	919	351	-62%
10	746,90	93, 334	722	188	-74%	1258	358	-72%
11	746,93	143,130	682	132	-80%	1394	199	-86%
12	60, 93	68, 133	668	343	-49%	1242	521	-58%
13	72, 106	89, 157	821	498	-39%	1514	726	-52%
14	204,188	184, 208	999	829	-17%	1442	1488	+3%
15	81, 94	72, 106	891	968	+9%	1688	1714	+2%

몇몇 주목할 만한 경우를 자세히 분석해보면 6번 실험의 경우 주요 속성기반의 편집연산 수는 감소했으나 모든 속성 기반의 편집연산 수는 오히려 증가했다. 이는 원본 문서에 없던 상당량의 수식들이 결과 문서에서 추가됨에 따라 수식의 서식관련 속성들이

삽입되어 발생한 현상이다. 10, 11번의 실험에서는 개선효과가 매우 현저한데 이는 3단계의 대응에서 <P>의 서브트리의 구조를 반영함에 따라 다수의 이동, 복사, 삽입 연산들이 한 번의 삽입연산으로 통합됨에 따른 결과이다. 15번 실험에서는 오히려 편집연산 수가 약간 증가했었는데 이 경우는 완전히 다른 두 문서를 비교함에 따라 모든 내용을 삽입/삭제하는 과정에서 발생한 현상이다.

제안된 알고리즘이 모든 경우를 완벽하게 개선하는 효과가 있는 것은 아니나 실험 결과에서 보듯이 상당한 개선효과가 있음을 알 수 있다.

5. 결 론

본 논문에서는 한글 문서에 대한 변화탐지를 위해 내용 기반의 대응 알고리즘을 제안하고 이를 X-treeDiff+에 구현하였다. 한글 문서는 서식 정보의 비중이 매우 커서 기존의 XML 문서의 변화탐지 알고리즘을 사용하는 것은 적절치 않다. 따라서 주요 서식 정보를 core_attributes로 정의하고 이를 기반으로 cMD와 tcMD 개념을 도입하여 내용 기반 대응 알고리즘을 제안하였다. 그리고 빈번히 발생하는 문단(<P>)와 이의 서브트리 구조의 효과적인 대응을 위해 대응 과정의 3단계를 수정한 결과 대부분의 한글 문서의 변화탐지의 성능이 개선되었음을 확인하였다.

참 고 문 헌

[1] S. Ronnau, J. Scheffczyk, and U. Borghoff, "Towards XML Version Control of Office Documents," In Proc. of ACE Symposium on Document Engineering, pp. 10-19, Nov. 2005.

[2] R. Wagner and M. Fischer, "The string-to-string correction problem," *Journal of the ACM*, 21, pp.168-173, 1974.

[3] K. Tai, "The tree-to-tree correction problem," Journal of the ACM, 26(3), pp.422-433, July 1979.

[4] S. Selkow, "The tree-to-tree editing problem," Information Processing Letters, 6, 1977.

[5] E. W. Myers, "An O(ND) Difference Algorithm and Its Variations," Algorithmica, 1(2), pp.251-266, 1986.

[6] S. Chawathe and H. Molina, "Meaningful Change Detection in Structured Data," In SIGMOD '97, pp.26-37, 1997.

[7] G. Cobéna, S. Abiteboul and A. Marian, "Detecting Changes in XML Documents," The 18th ICDE, 2002.

[8] K. Zhang and D. Shasha, "Simple fast algorithms for the editing distance between trees and related problems," SIAM Journal of Computing, 18(6), pp.1245-1262, 1989.

[9] S. Chawathe and H. G. Molina, "Meaningful Change Detection in Structured Data," In SIGMOD '97, pp.26-37, 1997.

[10] S. Lee and D. Kim, "X-treeDiff+: Efficient Change Detection Algorithm in XML Documents," LNCS 4096, pp.1037-1046, 2006.

[11] Y. Wang, D. DeWitt, J. Cai, "X-Diff: An Effective Change Detection Algorithm for XML Documents," in Proc. of ICDE, pp519-530, Mar., 2003.

[12] S. Chawathe, "Comparing Hierarchical Data in External Memory," Proc. of VLDB, Sept. 1999.

[13] diffxml, http://diffxml.sourceforge.net/

[14] R. Fontaine, "Change Control for XML:Do it right," In Proc. of XML Europe 2003.

[15] DeltaXML, http://www.deltaxml.com

[16] 김동아, "XML 문서에 대한 변화 탐지 및 관리," 단국대학교 전산통계학과 박사학위논문, pp.1-111, 2005.

[17] 한글과 컴퓨터, "한글 문서 파일 구조," http://www. hancom.co.kr/

[18] 한글과 컴퓨터, HWPML의 DTD, http://www. hancom.co.kr/

이 석 균 (Suk Kyoon Lee)

- 정회원
- 1982년 서울대 경제학과 졸업
- 1998년 U of Iowa, Computer Science, MS
- 1993년 U of Iowa, Computer Science, Ph.D
- 1997년 - 단국대학교 컴퓨터학부 교수

논 문 접 수 일 : 2010년 10월 31일
1차수정완료일 : 2010년 11월 20일
게 재 확 정 일 : 2010년 11월 29일