TVA 메타데이터 전송과정에서 단편화에 의한 성능 감소를 회피하기 위한 효율적인 캡슐화 방식

오 봉 진◆, 박 종 열*, 김 상 형**, 유 관 종°

An Efficient Scheme of Encapsulation Method to Avoid Fragmentation Degradation During TVA Metadata Delivery

Bong-Jin Oh◆, Jong-youl Park*, Sang-Hyung Kim**, Kwan-Jong Yoo°

요 약

최근에 XML은 가독성과 확장성이란 장점 때문에 IPTV나 디지털방송 서비스 분야에서 서비스나 콘텐츠 정보 표현 및 검색을 위한 기술로 많이 사용되고 있다. 특히 TV-Anytime에서 정의한 스키마나 전송 프로토콜을 기본 규격으로 채택하고 필요한 기능을 수용하기 위하여 규격을 확장하는 경향을 보이고 있다. 그러한 장점에도 불구하고 XML은 텍스트 기반으로 정보를 표현함으로써 정보의 양이 커진다는 문제가 존재하고 이를 위하여 다양한 인코딩 방식이 제공되고 있다. 그러나 전송과정에서 문서를 독립된 조각으로 단편화하여 블록단위로 캡슐화 하는 과정을 거치면서 인코딩 효율이 급격하게 떨어지게 되는 문제가 발생한다. 본 논문에서는 XML 문서를 캡슐화 하는 과정에서 단편화를 통해 감소되는 인코딩 효율을 보완할 수 있는 블록 단위의 문자열 테이블 방식을 제안하고 실험을 통한 성능 분석을 제공하였다.

Key Words : IPTV Contents Guide, TVA Encapsulation, String Table, IPTV 콘텐츠 가이드, TVA 캡슐화, 문자열 테이블

ABSTRACT

Recently, XML is used to describe details of service and contents for various fields such as IPTV and digital broadcast services because of it's high readability and extensibility. TV-Anytime's schema and delivery protocol have been especially adopted as basic standards for them, and extended to include their own private functions. However, XML describes documents using text-based method, and this causes to create big documents rather than traditional methods. Therefore, many encoding algorithms have been proposed to reduce XML documents like EXI, BiM, GZIP and fast-info set etc. Although these algorithms shows efficient compression effects for XML documents, but they can't avoid fragmentation degradation during encapsulation steep. This paper proposes an efficient encapsulation scheme of TV-Anytime to avoid fragmentation degradation of encoding effect using common string tables.

Ⅰ. 서 론

XML (Extended Markup Language)은 텍스트

※ 본 연구는 지식경제부의 IT R&D 프로그램의 일환인 "N-스크린 서비스를 위한 SmartTV용 협업 미들웨어 및 RUI 기술개발" 과제(과제번호:KI0039202)의 지원으로 진행되었습니다.
◆ 주저자 : 한국전자통신연구원 BigData소프트웨어연구소 소셜컴퓨팅연구팀, bjoh@etri.re.kr, 정회원
° 교신저자 : 충남대학교 컴퓨터공학과 멀티미디어정보통신연구실, kjyoo@cnu.ac.kr, 정회원
* 한국전자통신연구원 BigData소프트웨어연구소 소셜컴퓨팅연구팀, jypark@etri.re.kr
** 충남대학교 컴퓨터공학과 멀티미디어정보통신연구실, kims@cnu.ac.kr
논문번호 : KICS2011-07-314, 접수일자 : 2011년 7월 20일, 최종논문접수일자 : 2012년 6월 14일

기반으로 정보를 기술하는 언어로 높은 가독성과 확장성으로 인해 다양한 영역에서 서비스나 콘텐츠 정보를 기술하기 위하여 채택되고 있다. XML은 정보 표현을 위해 스키마라는 문법체계를 통해 실제 문서의 기술 방법을 정의한 후 그에 따라 문서를 작성하는 방식이다[1].

TV-Anytime (TVA)은 XML을 PVR (Personal Video Recoder)이나 디지털방송 서비스에 적용하기 위해 서비스, 콘텐츠 정보 표현용 스키마와 전송 프로토콜을 정의하였고, 많은 방송 규격에서 참조 문서로 활용하고 있다[2,3].

국내에서도 지상파, 케이블 방송이나 위성방송에서는 전송매체의 단방향 특성으로 인하여 서비스 정보를 영상과 함께 다중화 하여 송출하고 단말은 다중화 된 정보를 추출하여 수신하는 구조로 되어 있다. 또한 정보의 기술이 대역폭 절감을 위하여 실시간방송 채널과 송출되는 프로그램 정보를 표현하기 위한 범위로 국한되어 있으며, DVB-SI나 PSIP 등의 이진화 된 테이블 방식의 규격을 이용하여 서비스가 이루어지고 있다. 그러나 2003년 이후 IPTV, 스마트TV 등과 더불어 RF 튜너와 IP가 결합된 HbbTV 등의 등장으로 VOD (Video On Demand)나 데이터방송 서비스와 같은 양방향 특성을 가진 부가서비스의 제공이 일반화 되었다. 기존의 테이블 방식의 규격은 실시간방송을 위한 엄격한 구조를 가지고 있어 다양한 콘텐츠 그리고 양방향성이라는 특징을 지원함에 있어 많은 제약을 가지고 있다[4,5].

따라서, 최근에는 TV-Anytime을 기반으로 다양한 서비스와 콘텐츠 정보를 표현하고 전달하는 규격이 진행되고 시범적으로 운용되고 있으며, 국내의 IPTV와 유럽의 DVB-IPI, 북미의 ATIS EPG-Metadata 그리고 OIPF 등의 그 예이다. 최근 국내의 HbbTV 규격에서도 OIPF에서 정의한 메타데이터 검색 규격이 참조 규격으로 채택되어 향후 TV 매체를 통해 제공되는 서비스와 콘텐츠의 검색은 XML 기반의 프로토콜에 의해 이루어질 것으로 예상된다.[6-8].

서비스나 콘텐츠 상세 정보를 XML로 표현한 것을 TVA에서는 디스크립션이라고 한다. 생성된 TVA 디스크립션은 사용자가 서비스나 콘텐츠 수신 및 재생을 위하여 최초로 참조하는 정보로 그림 1과 같은 단계로 전달된다.

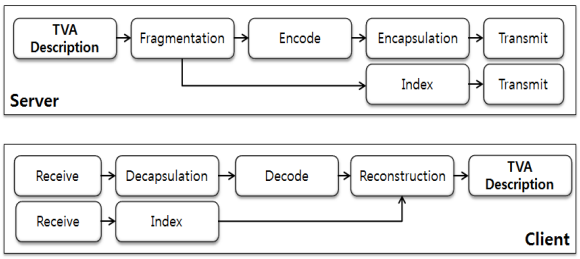

그림 1. TVA 디스크립션 전송 프로토콜
Fig. 1. TVA description delivery protocol

TVA 디스크립션은 우선 하나의 독립된 서브 디스크립션 (프래그먼트라고 함) 단위로 분리가 되어 선택적으로 문서의 크기를 줄이기 위하여 인코딩 알고리즘을 통해 이진화 한 후 컨테이너라는 블록 단위로 구성하여 송출한다. 각 프래그먼트가 어떤 컨테이너에 구성되어 있는지 정보를 색인화 하여 색인 정보를 별도의 컨테이너로 전송한다. 단말은 인덱스 컨테이너를 수신한 후 프래그먼트의 구성 정보를 확인하고 필요한 프래그먼트를 포함한 컨테이너를 수신하여 프래그먼트를 추출한다.[2,3].

인코딩 시 사용되는 알고리즘은 Gnu-ZIP (GZIP), BiM (Binary Format of Metadata), EXI (Efficient XML Interchange) 그리고 Fast-Info Set 등 원리에 따라 다양하게 제시되고 있다. 각 알고리즘들은 고유 메커니즘을 이용하여 문서의 크기를 줄이고 있는데 문자열 테이블을 이용하는 알고리즘이 좋은 효과를 보이고 있다. 이는 XML 문서가 텍스트 기반으로 기술되어 문서를 이루는 구성 요소들이 문자열 값을 갖는 경우가 많기 때문이다. 문자열 테이블은 문서에 나타나는 중복된 문자열을 인덱스로 대체함으로써 인코딩 효과를 얻으며 평균 문자열 히트가 높을수록 좋은 결과가 발생한다. 본 논문에서는 전역 문자열 테이블을 이용하여 같은 데이터 컨테이너에 포함되는 문서들의 평균 문자열 히트를 높여 최종적인 인코딩 효율을 높일 수 있는 TVA 캡슐화 구조를 제시한다.

본 논문의 구성은 2장에서 제안되는 구조에 대해 자세하게 설명하고, 3장에서 성능 평가를 하며 마지막으로 4장에서 결론을 내린다.

Ⅱ. 전역 문자열 테이블을 이용한 캡슐화 구조

2.1. 단편화에 의한 성능 감소

TVA 디스크립션의 전송 과정에서 디스크립션을 단편화 하는 경우 인코딩의 단위가 전체 디스크립

션에서 단편화 된 서브 디스크립션 단위로 축소되어 인코딩 효율이 떨어지게 된다. GZIP의 경우는 바이트열의 패턴을 분석하여 긴 패턴이 반복될수록 성능이 높아지는데 문서의 크기가 작은 경우 그 효율은 급격히 감소한다. EXI나 Fast-info set과 같이 문자열 테이블을 이용하여 인코딩을 하는 경우 역시 반복되는 문자열을 테이블의 색인으로 대체하는 방법과 deflate 방법을 결합하여 효과를 극대화한다. 같은 패턴의 반복은 문서의 크기와 밀접한 관계가 있음을 그림 2로부터 이해할 수 있다. 실험에 사용된 문서는 BBC가 2010년에 시험 송출한 채널들에 대한 ProgramInformation(PI) 디스크립션이다.

size	1	2	4	9	16	32	61
GZIP	0.32	0.76	0.77	1.75	2.03	4.07	3.62
EXI	0.11	0.52	0.56	1.53	1.76	3.63	3.09

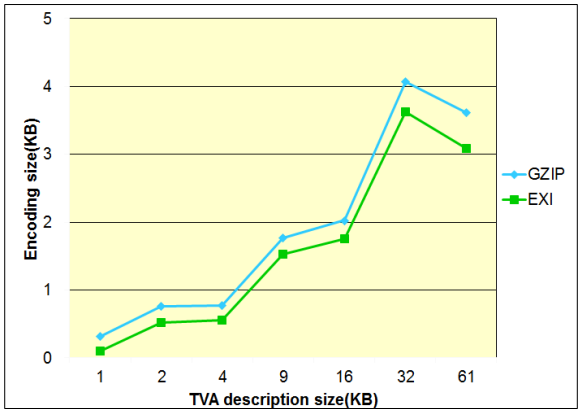

그림 2. XML 문서 크기에 따른 인코딩 효율
Fig. 2. Encoding size according to original XML size

2KB의 XML 문서에 대해 GZIP의 경우 1/3, EXI의 경우 1/4 정도로 인코딩 결과가 나오고, 61KB 문서에 대해 GZIP은 1/18, EXI의 경우는 약 1/20 정도의 인코딩 효율을 보여주는 것을 알 수 있다. 따라서 문서의 크기가 큰 경우 작은 문서에 비해 인코딩 효율이 분명 증가함을 알 수 있다.

그림 3은 문서 크기에 따른 문자열 히트 즉, 동일한 문자열이 나오는 횟수를 EXI 알고리즘으로 실험한 결과를 비교한 그래프이다. 또한 각 문서들을 독립된 서브 문서로 단편화 하여 인코딩하였을 때 발생된 문자열 히트 수도 측정하였다. 문서의 크기에 따라 발생되는 문자열 히트 수가 급격히 늘어나는 것을 알 수 있었고, 이는 문서의 크기가 커지면 문자열 중복의 기회가 많아지기 때문에 EXI와 같은 문자열 테이블 기반의 알고리즘에서 인코딩 효율에 긍정적인 영향을 미친다고 판단할 수 있다. 반면에 같은 문서를 단편화 한 경우 각 프래그먼트별로 EXI 알고리즘에 의해 인코딩되는 과정에서, 전체 문서 단위로 인코딩한 경우에 발생되던 문자열 히트 수를 잃어버려 거의 문자열 히트가 0에 가까움을 보여주고 있다. 이런 현상은 궁극적으로 문자열 기반의 인코딩 시스템에서 단편화 되지 않은 경우에 비해 문서 압축률을 떨어뜨리는 이유가 된다.

size	1	2	4	9	16	32	61
디스크립션	1	11	23	48	131	229	659
단편화	0	0	0	1	1	0	1

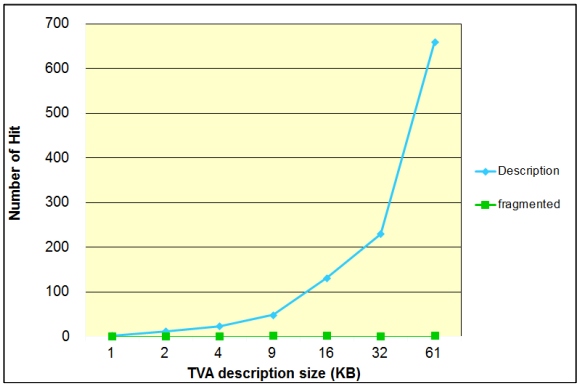

그림 3. XML 문서 크기에 따른 문자열 히트 수
Fig. 3. String hit number according to XML size

그림 4는 BBC에서 시범서비스로 송출하는 콘텐츠 상세 정보 기술을 위한 PI 프래그먼트의 예제이다. 하나의 PI 디스크립션에서 단편화된 2개의 프래그먼트를 발췌한 것으로 상위의 프래그먼트에서 빨간 박스로 표시된 7개의 문자열은 하위 박스에서 동일하게 반복하여 나타남을 알 수 있다. 따라서 개별적으로 인코딩하게 되면 문자열 히트가 각각 0이나 두 문서 사이의 연관성을 고려한다면 각 7개의 히트 수가 나타날 수 있음을 알 수 있다. 따라서 하나의 프래그먼트 단위로 인코딩함으로써 잃어버릴 수 있는 문자열 히트 수를 유지할 수 있는 캡슐화 방법이 존재한다면 전체적으로 인코딩 효율을 향상시킬 수 있고, 이는 XML 기반의 서비스 검색 시스템에는 매우 중요한 요구사항이 될 수 있다.

그림 4. BBC 프로그램 상세 정보 예
Fig. 4. Example of BBC Program Information

본 논문에서는 EXI를 이용하여 전역 문자열 테이블 방식 기반으로 하나의 컨테이너에 포함된 프래그먼트 간 연관성을 이용하여 인코딩 효율을 높일 수 있는 방식을 검증하도록 한다.

2.2. EXI 인코딩 방식

EXI는 스키마를 분석하여 구성요소(엘리먼트, 속성) 등을 노드로 갖는 오토마타를 생성한다. 이 오토마타의 흐름을 이벤트 코드의 연속으로 표현하여 문법 구성 요소는 인코딩한다. 그리고 각 노드에 부여될 수 있는 값 중 문자열 값의 경우는 2개의 문자열 테이블을 기반으로 인코딩한다[7-9].

```
<?xml version="1.0" encoding="UTF-8"?>
<xs: schema xmln:xs="http://www.w3c.org/2001/XMLSchema"
elementFormDefault="qualified">
        <xs:element name="A">
          <xs:complexType>
            <xs:sequence maxOccurs="1">
              <xs:element name="B" type="xs:string" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
</xs:schema>
```

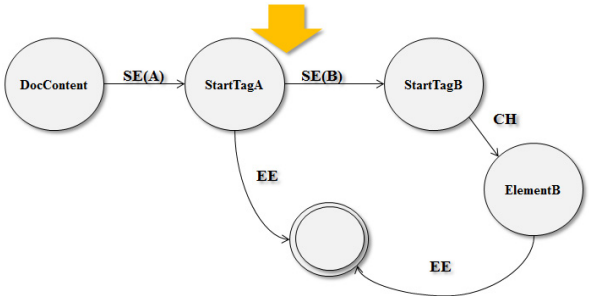

그림 5. EXI 오토마타 생성 예제
Fig. 5. Sample EXI automata generation

그림 5는 EXI의 오토마타를 설명하기 위한 스키마로부터 생성된 오토마타를 보여준다. 각 노드는 목적 노드로 이동할 때 이벤트 코드를 부여 받는데 경우의 수에 따라 이벤트 코드 수가 달라진다.

표 1. EXI 이벤트 코드 예제
Table 1. EXI event code of sample automata

Node	Event	Event code
DocContent	SE(A)	0
StartTagA	SE(B)	0
	EE	1
StartTagB	CH	0
ElementB	EE	0

표 1은 예제 오토마타로부터 각 링크에 부여된 이벤트 코드를 보여준다. 모든 노드가 1-2개의 목적 노드를 갖고 있어 1비트 코드로 표현될 수 있고 StartTagA만 2개의 이벤트 코드를 보유하고 있다.

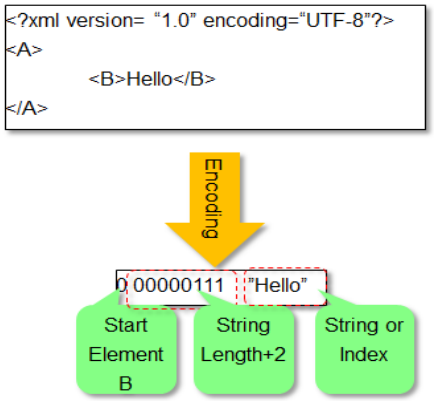

그림 6. 예제 XML문서 인코딩
Fig. 6. Sample XML document encoding

그림 6은 예제 스키마에 맞춰 기술된 XML 문서를 인코딩한 결과를 보여주는데, 하나의 경로만을 갖는 Start Document (SD)나 StartElement(SE) A 등은 이벤트가 발생함에도 불구하고 인코딩 스트림엔 삽입하지 않는다. 그리고 SE (B)의 경우 두 가지 가능한 경우 중 첫 번째로 0이라는 1비트 값이 인코딩 스트림에 삽입된다. 그리고 문자열 값인 "Hello"가 인코딩 되는데 첫 번째 발생한 것이므로 스트림 자체에 삽입이 되고 이를 명시하기 위한 인덱스가 앞에 1바이트 붙는다. 인덱스는 하의 다음에 문자열이 바로 온다는 것으로 문자열의 길이+2만큼 값을 갖는다. 만약 인덱스가 0이나 1이면 2가지 테이블 중 최소한 하나에 들어 있다는 것으로 다음에 오는 값은 그 테이블의 색인 값이 된다.

2.3. 전역 문자열 테이블 메타데이터 캡슐화 방식

TVA 디스크립션은 그림 1과 같은 전송 프로토콜 기반으로 단말로 전송되는 과정에서 독립된 서브 디스크립션인 프래그먼트 단위로 단편화된다. 하나의 프래그먼트는 그 자체만으로 하나의 유효한 TVA 디스크립션으로 인식될 수 있다. 예를 들어 TVA 디스크립션이 한 채널에 대한 하루치의 일정을 한 시간 단위로 나누어 24개의 Schedule 프래그먼트들로 구성될 수 있다. 이때 그림 7과 같이 전송과정에서 독립된 하나의 Schedule 프래그먼트를 순차적으로 추출하여 인코딩한 후 전송 단위인 데이터 컨테이너에 포함 시킨다

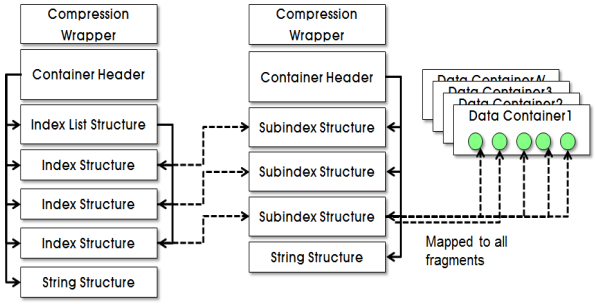

그림 7. 프래그먼트 구성정보를 위한 인덱스 구조
Fig. 7. Index structures for fragment's configuration

그림 7에서 보듯이 컨테이너에는 인코딩된 프래그먼트를 포함하는 데이터 컨테이너와 구성 정보를 분석할 수 있도록 색인을 포함하는 인덱스 컨테이너로 나뉜다[2,3].

단말은 우선 인덱스 컨테이너 중 색인 목록(Index List)과 색인 (Index) 구성 정보를 포함한 첫 번째 컨테이너를 수신하여 어떤 기준으로 데이터가 구성되었는지 확인한다. 그리고 시간, 장르 혹은 특정 채널 등의 기준을 색인으로 하여 데이터를 찾기 위하여 각 기준의 색인과 대응되는 서브인덱스 목록을 포함하는 인덱스 컨테이너를 수신한다. 서브인덱스가 각 프래그먼트의 위치를 포함하는 마지막 색인이 되며 (컨테이너 식별자, 프래그먼트 식별자)로 이루어진다.

서브인덱스를 분석하면 원하는 기준에 부합하는 모든 프래그먼트를 포함하는 데이터 컨테이너 목록을 알 수 있고, 각 데이터 컨테이너를 수신하여 포함된 프래그먼트를 추출하면 TVA 디스크립션을 복구할 수 있다.

현재의 EXI 기반의 인코딩은 데이터 컨테이너에 포함되는 각각의 프래그먼트 단위로 인코딩이 이루어지므로, 같이 캡슐화 되는 프래그먼트 간에 연관성을 전혀 활용하지 못한다. 따라서 문자열 테이블이 1-2KB 단위인 프래그먼트 정도의 범위 내에서 형성된다. 이런 경우는 프래그먼트 간에 중복 가능한 문자열을 색인으로 대체할 기회가 사라져 인코딩 효율에 있어 큰 이득을 볼 수 없다는 것을 판단할 수 있다.

본 논문에서는 그림 8과 같이 전역 테이블을 컨테이너 단위로 구성하여 캡슐화 과정에서 활용하는 방식을 제안한다.

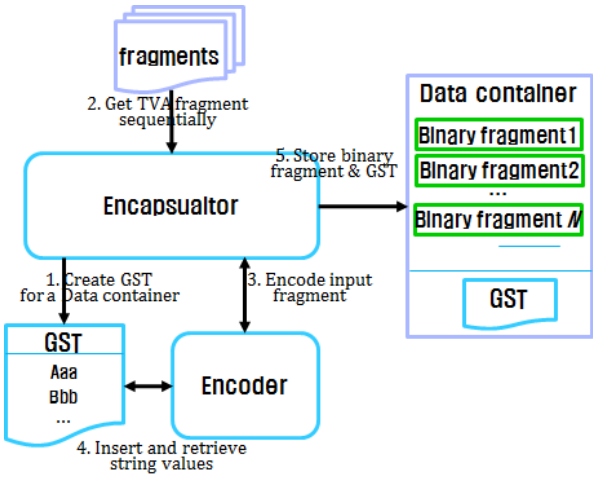

그림 8. 전역 테이블을 이용한 EXI 캡슐화 블록도
Fig. 8. Encapsulation diagram with Global String Tables

TVA 디스크립션로부터 단편화된 프래그먼트는 순차적으로 EXI 인코더로 입력된다. 인코더는 전역 문자열 테이블 방식을 지원하도록 확장되고 컨테이너 단위로 각 프래그먼트를 구성할 때 전역 문자열 테이블을 이용하도록 설정된다.

그림 9는 모든 데이터 컨테이너를 구성하기 위하여 각 프래그먼트를 전역 문자열 테이블 기반으로 인코딩하는 과정을 보여주는 순서도이다.

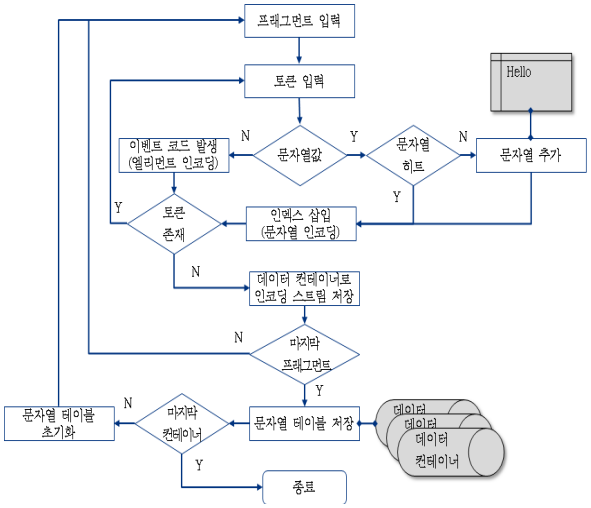

그림 9. 전역 테이블 기반의 메타데이터 인코딩 순서도
Fig. 9. Encoding diagram based on shared string table

데이터 컨테이너를 구성하기 위한 프래그먼트는 EXI 인코더로 입력되고 토큰 단위로 인코딩이 이루어지게 된다. 토큰이 일반 엘리먼트나 속성이면 생성된 오토마타에 따라 이벤트 코드가 발생하고 필요한 경우 인코딩 스트림에 삽입된다. 그리고 토큰이 문자열 값인 경우는 전역 문자열 테이블에 이미 존재하는 지 확인하고, 존재하는 경우 색인을 인코딩 스트림에 삽입한다. 그렇지 않은 경우는 전역 문자열 테이블에 문자열 값을 추가한 후 인코딩 스트림에 삽입한다. 각 프래그먼트 내에 토큰이 존재하지 않을 때까지 계속 위의 과정을 처리한 후 마지막 토큰이 처리되면, 현재 데이터 컨테이너를 위한 프래그먼트가 존재하는 지 여부를 검사한 후 최종 프래그먼트가 처리될 때까지 계속 인코딩 과정을 수행한다. 최종 프래그먼트가 처리되면 문자열 테이블을 데이터 컨테이너에 저장하고 데이터 컨테이너 구성을 완성한다. 그리고 추가의 데이터 컨테이너를 구성할 필요가 있으면 위와 같은 절차를 마지막 데이터 컨테이너까지 계속 진행하고 인코딩 과정은 종료된다.

문자열 테이블은 원래의 EXI와는 달리 각 데이터 컨테이너 별로 전역 테이블 하나만을 사용한다.

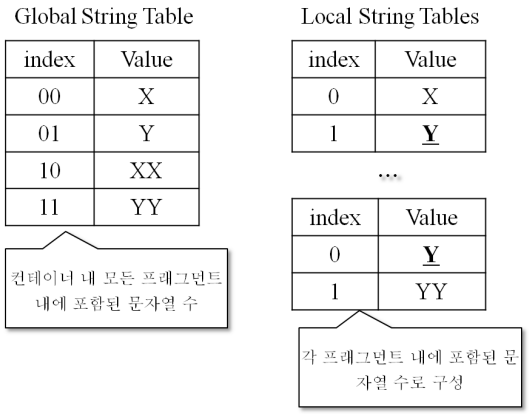

그림 10. 전역테이블과 EXI Table 비교
Fig. 10. Analysis of global string table and EXI tables

그림 10과 같이 원래의 EXI는 각 프래그먼트별로 인코딩된 스트림 내에 포함된 문자열 테이블을 이용한다. 이러한 경우 그림 10의 "Y" 문자열 값과 같이 중복된 값이 서로 다른 프래그먼트 내 테이블에 포함될 수 있다. 따라서 문자열을 구분하기 위한 1바이트와 문자열 길이만큼의 바이트가 더 필요하게 된다(EXI 문자열 인코딩 방식은 그림 6의 예제 참조). 문자열 히트가 발생한 경우 GST 기반의 경우는 전역 문자열 테이블 크기에 해당하는 인덱스 값만 인코딩되지만, EXI의 경우는 인덱스 구별을 위한 1바이트와 프래그먼트의 지역 문자열 테이블 크기에 해당하는 값이 조합된다. 전역 문자열 테이블 크기가 비정상적으로 크지 않은 이상 대부분의

경우 문자열을 인코딩하는 경우 GST 방식이 효율적임을 판단할 수 있다(섹션 4의 성능분석 참조).

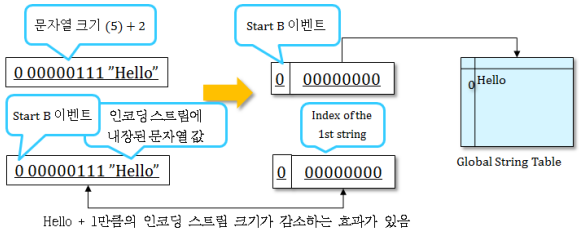

그림 11. 전역 테이블 기반 메타데이터 인코딩 예
Fig. 11. Global string table based encoding example

기존의 EXI 방식으로는 그림 11의 왼쪽과 같이 동일한 인코딩 스트림이 2개의 프래그먼트로부터 생성되어 98비트의 인코딩 스트림 크기를 갖게 된다. 반면에 전역 문자열 테이블 기반의 인코딩 스트림을 보면 SE(B)를 위한 1비트와 전역 문자열 테이블 내의 색인을 위한 1바이트만이 사용된다. 따라서 18비트의 인코딩 스트림과 문자열 테이블 내의 "Hello"를 위한 40비트 총 58비트만이 필요하다. 총 40비트의 절감 효과를 가져 온다고 볼 수 있다.

2.4. 전역 문자열 테이블 인코딩 성능분석

본 논문에서 제안하는 방법은 물리적으로는 분리되어 있으나 논리적으로 하나의 문서로 컨테이너에 프래그먼트를 묶어 줌으로써 문자열 히트수를 높여 인코딩 효율을 높이고자 하는 것이다. 본 섹션에서는 컨테이너 내에 포함된 각 프래그먼트가 기존의 EXI와 제안하는 방식으로 인코딩 되는 경우 개선 효과를 비교한다. 식 1은 EXI 방식을 통해 인코딩 되는 스트림의 크기를 수식으로 표현한 것이다.

$$P(T) = \sum_{i=0}^{|T|} \log_2(L(t)) + \sum_{i=0}^{|T|} encode(Value_i) \ , \qquad (1)$$

여기서, T = {Tn | 0 <= n <= N}, N은 XML 문서를 구성하는 토큰 수,

L(t) = T에 대해 방문되는 노드가 갖는 링크 수,

Valuet =T에 대해 이동되는 엘리먼트, 속성의 값, encode (Valuet) = 값을 유형에 따라 인코딩한 결과임.

따라서, 노드의 이동에 따라 발생하는 $Log_2(L(t))$ 크기의 이벤트와 여러 유형의 값을 인코딩 한 결과를 포함한 것이 최종 인코딩 스트림 크기가 된다.

본 논문에 제시하는 방식은 이벤트 코드와는 관련이 없으므로 우선 모든 *Valuet*를 문자열로만 가정한다면, *encode(Valuet)*가 문자열을 인코딩한 결과가 되고 인코딩 개선 효과를 다음과 같이 비교할 수 있다.

$$Encode(Value_t)\,of\,EXI = \sum_{i=1}^{N_s-h}(L_i+1) + \sum_{j=1}^{h}(\log_2 L_{lst}/8+1) \quad \text{이고,}$$

$$Encode(Value_t)\,of\,GST = \sum_{i=1}^{N_s-h_{gst}} L_i + \sum_{j=1}^{N_s} L_{gst}/8 \ ,$$

여기서, L_{lst} =프래그먼트의 문자열 테이블 크기,

L_{gst} =컨테이너 공유 문자열 테이블(GST) 크기,

N_s = 프래그먼트 내의 문자열 수,

h = 프래그먼트 내의 문자열 히트 수,

h_{gst} = GST 기반에서의 문자열 히트 수

L_i = 문자열의 크기

따라서, 인코딩 개선효과는 아래처럼 표현된다.

$$e_f = \frac{\sum_{i=1}^{N_s-h_{gst}} L + \sum_{j=1}^{N_s}(\log_2 N_{gst}/8)}{\sum_{i=1}^{N_s-h}(L+1) + \sum_{j=1}^{h}(\log_2 N_{lst}/8+1)} \qquad (2)$$

단, L = L_i를 평균으로 환산한 문자열 크기 값임

최선의 경우는 h=0이고 h_{gst}= N_s의 경우이므로 다음과 같이 식 2를 전개할 수 있다.

$$e_f = \frac{\sum_{j=1}^{N_s}(\log_2 N_{gst}/8)}{\sum_{i=1}^{N_s}(L+1)} = \frac{N_s*(\log_2 N_{gst}/8)}{N_s*(L+1)} = \frac{\log_2 N_{gst}}{8(L+1)}$$

N_{gst}보다는 L의 값이 충분히 크다고 가정할 수 있으므로 E_f 《1이고 인코딩 효율 개선이 된 것을 증명한다.

최악의 경우는 $h= h_{gst}$의 경우인데 전역 문자열 테이블 방식에서는 h_{gst}가 h를 포함하는 구조로 h_{gst}보다 h가 더 큰 경우는 존재할 수 없기 때문이다. 이 경우에서 $h= h_{gst} = 0$인 경우는 식 2가 다음과 같이 전개된다.

$$e_f = \frac{\sum_{i=1}^{N_s} L + \sum_{j=1}^{N_s} (\log_2 N_{gst}/8)}{\sum_{i=1}^{N_s} (L+1)} = \frac{N_s * L + N_s * (\log_2 N_{gst}/8)}{N_s * (L+1)} = \frac{L + (\log_2 N_{gst}/8)}{(L+1)} = \frac{8L + \log_2 N_{gst}}{8L+8}$$

따라서, N_{gst} >256의 경우 E_f > 1 이 되어 오히려 효율이 떨어질 수 있음을 보여준다.

만약 $h= h_{gst} = N_s$-1 즉 최고의 인코딩 효과가 발생하는 경우 식 2는 다음과 같이 전개된다. 참고로 $h = N_s$-1의 경우 N_{lst}는 1이 된다. 이는 1개의 문자열이 프래그먼트에서 N_s번 반복되는 경우이다.

$$e_f = \frac{L + \sum_{j=1}^{N_s} (\log_2 N_{gst}/8)}{L+1+(N_s-1)/8+N_s-1} = \frac{8L + N_s * \log_2 N_{gst}}{8L+(9N_s-1)}$$

여기서 L과 N_s가 무한대로 커진다면 1은 무시할 수 있으므로 결국 N_{gst} > 512이면 E_f > 1이 되어 역시 효율이 떨어질 수 있음을 보여준다.

따라서, $h= h_{gst}$인 경우는 전역 문자열 테이블의 크기가 적당한 수준으로 유지되는 경우 제안하는 방식이 효과적임을 최종 판단할 수 있다.

Ⅲ. 전역문자열테이블 인코딩 방식 성능실험

본 장에서는 표 2와 같은 실험환경을 구축하여 제안하는 캡슐화 구조의 인코딩 효율을 측정하였다.

표 2. 실험환경
Table 2. Experimental environments

항목	내용
호스트	▪ 1.6 GHz CPU, 1G RAM, UMPC
이뮬레이터	▪ EXIficient 0.6 기반 이뮬레이터 (JDK 1.5 이상에서 동작)
시험 XML문서	▪ BBC가 2010년 8.31 ~ 9.08까지 제공한 ProgramInformation, ProgramLocation 문서 활용 ▪ 1KB에서 64KB까지 17개 디스크립션 ▪ PI, PL별로 2개씩 4개의 컨테이너 구성
비교 알고리즘	▪ EXI 1.0 (EXIficient 오리지널 알고리즘 사용) ▪ GZIP (JDK GZIP OutputStream 사용)

EXI 알고리즘은 EXIficient 0.6기반 이뮬레이터를 구현하였고, 단편화와 인코딩 및 캡슐화를 진행하고 결과를 보여주도록 하였다. 전역 문자열 테이블 기능을 위하여 EXIficient의 StringEncoder와 StringDecoder 클래스를 확장하였으며 표3과 같은 BBC 시범 서비스를 위한 메타데이터를 사용하였다.

표 3. 실험문서
Table 3. Experimental TVA descriptions

No	Title	type	Size (byte)	컨테이너
1	20100908OneExtra_pi	PI	447	1
2	20100907BBCRThree_pi	PI	1,737	2
3	20100908BBCRFiveX_pi	PI	3,616	2
4	20100903BBCFour_pi	PI	8,623	2
5	20100903BBCRFiveL_pi	PI	15,831	2
6	20100902BBCSeven_pi	PI	31,967	2
7	20100904BBCWrld_pi	PI	61,718	1
8	20100831BBCWrild_pl	PL	29,056	3
9	20100902BBCSeven_pl	PL	17,355	3
10	20100902BBCWrld_pl	PL	29,287	4
11	20100903BBCFour_pl	PL	5,317	3
12	20100903BBCRFiveL_pl	PL	4,946	3
13	20100904BBCWrld_pl	PL	25,706	4
14	2010907BBCRFour_pl	PL	2,427	4
15	2010907BBCRThree_pl	PL	925	3
16	20100908BBCRFiveX_pl	PL	1,931	3
17	20100908OneXtra_pl	PL	445	4

1번부터 7번은 BBC의 각 채널에서 송출되는 프로그램 상세 정보를 기술한 PI 디스크립션을 1KB~61KB 크기별로 선택한 것이다. PI 디스크립션은 크기에 따라 적게는 1개, 많게는 63개의 프래그먼트로 구성되어 있었다.

8번부터 17번은 스케줄 정보를 포함하는 ProgramLocation (PL)로 하나의 프래그먼트만 포함하는 디스크립션의 형태를 갖고 있다. PI와 PL별로 2개씩 컨테이너를 구성하여 총 4개의 컨테이너를 구성, 시험하였으며 표3의 Container에 각 디스크립션이 포함되는 컨테이너 번호를 명시하였다.

4개의 컨테이너를 GZIP, EXI 원래 방식 그리고 GST(전역 문자열 테이블 방식)으로 인코딩한 결과는 그림 12와 같다.

Container	GZIP	EXI	GST
1	39,784	24,881	18,552
2	38,866	26,267	20,125
3	5,784	5,091	4,711
4	4,455	4,079	3,828

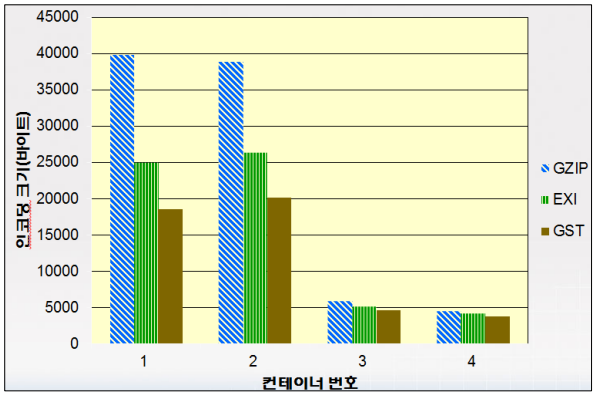

그림 12. 인코딩 효율 비교
Fig. 12. Comparison of encoding effect

그림 12를 보면 제안하는 전역 문자열 테이블 방법에 의하여 4개의 컨테이너를 인코딩한 결과 EXI에 비해 PI가 포함된 컨테이너는 약 25%, PL이 포함된 컨테이너는 8% 이상 개선된 인코딩 성능을 보여줬다. BBC의 PL 디스크립션의 경우 Schedule 1개의 프래그먼트로만 이루어져 기본적으로 PI에 비하여 전역 문자열 테이블의 장점을 많이 활용하지 못하였다. 그러나 보편적으로 3시간 단위로 스케줄이 구성되어 전송되는 방송 현실을 감안하면 PL도 8개의 프래그먼트로 구성될 수 있고, 이는 기존 EXI나 GZIP에서 인코딩 저하를 가져오는 반면에 제안 방식에서는 덜 영향을 받을 수 있다고 추측할 수 있다. 표 4는 17개의 디스크립션이 단편화되었을 경우 프래그먼트 당 평균 히트 수와 전역 문자열 테이블 방식을 사용하였을 경우 프래그먼트 당 문자열 히트 수를 비교한 것이다. 대부분의 경우 전역 문자열 테이블 방식을 사용한 경우 히트 수가 늘어남을 알 수 있고, 히트 수의 증가가 인코딩 효율 개선에 긍정적인 영향을 미친 것으로 판단된다.

표 4. 문자열 히트 수 비교
Table 4. String hit number analysis

No	프래그먼트수	EXI 평균 히트	GST 평균 히트 (컨테이너)	EXI 평균크기	GST 평균크기
1	1	0	0(1)	106	106
2	1	0	7(2)	352	265
3	4	0	6.5(2)	395.5	236.8
4	8	0.13	6.9(2)	480.125	292
5	13	0.07	9.2(2)	479.9	268.5
6	29	0	8.4(2)	485.6	262.5
7	62	0.02	10.0(1)	399.6	262.1
8	1	88	88 (3)	1819	810
9	1	51	53 (3)	1223	536
10	1	90	90(4)	1815	816
11	1	6	9(3)	723	347
12	1	9	18(3)	580	309
13	1	68	82(4)	1649	672
14	1	1	3(4)	490	277
15	1	1	2(3)	328	209
16	1	1	4(3)	418	244
17	1	0	1(4)	125	103

Ⅳ. 결 론

본 논문에서 제안하는 전역 문자열 테이블을 이용하면 문자열 테이블 기반으로 인코딩하는 모델에 대해서 충분한 성능개선 효과를 기대할 수 있음을 보여줬다. 디스크립션이 종류에 따라 그리고 구성되는 프래그먼트의 숫자에 따라 성능에 차이는 있었으나 BBC PI, PL의 경우 8%~25%까지 개선된 성능을 보여줬다. 특히 문자열 히트수가 기존 방식에 비하여 증가하였고 이는 인코딩 효율 개선에 절대적인 기여를 한 것으로 판단된다.

References

[1] W3C, *Extensible Markup Language (XML) 1.0 5th Ed*, W3C Recommendation, 2006.

[2] ETSI, Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 3: Metadata; Sub-part 1: Phase 1- Metadata schemas, *ETSI TS 102 822-3-1 V1.5.1*, 2009.

[3] ETSI, Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Sub-part 2: System aspects in a uni-directional environment, *ETSI TS 102-822-3-2 V1.5.1*, 2009.

[4] Daeyoung koh and Jongsu Lee, "Analysis of Consumers' Choices and Time-Consumption Behaviors for Various Broadcasting and Telecommunication Convergence Services", *ETRI Journal*, vol.32, no.2 , 2010.

[5] Sungkyu Cho, Donhwi Shin, Heasuk Jo, Donghyun Choi, Dongho Won, and Seungjoo Kim, Secure and Efficient Code Encryption Scheme Based on Indexed Table, *ETRI Journal*, vol.33, no.1, 2011.

[6] Minje Kim, Minsik Park, Seung-jun Yang, Ji Hoon Choi, and Han-kyu Lee, "System Aspects of TV-Anytime Metadata Codec in a Uni-directional Broadcasting Environment," *ISCE*, 2007.

[7] Young-Guk Ha, Beom-Seok Jang, Bong-Jin Oh, Yu-Seok Bae, Eui-Hyun Baik, "Effective Encoding of TV-Anytime Metadata Using EXI," *ICCE*, 2011.

[8] Bong-Jin Oh, Sunggeun Jin, EuiHyun Paik, and KwanJong Yoo, "A schema digest based Metadata Encapsulation Architecture with Shareing String Tables, *CA'11*, 2011.

[9] W3C, *Efficient XML Interchange (EXI) Format 1.0*, W3C Recommendation, 2011.

오 봉 진 (Bong-Jin Oh)

1993년 2월 부산대학교 전자계산학과 졸업
1995년 2월 부산대학교 전자계산학과 석사
2012년 2월 충남대학교 컴퓨터공학과 박사
<관심분야> IPTV, N-스크린, XML 인코딩

박 종 열 (Jong-youl Park)

1996년 8월 충남대학교 컴퓨터 공학과 졸업
1999년 2월 광주과학기술원 정보 통신공학 석사
2004년 2월 광주과학기술원 정보통신공학 박사
<관심분야> IPTV, 분산 시스템, 보안, XML 인코딩

김 상 형 (Sang-Hyong Kim)

1997년 2월 한밭대학교 전자계산학과 졸업
2003년 2월 충남대학교 컴퓨터과학과 석사
2003년 3월~충남대학교 컴퓨터과학과 박사과정
<관심분야> 영상 스트리밍, 무선 네트워크, QoS

유 관 종 (Kwan-Jong Yoo)

1976년 2월 서울대학교 계산통계학과 졸업
1978년 2월 서울대학교 계산통계학과 석사
1985년 2월 서울대학교 계산통계학과 박사
<관심분야> Scalable Coding, 멀티미디어 응용, 병렬처리