논문-05-10-1-06

# TV-Anytime 메타데이터의 부호화기 및 복호화기의 구현

김 명 훈[a)‡], 김 혁 만[a)], 양 승 준[b)], 김 재 곤[b)]

# Implementation of Encoder and Decoder for TV-Anytime Metadata

Myounghoon Kim[a)‡], Hyeokman Kim[a)], SeungJun Yang[b)] and JaeGon Kim[b)]


## 요 약

본 논문은 디지털 방송의 제한된 데이터 전송 대역폭 환경에서 효율적으로 메타데이터를 전송하기 위한 방법으로 TV-Anytime 메타데이터를 TV-Anytime 규격에 따라 이진 부호화하고 복호화하는 시스템의 구현에 관한 것이다. 먼저 TV-Anytime 시스템의 적용 환경을 살펴보고, TV-Anytime 부호화기 및 복호화기의 세부 모듈과 기능을 서술한다. 또한 서술된 기능을 바탕으로 TV-Anytime 메타데이터를 프래그먼트 단위로 분할, 부호화, 복호화 및 프래그먼트의 관리에 대한 설계 및 구현 방법을 제안한다. 구현된 TV-Anytime 부호화기 및 복호화기는 ECG(Electronic Content Guide) 및 세그먼트 정보 서비스 등을 제공하는TV-Anytime 메타데이터 기반의 맞춤형 방송 시스템의 핵심 모듈로 사용될 수 있다.

## Abstract

In the paper, we propose a TV-anytime codec that encodes and decodesTV-Anytime metadata according to the TV-Anytime specification so that the resulting binary TV-Anytime metadata can be transferred efficiently through the broadcasting network where the data bandwidth is restricted.. We describe the broadcasting environment that the TV-Anytime codec will be applied to, and the required functionalities of the software modules in detail. For the design of software modules, we show how to implement the modules for metadata fragmentation, encoding, decoding, and the fragments management. The proposed TV-Anytime codec can be utilized as the core components to a personalized digital broadcasting system providing ECG(Electronic Content Guide) and segmentation information services according to TV-Anytime standard.


Keywords : TV-Anytime, metadata, fragment, access unit, container, BiM, TeM

## Ⅰ. 서 론

방송 기술이 아날로그에서 디지털로 전환되는 것은 단지 화질과 음질의 향상뿐만 아니라 새로운 형태의 방송 서비스를 가능하게 하는 것을 의미한다. 또한 저장매체(Local Storage)가 내장된 디지털 방송 수신기(Set-top Box, STB) 혹은 PVR (Personal Video Recorder)이 확산됨에 따라 기존 방송 환경에서는 볼 수 없었던 새로운 서비스가 가능하게 되었다. 그 중에서도 메타데이터(Metadata) 서비스는 새로운 방송 기술 중 가장 주목을 받고 있는 기술이다[1,2,3]. 메타데이터 서비스를 위해서는 효율적으로 메타데이터를 전송 하는 방법 또한 필요하다. 디지털 방송 환경의 제한된 대역폭과 반복적 전송이라는 특성에 따라 메타데이터를 효율적으로 전송하기 위해서는 데이터 크기를 줄일 필요가 있고, 중복 전송 되는 데이터의 처리 방법이 필요하다.

a) 국민대학교 컴퓨터학과
Dept. of Computer Science, Kookmin University
b) 한국전자통신연구원 디지털방송연구단 방송미디어연구그룹
Broadcasting Meida Research Group, Digital Broadcasting Research Division Electronics and Telecommunication Research Institute(ETRI)

TVA(TV-Anytime) 표준은 저장매체를 내장한 STB 환경에서 방송 프로그램을 효율적으로 검색, 브라우징하기 위한 메타데이터의 표현, 전송, 처리 등을 포함한다.

TVA 표준은 2001년 2월에 처음 발표되었고 그 후 2003년에 단방향 방송에 대한 부분인 Phase 1이 완성되었다. Phase 1 표준에서는 단방향 방송 환경에서 메타데이터의 효율적인 처리 기법을 표준화하였다. 현재는 양방향 방송을 위한 Phase 2 표준화가 진행되고 있다

TVA Phase 1 표준의 파트 3인 메타데이터 표준은 크게 다음의 두 부분으로 구성되어 있다. TVA Phase 1 표준 파트3-1은 TVA 메타데이터의 데이터 형식을 규정하고 있다[4]. TVA 메타데이터는 XML 스키마로 작성된 TVA 스키마(XML 문서 타입)에 따라 XML로 기술된다. 그러나 XML은 비정형 데이터를 정형적으로 처리하기 위해 텍스트 태그(tag)를 사용하기 때문에, 일반적으로 데이터 크기가 매우 커지게 된다. 따라서 XML 데이터의 크기를 줄일 수 있는 압축 기법이 필요하다. 또한 방대한 양의 방송용 메타데이터는 모두 하나의 파일로 전송하기 보다는 조각내어 전송하는 것이 유리하다. 특히 전송할 관련 메타데이터가 방송 전에 모두 준비되는 환경도 있으나 실시간 중계방송과 같이 메타데이터가 실시간으로 생성되기도 하며, 또는 방송 편성시간의 변경과 같이 이미 전송된 데이터중 일부분만 수정할 필요도 있다. 이와 같은 환경에 대비하기 위해 전송단에서 XML로 기술된 TVA 메타데이터를 프래그먼트(Fragment) 단위로 분할하여 압축한 후 전송하고, 수신기에서는 수신한 프래그먼트를 복호화하여 현재 유지하고 있는 전체 XML 문서에 첨가 혹은 수정한다. 이를 위해 TVA Phase 1 표준 파트 3-2는 TVA 메타데이터의 분할(Fragmentation), 부호화, 복호화 등을 규정한다[5].

메타데이터는 DVB나 ATSC와 같은 디지털 전송방식에 따라 방송 스트림으로 다중화되어 전송된다. TVA에서는 DVB나 ATSC와 같은 특정 전송방식에 관련되는 문제는 표준화에서 제외하고 있다. 따라서 TVA 표준 파트 3의 메타데이터를 특정 전송방식에 적용하기 위해서는 특정 전송방식을 고려한 새로운 표준이 필요하다. DVB에서는 이미 TVA를 위한 전송 규격을 논의하여 표준으로 확정하였으며[6], ATSC에서는 TVA 메타데이터의 일부를 Advanced EPG 표준으로의 채택을 고려하고 있다.

본 논문은 TVA Phase 1 파트 3-2의 TVA 메타데이터의 분할, 부호화, 복호화 모듈 구현에 관한 것이다. TVA 메타데이터를 분할 및 부호화하여 전송할 수 있는 단위인 컨테이너(Container)로 만드는 기법, 그리고 그것을 복호화해서 프래그먼트 단위로 메타데이터를 다루는 기법의 구현에 대해 설명한다. 구성은 다음과 같다. 2장에서는 TVA 부/복호화기의 적용 환경 및 입력에 대해 설명한다. 3장에서는 TVA 부/복호화기의 구조를 설계한다. 4장에서는 설계한 구조를 어떻게 구현하였는지를 서술한다. 그리고 5장에서는 결론 및 추후 연구에 대해 서술한다.

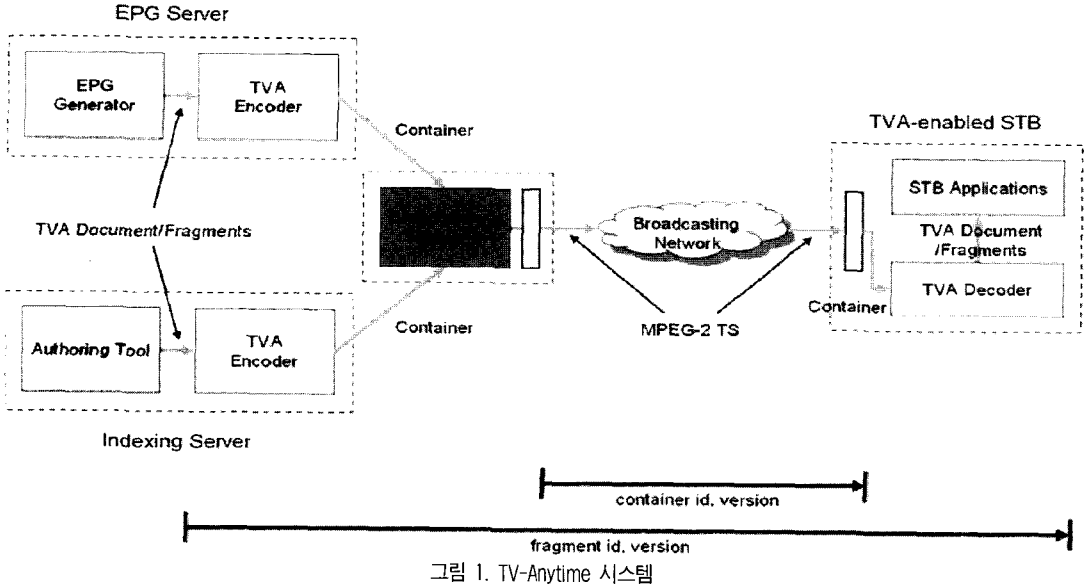



그림 1. TV-Anytime 시스템
Fig 1. TV-Anytime system

## Ⅱ. TV-Anytime 메타데이터 부/복호화기의 적용 환경 및 입력

### 1. TV-Anytime 적용 환경

TVA 메타데이터중 방송국에서 수신기로의 단방향 전송 대상으로는 프로그램 편성 정보(EPG Information)와 프로그램의 세그먼트 정보(Segmentation Information)를 들 수 있다. 그림 1은 방송국과 시청자의 STB에 TVA 표준에 따른 TVA 부호화기(Encoder)와 TVA 복호화기(Decoder)를 적용한 시스템을 보여주고 있다.

방송국은 EPG 서버와 색인 서버에서 EPG 생성기나 세그먼트정보 저작도구와 같은 응용 프로그램이 전송할 TVA 메타데이터를 생성한다. 생성된 TVA 메타데이터는 TVA 부호화기를 거쳐서 분할 및 부호화되어서 컨테이너로 전환되어 송출장비로 전달된다. 송출장비에서 컨테이너는 MPEG-2 transport stream(TS)에 삽입되어 방송망을 통해 시청자의 STB에 전송된다. 시청자의 STB에서는 전송되는 TS에서 컨테이너를 분리하고, 분리된 컨테이너는 TVA 복호화기에 의해 프래그먼트로 복호화된다. TVA 복호화기는 복호화된 프래그먼트들을 관리하고, 시청자는 STB 안의 여러 응용 프로그램을 통해 그 내용을 서비스 받는다.

TVA 메타데이터 복호화기와 부호화기는 특정 프래그먼트를 식별하기 위해 fragment_id와 version을 사용한다. 한편 MPEG-2 TS 전송 시스템에서는 특정 컨테이너를 식별하기 위해 container_id와 version을 사용한다. 이 값들은 시청자의 STB에서 MPEG-2 TS을 통해 전송받는 컨테이너를 식별하기 위한 것으로, 특정 전송 방식에 따라 달라질 수 있다. 따라서 TVA 표준에서는 container_id와 version에 대해 정의하지 않고 있다. 이에 대해서는 현재 유럽 표준인 DVB 표준에 따른 TVA의 전송 규격에서 다루고 있다[6]. 단, 컨테이너의 전송에 관련된 부분은 다루지 않는다.

### 2. TV-Anytime 메타데이터 부호화기의 입력

TVA 메타데이터 부호화기에 들어올 수 있는 가능한 입력은 TVA 스키마에 유효(Valid)한 TVA 문서, 그리고 TVA 표준에서 프래그먼트 단위로 정의된 TVA 프래그먼트로 나눌 수 있다. TVA 문서는 완벽한 전체 문서일 수도 있고 계속 생성되는 중간단계 문서일 수도 있다. 하지만 모두 TVA 스키마에 유효한 문서여야 한다. 만약 입력 TVA 문서가 유효하지 않다면 TVA 부호화기는 이 문서를 다룰 수 없다. TVA 프래그먼트 입력은 TVA 문서의 일부분으로서, MPEG-7 표준에서 정의된 FUU(Fragment Update Unit)와 거의 동일한 형태이다[7].

TVA 메타데이터 부호화기는 위의 두 가지 형태 중 하나의 입력을 받아 컨테이너를 생성한다. 컨테이너는 프래그먼트들을 포함하며, 각각의 프래그먼트에 대응하는 fragment_id와 version을 포함하고 있다. 이 정보는 후에 TVA 메타데이터 복호화기에서 프래그먼트 단위로 정보를 다루기 위해 필요하다. TVA용 STB에서는 STB 응용 프로그램이 TVA 복호화기에 전체 TVA 문서 또는 원하는 프래그먼트를 요구하며, TVA 복호화기는 전송 받은 컨테이너에서 프래그먼트를 복호화하여 현재 메모리에서 유지하고 있는 TVA 문서에서 문서 전체 혹은 해당 프래그먼트를 선택하여 반환한다.

## Ⅲ. TV-Anytime 메타데이터 부/복호화기의 구조 설계

### 1. 기능 요구 조건

TVA 메타데이터 부호화기 및 복호화기는 내부적으로 이진 스트링으로 압축된 BiM(Binary Metadata) 형태뿐만 아니라 텍스트로 기술된 TeM(Textual Metadata) 형태의 프래그먼트를 모두 지원할 수 있도록 설계하였다. TeM과 BiM은 MPEG-7 표준에서 정의한 XML 문서의 압축 포맷으로[7], TVA에서는 문서의 압축을 위해서는 MPEG-7의 방식을 적용하고, Zip library를 적용한 압축을 추가하여 압축 성능을 높인다[5]. 대부분의 TVA 응용에서는 BiM 형태가 사용되겠지만, 내부 테스트용 혹은 대역폭 제한이 크지 않은 환경에서는 BiM 복호화를 위한 부담을 피할 수 있는 TeM 형태가 사용될 가능성이 있다. 따라서 TVA 부/복호화기는 BiM 혹은 TeM 형태의 프래그먼트를 포함하는 컨테이너를 모두 처리할 수 있도록 설계하였다. 하지만 TVA-init 메시지는 TVA 표준에서 TeM 형식을 정의하지 않았기 때문에 항상 BiM 형태로만 컨테이너에 포함된다.

2장에서 설명한 데로 제안하는 TVA 메타데이터 부호화기는 입력으로서 유효한 TVA 문서뿐만 아니라 TVA 프래그먼트도 처리할 수 있도록 설계하였다. 실시간 방송에

대한 세그먼트 정보와 같이 방송 중에 계속적으로 메타데이터가 생성되어 점진적으로 전송되는 경우(Progressive Delivery)에는 한번에 완벽한 전체 TVA 문서 입력을 기대할 수 없다. 이 경우에는 현재까지의 TVA 스키마에 유효한 부분 문서 혹은 TVA 프래그먼트가 생성 순서대로 입력되므로, TVA 부호화기에서 이에대한 대처가 매우 중요하다.

제안하는 TVA 메타데이터 부/복호화기는 현재의 TVA 메타데이터 표준에서 구체적으로 기술하지 않은 몇 가지 중요한 문제에 대처할 수 있도록 설계하였다. 먼저 TVA 부호화 스펙에서는 생성하는 컨테이너의 크기를 제한하지 않고 있다. 따라서 매우 큰 크기의 컨테이너가 생성될 가능성이 있으며, 이런 컨테이너는 특정 전송방식에 따라서는 전송에 어려움이 있을 수 있다. 따라서 컨테이너의 크기를 TVA 부호화기에서 제한할 수 있도록 설계하였다. 또한 TVA 복호화 스펙에서는 단지 부호화된 컨테이너를 복호화하는 방법만 기술하고, 이들을 어떻게 관리하는가에 대한 내용은 배제하였다. 그리고 설계한 TVA 복호화기는 메모리에서 컨테이너를 효율적으로 관리하는 기능을 포함하였다. 이 기능이 포함됨으로써 STB 응용 프로그램은 원하는 프래그먼트를 TVA 복호화기에 요청하기만 하면 되므로 매우 쉽게 TVA 복호화기와 결합될 수 있다. 마지막으로 TVA 부/복호화기의 각 구성 모듈의 중간 결과물이 인메모리(In-memory) 형태로 다른 모듈에 전달될 뿐만 아니라, 파일로도 출력될 수 있게 설계하였다. 이는 모듈 테스트 용도로 매우 유용하다.

## 2. TVA 부/복호화기의 구조

TVA 메타데이터 부/복호화기의 구조는 그림 2와 같다. 그림 2에서 TVA 메타데이터 부호화기는 Fragment Stream Generator, BiM 부호화기, Encapsulator로 구성된다. Fragment Stream Generator는 다음 세 가지 기능을 한다. 첫 번째는 TVA-init 메시지를 만드는 기능이다. Fragment Stream Generator는 BiM 부호화기의 초기값을 설정을 하고, 동일한 초기값으로 TVA 복호화기의 BiM 복호화기를 설정하기 위해 TVA-init 메시지에 설정 값을 실어 보낸다. 메시지에는 Zip 라이브러리(library)에서 사용될 버퍼 크기, 인덱싱 여부, 그리고 복호화할 때 필요한 스키마 정보 등을 포함한다. 두 번째 기능은 변화된 프래그먼트를 탐지하여 FragmentUpdateCommand를 설정하고, ContextPath(스키마에서 프래그먼트의 위치)를 알아내어 프래그먼트와 함께

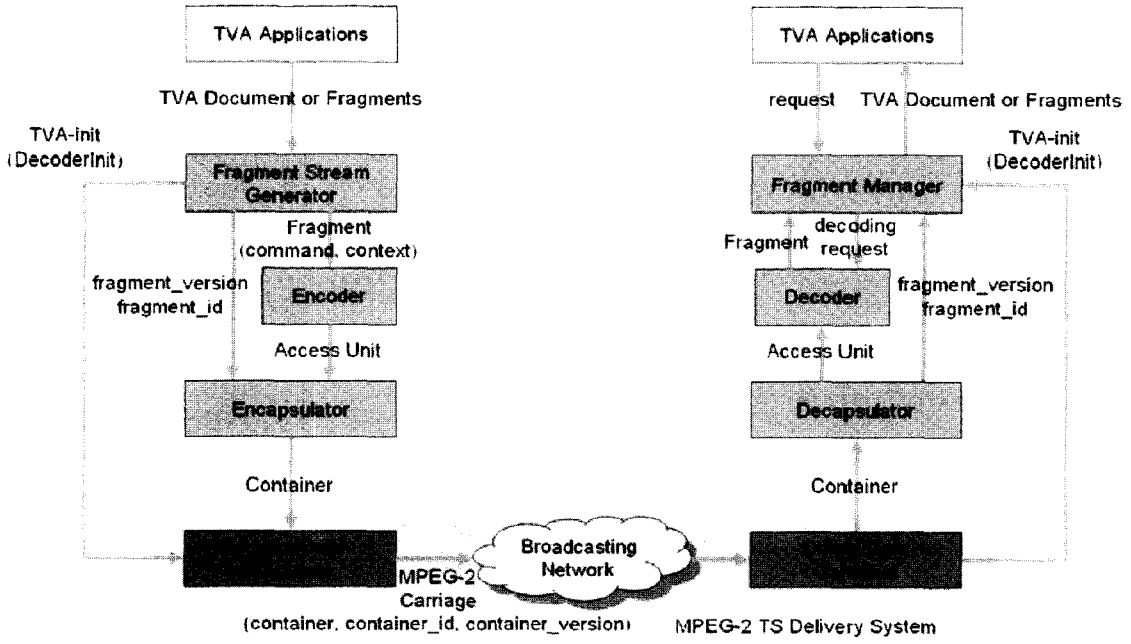



그림 2. TVA 메타데이터 부호화기 및 복호화기의 구조
Fig 2. Structure of TVA metadata encoder and decoder

FUU 형태로 BiM 복호화기에게 전달한다. 만약 시스템이 BiM 형태가 아닌 TeM 형태의 데이타를 요구한다면 BiM 복호화기를 거치지 않고 TeM 형태의 AU(Access Unit)를 Encapsulator에게 전달한다. 세 번째 기능은 fragment_id와 version을 부여하고 관리하는 일이다. Fragment Stream Generator는 BiM 부호화기에게 특정 FUU 집합의 압축을 요청한다. 동시에 그 FUU 집합에 속한 fragment_id와 version을 Encapsulator에게 넘겨주어, Encapsulator가 BiM 부호화기의 출력인 AU와 함께 fragment_id와 version을 묶어 컨테이너를 생성할 수 있게 한다.

BiM 부호화기는 텍스트 형태의 FUU를 압축하여 비트 스트링으로 변환하는 기능을 담당한다. 이때 압축된 비트 스트링에는 각 프래그먼트 마다 해당 FragmentUpdate-Command와 ContextPath가 부가되어 BiM 형태의 FUU들을 포함하는 AU를 생성한다. 만일 응용 프로그램이 TeM 형태의 AU를 요구하면 BiM 부호화기는 실행되지 않는다. 여기서 언급하는 TVA 표준의 FUU와 MPEG-7의 FUU는 형태는 동일하지만 약간의 차이점을 가지고 있다. MPEG-7의 FUU와는 달리 TVA 표준에서는 FUU가 될 수 있는 단위를 프래그먼트 단위로 구분 지어 정의해 놓았기 때문에 항상 FUU는 프래그먼트 단위의 갱신을 기술하고 있다.

Encapsulator는 AU와 그 AU에 포함된 FUU들의 fragment_id와 version을 입력 받아 이들을 묶어 하나의 컨테이너를 생성한다. 이렇게 생성된 컨테이너가 TVA 부호화기의 최종 출력이 된다. fragmentid와 version이 압축된 혹은 압축되지 않은 FUU와 분리되어 묶여서 생성되는 이유는, BiM 형태의 경우 TVA 복호화기에서 압축된 FUU를 풀지 않고도 AU에 포함된 FUU들의 fragment id와 version을 알 수 있게 하기 위해서이다.

한편 TVA 메타데이터 복호화기는 Decapsulator, BiM 부호화기, Fragment Manager로 구성된다. Encapsulator는 전송 받은 컨테이너의 AU 부분과 헤더 부분을 분리하는 역할을 한다. 분리된 헤더에는 AU에 포함된 fragment_id와 version이 기록되어 있다. 분리된 AU는 BiM 복호화기에 입력 되고, 분리된 헤더는 프래그먼트 매니져(fragment manager)가 사용하게 된다.

BiM 복호화기는 AU를 입력 받아 거기에 포함된 BiM 형태의 FUU들의 압축을 풀어 텍스트 형태의 프래그먼트를 출력한다. BiM 복호화기는 프래그먼트 매니져에 의해 구동된다. 즉, 프래그먼트 매니져가 원하는 fragment_id와 version을 선택적으로 BiM 복호화기에게 전달하여 복원 요청을 하면, BiM 복호화기는 입력 AU에서 해당 FUU만 선택적으로 복원한다. 이렇게 함으로서 복원하는 프래그먼트의 수를 최소화하여 TVA 복호화기의 효율을 높일 수 있다.

프래그먼트 매니져는 응용 프로그램의 요청에 따라 응용 프로그램이 원하는 TVA 문서 또는 프래그먼트를 넘겨준다. 이를 위해 먼저 프래그먼트 매니져는 TVA-init 메시지을 받아서 BiM 복호화기를 설정한다. 응용 프로그램의 요청에 따라 Encapsulator와 BiM 복호화기를 호출하여 전체 TVA 문서 혹은 원하는 특정 TVA 프래그먼트를 검색하여 넘겨준다. 이를 위해 프래그먼트 매니져는 TVA 복호화기 들어오는 모든 프래그먼트를 메모리의 캐시에서 유지한다.

## Ⅳ. TV-Anytime 메타데이터 부/복호화기의 구현

여기서는 3장에서 설계한 TVA 메타데이터 부호화기 및 복호화기의 구현에 대해 기술한다. TVA 부호화기는 마이크로소프트의 Visual Studio로 개발하였으며, 복호화기는 Linux STB에서 gcc 3.1로 구현하였다. 한편 BiM 부호화기와 복호화기는 Expway사의 BinXML 2.1을 사용하였다[8]. BiM 부/복호화기 자체가 매우 방대한 소프트웨어 모듈이므로, TVA 표준에서 정의한 기능만 구현하였으며, 스펙에서 정의하지 않은 부분은 상용 소프트웨어 모듈을 이용하였다.

### 1. TVA 메타데이터 부호화기의 구현

3장에서 설명한 Fragment Stream Generator는 사실상 TVA 메타데이터 부호화기 전체를 관리하고 있는 모듈이다. 이 모듈에서 BiM 부호화기를 설정하고, 입력 문서로부터 프래그먼트를 구별하여 이들의 fragment_id와 version을 관리하며, BiM 부호화기와 Encapsulator를 호출하여 컨테이너를 생성한다. 따라서 그림 3의 TVA 메타데이터 부화화기 모듈의 구조에서 BiM 부호화기와 Encapsulator를 제외한 모든 부분이 Fragment Stream Generator이다.

Fragment Stream Generator는 TVA-init 메시지를 생성한다. 이 메시지를 통해 BiM 부호화기 설정 값과 동일한 값으로 TVA 복호화기의 BiM 복호화기를 설정할 수 있다. TVA 복호화기로 TVA-init 메시지가 전송될 때 TVAMain 프래그먼트가 메시지 안에 포함될 수 있다. 만일 메시지에 이 프래그먼트가 포함되어 있지 않다면, 첫 번째 전송할

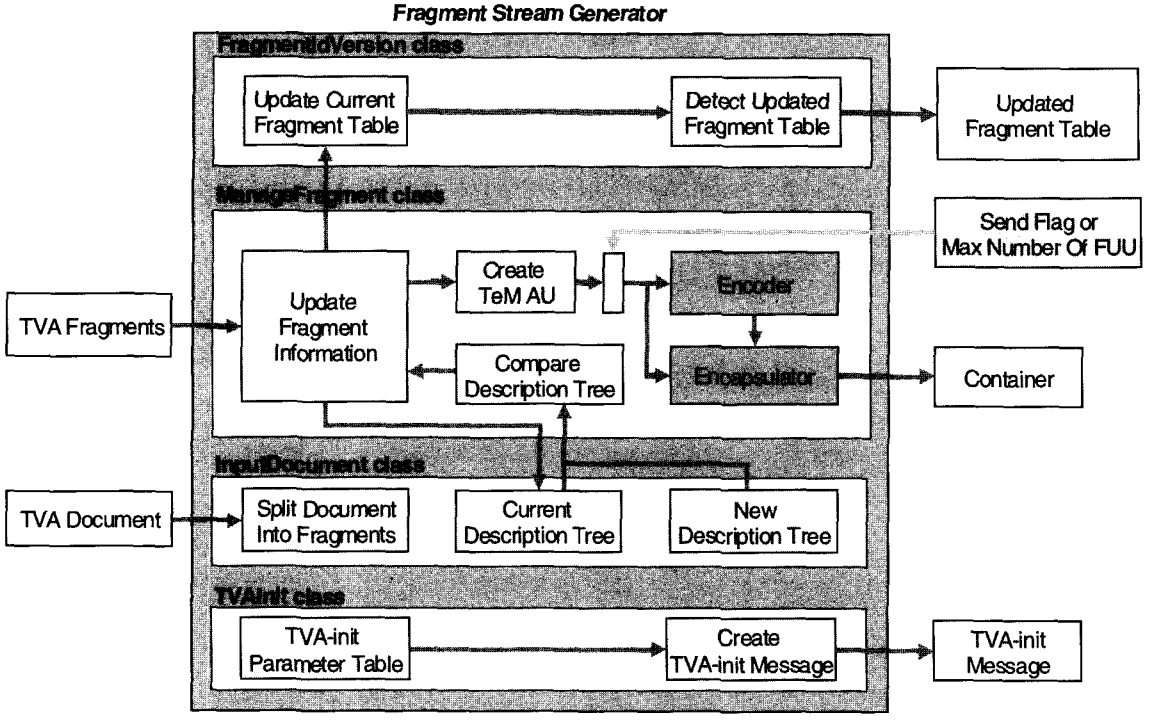



그림 3. TVA 메타데이터 부호화기의 주요 모듈
Fig. 3. TVA metadata encoder modules

AU에 이 프래그먼트를 포함해 전송해야 한다. 이 프래그먼트는 TVA 복호화기의 Current Description Tree(CDT)를 초기화 해주는 역할을 한다. CDT는 TVA 복호화기가 주 메모리에서 DOM Tree로 저장하고 있는 현재 유지하는 최신 버전의 TVA 문서를 말한다. 따라서 구현한 Fragment Stream Generator는 TVAMain 프래그먼트가 들어있는 컨테이너를 생성하는 기능을 가지고 있다.

한편 TVAMain 프래그먼트는 TVA 스키마에 유효하게 전송하기 힘들다. 왜냐하면 TVA 표준 스키마에서 프래그먼트가 가능한 요소(Element)의 대부분은 minOccur가 0이나, 유독 "Review"와 "SegmentInformation" 요소만minOccur가 1로 정의되어 있기 때문이다. 따라서 유효한 TVAMain 프래그먼트를 보내기 위해 항상 하나의 빈 Review 및 Segment-Information 요소를 추가하여 TVAMain 프래그먼트를 생성한다. 또한 고정되어 전송되는 TVAMain 프래그먼트와 "OnDemandProgram" 요소 사이에 "OnDemand-Service" 요소가 존재할 수 있다. 이 요소는 minOccur가 1인 "OnDemand-Program" 프래그먼트를 자식 노드로 가지면서도 TVAMain fragment에 포함되어 있지 않다. 만약 이 프래그먼트를 TVAMain에 포함시키려면, TVA 표준 스키마에 유효하게 만들기 위해 또 고정의 "OnDemand-Program" 프래그먼트를 임의적으로 추가해야 한다. 그렇지만 그 방법은 "OnDemandService"의 기능을 발휘하는 적절한 방법이 아니다. "OnDemandService"의 필수 속성인 "serviceIDRef"를 보면 이 요소도 프래그먼트와 비슷한 성격을 띤 것을 알 수 있다. 따라서 그 해결책으로 "OnDemand-Service"를 새로운 프래그먼트로 추가시켰다.

구현한 TVA 부호화기는 TeM 형식과 BiM 형식 FUU로 이루어진 AU를 모두 생성할 수 있다. 이러한 기능 요구를 구현하기 위해서 내부의 중간 결과물을 TeM 형태의 AU로도 생성하였다. 그림 3에서 Create TeM AU는 이 기능을 수행하는 모듈이다. 이 모듈은 TVA 문서 혹은 프래그먼트가 입력으로 들어왔을 때, 기존과 변화된 프래그먼트만을 알아내어 TeM 형식의 AU로 생성한다. 만일 응용 프로그램이 BiM 형태의 전송을 요구한다면생성된 TeM AU를 BiM 부호화기에게 넘겨주어, BiM AU를 생성한 후 Encapsulator 거쳐 컨테이너를 생성한다. 만약 응용이 TeM 형태만 요구하면, 곧바로 Encapsulator를 사용하여 컨테이

너를 생성한다.

Fragment Stream Generator는 2장에서 설명한 두 가지 타입의 입력에 따라 다른 방법으로 동작하도록 구현하였다. 만약 TVA 문서 입력이 들어오면, 프래그먼트 단위로 분할하는 모듈에 넘겨진다. 이 모듈은 입력 문서에서 TVA 표준에서 정의된 모든 프래그먼트를 찾아내고, 그 프래그먼트의 루트 요소의 ContextPath를 저장한다. 저장된 위치 정보를 가지고 이 프래그먼트를 현재 모듈 내에서 유지하고 있는 CDT와 비교한다. 또한 CDT의 ContextPath 위치에 어떠한 FragmentUpdateCommand를 적용해야 하는지를 결정한다. 스펙에서 정의한 FragmentUpdateCommand는 replaceContent와 deleteContent 두 가지가 있다. replaceContent는 ContextPath에 위치하는 프래그먼트를 입력 Payload의 프래그먼트로 변경한다. 만일 CDT에서 그 위치에 프래그먼트가 없다면 입력 Payload의 내용(새로운 프래그먼트)을 추가한다. deleteContent는 ContextPath가 가리키고 있는 위치의 프래그먼트를 삭제한다. 그림 3의 Compare Description Tree가 이 기능을 수행한다.

변경을 위한 replaceContent를 알아내기 위해서는 CDT의 해당 프래그먼트와 입력 TVA 문서의 지정된 프래그먼트가 원래 같은 것이었다가 변경되었다는 근거를 찾아야 한다. 두 프래그먼트의 변경 여부를 알기 위해 프래그먼트의 루트 요소에서 ID 타입을 가진 속성을 서로 비교한다. ID 타입은 CDT 내에서 유일한 값을 가지고 있기 때문에, 같은 ID 값을 갖는 루트 요소의 하부 구조가 같다면 두 프래그먼트가 같다고 판단할 수 있다. 만일 서로 같은 속성 값을 가지고 있는데 두 프래그먼트의 내용이 다르다면, 그것은 변경을 의미하므로 FragmentUpdateCommand를 replaceContent로 정한다. 만일 루트 요소와 같은 ID 값을 가진 요소가 CDT에 없다면, CDT에 새로운 프래그먼트를 추가한다. 추가의 의미를 가지고 있는 replaceContent와 삭제를 의미하는 deleteContent의 판단 여부는 CDT와 입력 TVA 문서의 프래그먼트를 비교하여, 한쪽에는 있는데 다른 쪽에는 없는 경우를 찾는다. 만약 CDT에는 있는데 입력 TVA 문서에는 없다면 deleteContent로 간주한다. 그 반대의 경우는 추가의 의미인 replaceContent로 간주한다.

Compare Description Tree 모듈은 FragmentUpdateCommand, ContextPath, Payload을 알아내고, 그 정보를 Update Fragment Information 모듈로 전달한다. 이 모듈에서는 입력된 정보를 TeM 형태의 FUU로 만든다. 위의 세 정보는 FUU를 구성하는 기본 단위이다. 생성된 FUU는 Create TeM AU 모듈로 보낸다. 또한 CDT를 갱신하고, CDT에서 유지하고 있는 모든 fragment_id와 version을 갖고 있는 프래그먼트 테이블을 갱신한다. Create TeM AU를 거쳐 만들어진 AU는 응용 프로그램의 요구에 따라 BiM 부호화기를 사용할 수도 있고, 그렇지 않을 수도 이다. AU는 마지막으로 Encapsulator를 거쳐서 컨테이너로 생성된다.

Fragment Stream Generator의 또 다른 타입의 입력은 TVA 프래그먼트이다. 이 형태의 입력은 이미 FUU를 만들기 위한 모든 정보, 즉 프래그먼트의 FragmentUpdateCommand, ContextPath, Payload가 포함된 형태이다. 따라서 TVA Fragment가 입력으로 들어오면 FragmentUpdateCommand를 판단할 필요가 없기 때문에 CDT와 비교하지 않는다. 이 경우에는 입력이 Update Fragment Information 모듈로 직접 전달되어 TeM 형태의 AU를 만들고, 나머지 부분은 TVA 문서 입력과 동일하게 처리된다.

TVA 문서 입력과 TVA 프래그먼트 입력 처리에서 또 다른 차이는 컨테이너가 생성되는 시기이다. TVA 문서 입력의 경우는 문서가 입력될 때 마다 컨테이너를 생성할 수 있다. 따라서 컨테이너에 포함되는 FUU의 수를 제한하지 않으면, 크기가 매우 큰 AU가 하나의 컨테이너로 생성될 수 있다. 이를 방지하기 위해 TVA 문서 입력일 경우, 하나의 컨테이너에 들어갈 수 있는 FUU의 최대수를 제한하는 기능을 구현하였다. 한편 TVA 프래그먼트 입력일 경우는 하나의 컨테이너가 하나의 AU로 만들어질 수 있다. 이렇게 되면 최종 생성된 컨테이너의 수가 매우 많아져서 전송이 매우 빈번해진다. 이를 방지하기 위해 프래그먼트를 하나씩 전송하는 방식과 여러 개를 모아서 전송하는 방식을 설정하는 플래그(send flag)를 추가하였다. 즉 이 플래그가 참일 때는 프래그먼트가 입력될 때마다 컨테이너가 만들어지게 되고, 그렇지 않으면 TeM 형식의 AU 안에 FUU를 계속 추가하기만 한다. 추가된 프래그먼트의 수가 미리 설정된 프래그먼트의 최대수에 도달하면 이때 컨테이너가 생성된다. 이러한 방법으로 컨테이너의 생성되는 시기를 정할 수 있다. 그림 3의 Create TeM AU와 BiM 부호화기 사이에 위치하는 모듈이 이 기능을 담당한다.

Fragment Stream Generator에서 중요한 기능 중 하나가 fragment_id와 version을 관리하는 것이다. fragment_id는 CDT에서 유일한 값을 지닌다. CDT의 모든 fragment_id와 version은 프래그먼트 테이블에 저장이 된다. 그리고 새로운 FUU가 생길 때 마다 그 변화를 이 테이블에 기록하

고, Encapsulator에게는 테이블에서 갱신된 부분만을 축출하여 전달한다.

Encapsulator모듈은 Fragment Stream Generator에서 보내온 프래그먼트 테이블과 TeM 혹은 BiM 형태의 AU를 받아서 컨테이너를 생성한다. 만일 BiM 형태의 AU가 들어온다면 데이터 저장소(data repository)의 타입은 바이너리(binary)로 설정하여 BiM AU를 컨테이너에 저장하며, TeM 형태의 AU가 입력일 때는 스트링(string) 타입의 데이터 저장소를 사용하여 컨테이너를 만든다.

## 2. TVA 메타데이터 복호화기의 구현

그림 4는 구현한 TVA 복호화기의 모듈 구조이다. 그림에서 프래그먼트 매니져는 TVA 복호화기 전체를 제어하고 관리하며, 입력되는 컨테이너 안의 FUU들을 읽어 들여 CDT를 항상 최신의 정보로 유지하는 역할을 하는 모듈이다. 컨테이너를 복호화하려면 먼저 TVA-init 메시지가 필요하다. 부호화기로부터 TVA-init 메시지를 전송 받으면, 메세지 내의 파라미터 초기값을 이용하여 TVA 복호화기의 초기값을 설정한다. TVA-init 메시지 안에 존재하는 DecoderInit의 InitialDescription에는 TVAMain 프래그먼트가 저장될 수 있다. 만일 InitialDescription이 없는 경우에는 메시지 이후 첫번째 전송되는 컨테이너에서 이 프래그먼트를 얻어야 한다. 구현한 복호화기에서는 TVA-init 메시지와 첫번째 컨테이너 양쪽 모두에서도 TVAMain 프래그먼트가 전송되지 않을 경우, 프래그먼트 매니져가 내부에서 유지하고 있는 TVAMain fragment를 사용할 수 있도록 구현하였다.

비트스트링 형태의 컨테이너가 프래그먼트 매니져에 입력되면, 프래그먼트 매니져는 Decapsulator를 호출하여 AU를 얻는다. 이때 프래그먼트 매니져에서 생성되는 AU가 BiM인지 TeM인지를 알 수는 없다. BiM인지 TeM인지의 여부는 AU의 속성인 structure_id를 사용해서 알아낸다.

BiM일 경우에는 BiM 복호화기 모듈을 호출하여 복원된 TeM FUU를 포함하는 AU를 얻는다. 이때 복원 실행 횟수를 줄이기 위해서 Decapsulator 알아낸 fragment_id와 version을 프래그먼트 매니져에서 내부적으로 유지하고 있는 프래그먼트 테이블의 fragment_id와 version과 비교한

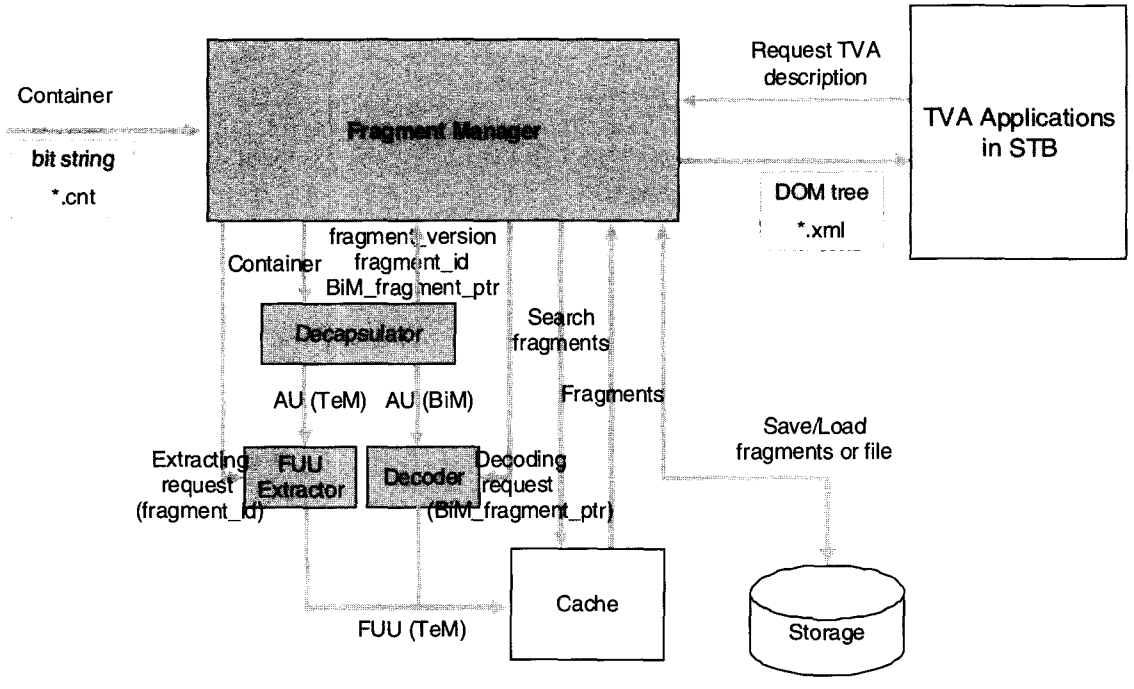



그림 4. TVA 메타데이터 복호화기의 주요 모듈
Fig 4. TVA metadata decoder modules

다. 만일 입력 fragment_id가 테이블에 없는 경우와 테이블에 있더라도 version이 테이블의 값보다 클 경우에는 복원 과정을 거치 TeM AU로 만들지만, 그렇지 않을 경우는 이미 동일한 프래그먼트 가 존재하는 것이므로 그 프래그먼트는 버린다.

TeM AU는 XML 문서 형태이기 때문에 CDT로 읽어들여 포함된 각각의 FUU의 FragmentUpdateCommand, ContextPath, Payload를 알아낸다. FragmentUpdateCommand가 ReplaceContent일 경우, 프래그먼트 테이블에서 ContextPath에서 마지막 노드의 포지션 코드 값을 찾아서 이미 존재하는 경우에는 이전에 추가되었던 Payload 현재의 입력 Payload로 대체하고, 없는 경우에는 해당 경로에 추가시키고 프래그먼트 테이블을 갱신한다. FragmentUpdateCommand가 DeleteContent일 경우에는 내부 테이블에서 해당 노드를 찾아서 노드를 삭제하고 테이블을 갱신한다.

fragmentid 값은 16 비트이므로 일정 기간이 지나 최대값에 도달하면, 다시 0부터 재사용된다. 따라서 fragment_id 값의 재사용을 위해서, 일정기간 동안 전달되지 않은 fragment_id가 프래그먼트 테이블에서 가리키는 노드를 삭제해야 한다. 구현한 TVA 복호화기에서는 이를 위해 프래그먼트 테이블의 모든 프래그먼트에 각각 프래그먼트의 나이 카운터를 유지하고, 복호화기 모듈 전체가 사용하는 하나의 시스템 카운터를 유지한다. 시스템 카운터는 컨테이너를 한번 복원할 때마다 1씩 증가하고, 입력 프래그먼트가 해당하는 노드가 CDT에 반영되어 프래그먼트 테이블에서 해당 노드의 값이 변경될 때마다 현재의 시스템 카운터 값을 프래그먼트의 나이 카운터 값으로 할당한다. 응용 프로그램에서 원하는 순간에 TVA 부호화기의 시스템 카운터 값과 프래그먼트 테이블에서 프래그먼트의 나이 카운터 값의 차가 미리 설정된 임계치 보다 큰 프래그먼트들은 모두 삭제할 수 있는 기능을 구현하였다.

한편 복원의 최종 결과인 CDT를 파일 형태로 출력할 때, 다시 TVA 복호화기에 이 파일을 로딩하여 재사용하기 위해서는 fragment_id와 Position Code를 프래그먼트 별로 저장해야 한다. 이를 위해 프래그먼트 테이블의 각 프래그먼트 노드에 fragment_id, version 속성을 이용한다. 즉, CDT에 FUU를 적용할 때 fragment_id에 Id 값을 저장하고, version에 Position Code값을 저장한다. 따라서 CDT를 파일로 출력할 때는 현재의 DOM tree를 그대로 저장하면 되고, 역으로 파일을 CDT로 읽어올 때는 노드의 fragment_id, version 속성에서 Id 값과 Position Code 값을 읽고, version은 -1, 나이 카운터 값은 TVA 복호화기의 시스템 카운터 값을 할당한다. 특히 version의 경우, 만일 파일이 굉장히 오래 전의 CDT를 저장했기 때문에 특정 fragment_id의 version 값이 크더라도 이미 그 fragment_id는 재사용되어 낮은 version 값을 가질 수도 있다. 이 경우에는 큰 version 값 때문에 새로 입력된 낮은 version의 프래그먼트가 버려지고, 오래된 프래그먼트를 계속 유지하는 경우가 생길 수도 있다. 이를 방지하기 위해 파일에서 읽어 들인 프래그먼트는 무조건 새로 전송된 프래그먼트에 의해 대체되도록 컨테이너의 프래그먼트가 가질 수 있는 version의 최소값인 0보다 작은 -1을 할당한다.

Decapsulator 입력으로 주어진 컨테이너를 컨테이너 헤더의 정보로 각 구조(structure) 별로 나눠서, 데이터 저장소의 BiM AU나 TeM AU를 만든다. BiM AU일 경우 Encapsulate Structure의fragment_id, fragment_version, fragment_ptr의 정보를 얻어 프래그먼트 매니져로 하여금 FUU를 복호화 여부를 판단하도록 한다. TeM AU의 경우에는 Payload의 루트 요소안에 속성 중 fragment_id와 version으로 FUU가 유효한지를 파악하여 CDT에 추가하거나 버린다.

## V. 결 론

표 1은 원본 문서를 위의 구현된 부호화기에 의해 나온 결과물을 FUU의 개수 제한에 따른 시간과 결과물의 크기로 나타낸 것이다. 그 크기로 압축률을 계산해 본 결과 평균 87.7% 의 압축률을 보였으며 10KB당 약 1초의 시간이 걸려 부호화되었다. 그리고 표에 나타나 있듯이 FUU의 개수 제한에 의해서 부호화하는 시간이 빨라지고 압축률 또한 향상 되었다. 하지만 FUU의 개수 제한의 숫자가 늘어날수록 컨테이너 하나에 포함될 FUU의 개수가 늘어나서 컨테이너 하나의 크기를 커지게 한다. 이것은 대역폭이 작은 환경에서는 문제가 될 수 있기 때문에 그 숫자를 적당히 조절하면서 부호화를 해야 한다. 한가지 더 생각해야 할 점은 FUU를 이용한 문서의 압축은 변화되는 부분만을 찾아서 부호화 하는 방법을 취하고 있으므로 위의 결과보다 더 작은 양의 데이터 크기를 사용해서 문서를 갱신할 수 있다.

TVA 스키마에 유효한 TVA 문서를 TVA 전송 표준에 맞게 프래그먼트로 분할, 부호화하여 컨테이너를 생성하는

표 1. TVA 부호화기를 사용한 결과
Table 1. A Result Table of TVA Encoder

| (원본 문서 크기) | FUU 개수 제한에 따른 결과물 크기 (단위 : KB) | | | | FUU 개수 제한에 따른 부호화하는데 걸린 시간 (단위 : sec) | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 10 | 100 | 1000 | 1 | 10 | 100 | 1000 |
| Test0.xml (95 KB) | 5.00 | 4.82 | 4.80 | 4.08 | 1.219 | 1.156 | 1.125 | 1.125 |
| Test1.xml (74 KB) | 1.50 | 1.26 | 1.24 | 1.24 | 1.469 | 1.360 | 1.375 | 1.375 |
| Test2.xml (7 KB) | 2.43 | 1.94 | 1.90 | 1.90 | 2.406 | 2.282 | 2.266 | 2.250 |
| Test3.xml (3 KB) | 1.05 | 0.86 | 0.84 | 0.84 | 1.219 | 1.172 | 1.125 | 1.141 |
| Test4.xml (2 KB) | 0.55 | 0.45 | 0.45 | 0.45 | 0.562 | 0.547 | 0.547 | 0.547 |
| Test5.xml (51 KB) | 4.74 | 4.57 | 4.55 | 4.55 | 1.125 | 1.047 | 1.031 | 1.031 |
| Test6.xml (522 KB) | 6.06 | 5.60 | 5.58 | 5.58 | 2.203 | 2.047 | 2.047 | 2.000 |
| Test7.xml (690 KB) | 51.9 | 51.1 | 51.0 | 51.0 | 8.141 | 6.734 | 6.532 | 6.500 |
| Test8.xml (2341 KB) | 164.0 | 162.0 | 161.0 | 160.0 | 58.515 | 22.422 | 18.937 | 18.656 |
| Test9.xml (697 KB) | 63.6 | 61.7 | 61.5 | 61.5 | 20.328 | 12.953 | 12.484 | 12.328 |

TVA 메타데이터 부호화기, 그리고 컨테이너를 입력으로 하여 역으로 TVA 문서를 얻는 TVA 복호화기의 구조 및 기능, 그리고 이들의 설계 및 구현을 제시하였다. 구현한 TVA 메타데이터 부호화기와 복호화기는 TVA을 사용한 ECG 및 세그먼트 정보 서비스 등을 제공하는 맞춤형 방송 시스템에 활용될 수 있으며, 맞춤형 방송 시스템에 통합하기 위한 연구는 별도로 진행 중이다.

구현한 시스템은 TVA 표준에서 정의하지 않은 많은 추가적 기능을 갖추고 있다. 입력으로 TVA 문서 뿐만 아니라 TVA 프래그먼트를 직접 받는 기능, TeM과 BiM 모두를 지원하는 기능, 각 단계에서 메모리 출력과 파일 출력을 모두 지원하는 기능, 부호화기에서 생성되는 컨테이너의 크기를 제어하는 기능, 복호화기에서 메모리 관리를 위해 오래된 프래그먼트를 자동 제거하는 기능, 파일 로딩 시 전송된 프래그먼트가 파일의 프래그먼트 보다 우선함으로써 항상 최신 프래그먼트를 유지하는 기능, 응용 프로그램의 내용 검색을 지원하는 기능 등은 시스템을 구현하면서 얻은 경험을 바탕으로 추가된 기능이다.

현재 구현된 시스템은 아직 최적화되어있지 않다. 특히 TVA 복호화기는 매우 작은 메모리와 낮은 성능의 CPU를 장착한 STB에서 실행되어야 하므로, 최적화에 특히 많은 노력이 필요하다. 또한 향후 양방향 환경에서 TVA 서비스가 제공될 경우에는 TVA 부호화기도 STB에서 실행되어야 하므로 여기서도 최적화가 매우 중요하다. 또한 현재 구현된 시스템은 검색 효율 개선을 위한 색인 부분은 구현에서 제외하였다. 앞으로는 최적화를 위한 추가적인 연구와 색인부분의 구현에 대한 연구가 필요할 것이다.

## 참 고 문 헌


[1] T.J. Sargeant, "Braodcast services enabled by local storage and connectivity", Proc. EUROCON 2003, Vol.2, pp.375-378, Sept. 2003.

[2] A.M. Ferman, P. Van Beek, J.H. Errico, M.I. Sezan, "Multimedia content recommendation engine with automatic inference of user preferences", Proc. Int'l Conf. on Image Processing (ICIP) 2003, Vol.3, pp.49-52, Sept. 2003.

[3] B. Marusic, M. Leban, "The myTV system a digital interactive television platform implementation", Proc. IEEE Int'l Conf. on Multimedia and Expo (ICME) 2002, Vol.2, pp.305-308, Aug. 2002.

[4] TV-Anytime Forum, "TV-Anytime Phase 1, Part 3: Metadata, Sub-part 1: Metadata schemas", ETSI Standard, ETSI TS 102 822-3-1, V1.1.1, Oct. 2003.

[5] TV-Anytime Forum, "TV-Anytime Phase 1, Part 3: Metadata, Sub-part 2: System aspects in a uni-directional environment", ETSI Standard, ETSI TS 102 822-3-2, V1.1.1, Oct. 2003.

[6] TV-Anytime Forum, "DVB: Carriage and signaling of TV-Anytime information in DVB transport streams", ETSI Standard, ETSI TS 102 323, V1.1.1, Sep. 2004.

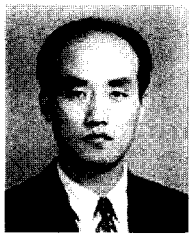
[7] ISO/IEC JTC1/SC29/WG11 (MPEG), "Information Technology Multimedia Content Description Interface Part 1: Systems", International Standard 15938-1, ISO/IEC FDIS 15938-1:2001, Sep. 2001.

[8] Expway, "BinXML-TV 2.1", http://www.expway.com.

## 저 자 소 개

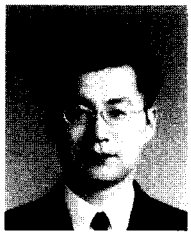

**김 명 훈**

- 2003년 : 국민대학교 컴퓨터학과 (학사)
- 2005년 : 국민대학교 컴퓨터학과 (석사)
- 2005년~현재 : 고려대학교 전자컴퓨터 공학과 (박사)
- 주관심분야 : 메타데이타 처리, 비디오 저작도구, 디지털 방송, MPEG-2, MPEG-4



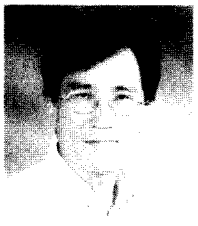
**김 혁 만**

- 1985년 : 서울대학교 컴퓨터공학과 (학사)
- 1987년 : 서울대학교 컴퓨터공학과 (석사)
- 1996년 : 서울대학교 컴퓨터공학과 (박사)
- 1999년 : 한국통신 멀티미디어연구소 선임 연구원
- 1999년~현재 : 국민대학교 컴퓨터학부 교수
- 주관심분야 : 메타데이타 처리, XML, 비디오 데이터베이스, 비디오 저작도구



**양 승 준**

- 1999년 : 순천대학교 전산학과 (학사)
- 2001년 : 전남대학교 전산학과 (석사)
- 2001년~현재 : 한국전자통신연구원 방송미디어연구그룹 연구원
- 주관심분야 : TV-Anytime, 디지털방송, 맞춤형방송, MPEG-7, MPEG-21



**김 재 곤**

- 1990년 : 경북대학교 전자공학과 (학사)
- 1992년 : KAIST 전기 및 전자공학과 (석사)
- 2005년 : KAIST 전기 및 전자공학과 (박사)
- 2001년~2002년 : 뉴욕 콜롬비아대학교 방문연구원
- 1992년~현재 : 한국전자통신연구원 방송미디어연구그룹 선임연구원/방통융합미디어연구팀장
- 주관심분야 : 영상통신, 비디오신호처리, 비디오적응, MPEG-7, MPEG-21