스키마를 이용한 XML 문서의 압축과 복원

염지현° 김혁만
국민대학교 전산과학과 멀티미디어 데이터베이스 연구실
{jhyum, hmkim}@kookmin.ac.kr

Compression/Decompression of XML Instance Documents Conforming to a Schema

Jihyun Yum° Hyeokman Kim
Multimedia Database Lab., Dept. of Computer Science, Kookmin University

요 약

본 논문은 MPEG-7 BiM 규격에 따라 XML 스키마 정의를 기반으로 바이너리 형태로 압축하고 복원하는 시스템의 구현에 관한 것이다. MPEG-7 BiM 압축기 및 복원기의 세부 모듈과 기능을 서술하고, 설계 및 구현 방법을 제안한다. 구현된 MPEG-7 BiM 압축기 및 복원기는 대역폭의 제약이 심한 방송 분야에서 메타데이터 전송을 위한 핵심 모듈로 사용 될 수 있다.

1. 서 론

현재 방송 통신 산업은 유·무선 통합 및 통신과 방송 기술이 융합되면서 양방향 서비스로 변화되고 있다. 이 시점에서 디지털 콘텐츠를 제작, 전송, 처리 할 수 있는 방법으로 메타데이터(metadata)가 부각되고 있다. 일반적으로 메타데이터는 XML로 기술된다. XML은 텍스트 포맷이고 의미를 구별하기 위하여 태그를 사용하기 때문에, XML 인스턴스(instance) 파일은 바이너리 파일과 비교하면 그 크기가 매우 크다. 따라서, 대역폭의 제약이 심한 방송 환경에서 효과적으로 XML 인스턴스 문서를 압축할 수 있는 방법이 요구된다.

현재까지 XML 인스턴스 문서를 압축하는 방법들에 대한 연구가 활발이 이루어지고 있다. Liefke는 기본적으로 zlib과 다른 특정 압축기 및 사용자가 정의한 압축기를 복합적으로 사용하여 XML 문서를 압축하는 XMill을 개발하였다[1]. MPEG Forum에서는 XML 스키마를 이용한 BiM 압축 기법을 제안하였으며, 맞춤형 방송 표준인 TV-Anytime Forum에서 TV-Anytime 메타데이터[2]의 압축/복원에 BiM을 적용하기로 채택하여, 그 중요성이 더욱 증대되고 있다.

본 논문은 MPEG-7 파트 1의 BiM 압축 및 복원 스펙의 구현에 관한 것이다. 본 논문의 구성은 다음과 같다. 2장에서는 본 논문에서 제안한 MPEG-7 BiM 압축/복원기의 구조를 설계한다. 3장에서는 설계한 구조를 어떻게 구현하였는지를 서술한다. 그리고 4장에서는 결론 및 추후 연구에 대해 서술한다.

2. BiM 압축/복원기의 구조 설계

본 논문에서 설계한 BiM 압축기의 구조는 그림 1과 같다. 그림 1에서 BiM 압축기는 크게 schema validator, schema manager, Instance manager로 구성된다. Schema validator는 XML 스키마 문서와 XML 인스턴스 문서를 입력으로 하여, 문서들의 유효 적합성을 점검한다. 스키마 문서는 단일의 문서로 구성될 수도 있으며, 임포트(include)/인클루드(include)/재정의(redefine) 매커니즘을 사용하여 여러 개의 스키마 문서가 복잡하게 연결될 수도 있다[3]. XML 인스턴스 문서는 스키마 문서에 항상 유효(valid) 해야 한다. 따라서 스키마 문서의 정의가 올바르지 않거나 인스턴스 문서가 스키마 문서에 유효하지 않은 경우, 압축 작업을 수행하지 않는다.

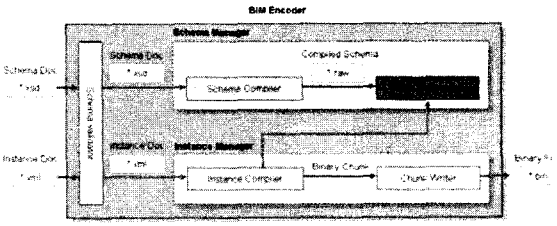

그림 1. BiM 압축기의 구조

Schema manager는 스키마 문서를 입력으로 받아 SAX Parser를 사용하여 파싱한다. 파싱 과정을 통하여 스키마의 정의 내용을 내부 자료 구조로 재정의하고, 스키마 문서의 각 컴포넌트를 기호화하여 간소화시킨 raw 파일을 만든다. 생성된 raw 파일을 사용하여 FSA(Finite State Automata)를 구성한다.

Instance manager는 인스턴스 문서를 schema manager에서 생성한 FSA를 사용하여 압축하는 작업을 수행한다. 압축된 이진 청크(binary chunk)는 chunk writer가 바이너리 파일 형태로 출력한다.

BiM 복원기의 구조는 그림 2와 같다. 그림 2의 BiM 복원기는 크게 schema manager와 binary manager로 구성된다. BiM 복원기는 스키마 문서 또는 raw 파일, 및 이진 압축된 인스턴스 문서를 입력으로 한다. Schema manager는 그림 1의 BiM 압축기에서 언급한 것과 동일하다. 단, 입력 파일로 BiM 압축기에서 이미 생성한 raw 파일을 입력으로 받았을 경우에는 스키마 컴파일 작업은 수행하지 않고 바로 FSA를 생성한다. 결과적으로, 복원 시 raw 파일이 주어질 경우에는 BiM 압축기에서 이미 수행한 동일한 작업을 중복 수행을 하지 않아도 되는 이점이 생긴다.

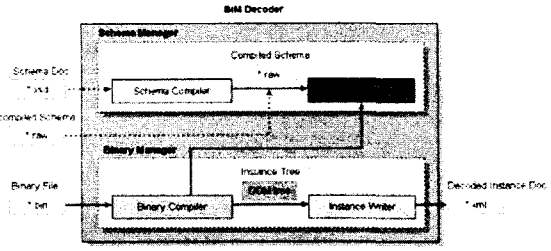

그림 2. BiM 복원기의 구조

Binary manager는 schema manager에서 생성한 FSA를 사용하여 이진 압축된 인스턴스 문서의 복원 과정을 수행한다. 복원의 결과로 메모리에 인스턴스 문서의 DOM tree 구조가 재구성된다. 이 DOM tree를 instance writer를 통해서 원래의 인스턴스 문서로 출력한다.

3. BiM 압축/복원기의 구현

여기서는 2장에서 설계한 BiM 압축기 및 복원기의 세부 구현 및 모듈간의 상호 동작에 대해 기술한다. 본 논문에서 구현한 BiM 압축/복원기는 Linux Fedora Core 4.0 (64 bit) OS 환경에서 C++로 구현하였다. 설계한 모듈 중 schema validator는 압축 과정을 수행하기 위한 전처리 과정으로서, Apache사에서 개발한 Xerces 파서 (v2.7.0)를 사용하였다. 따라서 여기서는 설계한 MPEG-7 BiM 압축/복원기 모듈 중 schema manager, instance manager, binary manager에 대하여만 기술한다. 본 장에서는 구현한 BiM 압축/복원기의 자료 구조 및 동작을 구현한 클래스 계층구조를 중심으로 기술한다. 여기서 기술하는 클래스 계층구조는 실제로 구현한 내용 중 중요 부분만을 요약한 것이다.

3.1. Schema manager의 구현

Schema manager는 BiM 압축/복원기의 공통 모듈로서, 스키마 문서를 파싱하여 문서의 내용을 내부 구조로 표현하고, raw 파일을 출력하는 schema compiler와 FSA를 생성하는 FSA constructor로 구성된다. Schema compiler가 파싱한 스키마 문서를 내부 구조로 표현하기 위해서 본 논문에서 설계한 클래스 계층구조는 UML 다이어그램으로 표현한 그림 3과 같다.

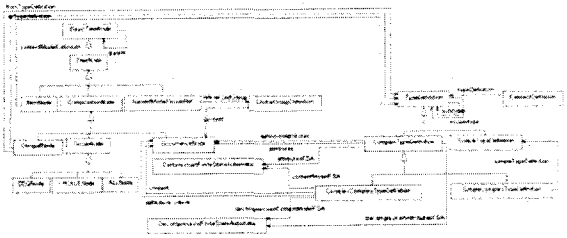

그림 3. 스키마 문서 표현을 위한 클래스 계층구조

스키마를 정의하기 위한 핵심 스키마 컴포넌트 요소인 타입은 크게 복합 타입(complex type)과 단순 타입(simple type)로 분류된다.[3] 스키마 문서를 표현하기 위한 클래스 계층 구조의 최 상위 클래스로 TypeDefinition을 정의하고, 타입의 분류에 따라 ComplexTypeDefinition과 SimpleTypeDefinition을 하위 클래스로 나누었다. 그리고 ComplexTypeDefinition은 타입 상속 방법에 따라, complexContent 요소를 사용하여 확장하는 ComplexContentTypeDefinition과 simpleContent 기법을 사용하여 정의하는 SimpleContentTypeDefinition을 나누어 설계하였다. 이외에 컴포지터(compositor)와 지명 모델 그룹(named model group)을 위한 클래스도 정의하였다.

예를 들어, 그림 3의 클래스들 중 ComplexTypeDefinition 클래스의 정의는 다음과 같다.

```
class ComplexTypeDefinition : public TypeDefinition {
  bool isAbstract;
  bool isMixed;
  OccurrenceNode *attributes;
  OccurrenceNode *content;
  CompressionFSA *compressionAttributesFSA;
  DecompressionFSA *decompressionAttributesFSA;
  CompressionFSA *compressionContentFSA;
  DecompressionFSA *decompressionAttributesFSA;
  bool hasAttributes();
  void setAbstract(bool);
  bool getAbstract();
  void generateFSA();
  void setMixed(bool);
  void getMixed();
  void generateFSA();
  FiniteStateAutomata* generateAttributesFSA();
  FiniteStateAutomata* generateContentModelFSA();
  void realize(TypeDefinitions *);
  . . .
}
```

ComplexTypeDefinition 클래스는 위에서 설명한 것과 같이, 복합 타입의 정의를 나타낸다. 만약, 복합 타입의 " abstract" 속성이 " true" 일 경우에는, " isAbstract" 변수를 true로 설정한다. 그리고, " mixed" 속성을 사용할 경우, " mixed" 변수 값을 true로 설정한다. 이 후에, FSA constructor 모듈에서 generateFSA() 메소드를 사용하여, 현재 타입에서 정의한 애트리뷰트와 내용 모델의 FSA를 구성한다. 이를 위해 generateAttributeFSA()와 " generateContentFSA() 메소드를 호출한다.

이와 같이 schema manager가 내부적으로 사용하는 자료 구조를 갖게 함으로서, 후에 압축 및 복원 과정에서 XML 스키마 문서의 정의 내용을 얻기 위해 다시 파싱하는 과정을 거치지 않고, 효율적으로 스키마 문서의 구조를 접근 할 수 있도록 한다.

Schema compiler는 그림 3에서 정의한 자료 구조를 구성하고 입력 스키마 문서를 단순화 시킨 raw 파일을 생성한다.

FSA constructor는 생성된 raw 파일을 사용하여 FSA를 구성한다. MPEG-7 BiM 스펙에서 기술된 FSA의 구조를 먼저 살펴보면, FSA는 그림 4와 같이 상태(state)와 전이(transition)로 구성된다[4]. 상태에는 타입 상태(type state)가 있으며, 전이는 루프 전이(loop transition)와 코드 전이(code transition)로 나누어진다. 루프 전이에는 루프 시작 전이(loop start transition), 루프 연속 전이(loop continue transition), 루프 끝 전이(loop end transition)가 있고, 코드 전이에는 전환 전이(shunt transition)가 있다. FSA의 진행 과정은 토큰(token)이 시작 상태(start state)에서부터 끝 상태(end state)로 이동하면서 이루어 진다. 이 때, 토큰이 현재 상태에서 임의의 다른 상태로 이동하는 것을 " crossed" 라 하고, 토큰이 임의의 상태로 " crossed" 되면 해당 상태가 " activated" 되었다고 한다. " crossed" 와 " activated" 를 반복하면서 타입 상태가 " activated" 되면, 내부적으로 해당 FSA 객체를 호출함으로써, 점진적으로 압축 과정을 수행한다.

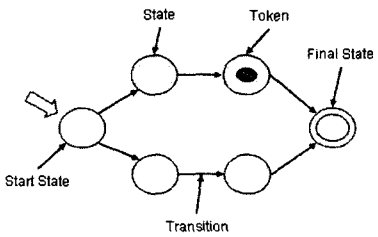

그림 4. Finite state automata의 구성

본 논문에서는 위와 같은 FSA의 동작을 그림 5와 같은 UML 다이어그램으로 표현한 클래스 계층구조로 구현하였다. FSA는 기능에 따라 압축 FSA 및 복원 FSA로 나누고, FSA의 구성요소인 상태와 전이는 각각 State와 Transition 클래스로 구현하며, 상태와 전이에 관련된 데이터들은 각각 연결 리스트 구조로 표현하였다.

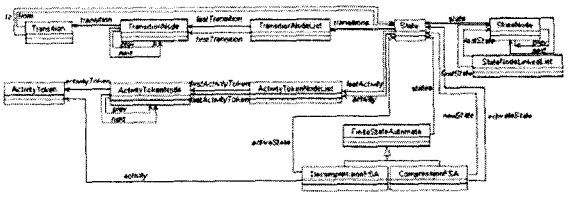

그림 5. Finite state automata 구성을 위한 클래스 계층구조

예를 들어, FiniteStateAutomata 클래스의 정의는 다음과 같다. FiniteStateAutomata 객체는 그림 3의 ComplexTypeDefinition 객체가 generateFSA() 메소드를 호출함으로써 생성된다. FiniteStateAutomata 클래스는 상태 객체를 생성하여 states 리스트에 상태를 추가하고, 상태와 상태를 전이로 연결 하거나 다른 임의의 오토마타의 상태들과 현재 상태들을 병합하는 과정을 수행한다. 이와 같이, 상태와 전이를 생성하고 병합하는 과정을 raw 파일에서 표현된 스키마 컴포넌트의 정의 순서에 따라 반복적으로 수행하면서 FSA가 구성된다.

```
class FiniteStateAutomata {
  StateLinkedList *states;
  void addState(State *);
  void merge(FiniteStateAutomata *);
  void merge(State *, State *);
  State* getFirstStartState();
  State* getFirstFinalState();
  StateLinkedList* getStartState();
  StateLinkedList* getFinalState();
...
}
```

이러한 FSA 클래스 계층구조를 사용하여, 압축 시에는 인스턴스 raw 파일에 명시된 엘리먼트의 바인딩 타입의 타입 상태 노드를 이동하면서 이동 경로인 전이의 값을 이진 스트링으로 변환한다. 반대로 복원 시에는 이진 스트링의 값으로 상태를 이동하며 해당 타입 상태의 정의를 도출하여 DOM tree를 재구성한다.

3.2. Instance manager와 binary manager의 구현

BiM 압축/복원기의 Instance manager와 binary manager는 그림 6에서 정의한 클래스 계층 구조를 사용한다. TypeEncoder 클래스를 기반으로 타입의 특성에 따라 SimpleTypeInstance와 ComplexTypeInstance 클래스로 나누며, ComplexTypeInstance 클래스는 타입의 상속 방법에 따라 SimpleContentTypeInstance와 ComplexTypeInstance 클래스로 세분화하여 정의한다.

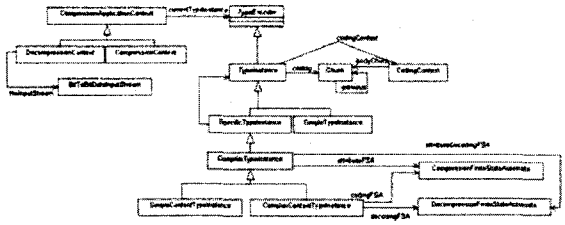

그림 6. instance manager와 binary manager를 위한 클래스 계층구조

예를 들어, 그림 6의 클래스들 중 TypeInstance와 ComplexTypeInstance 클래스의 정의는 아래와 같다.

```
class TypeInstance : public TypeEncoder {
  void encodeTypeInfo();
  void decodeTypeInfo();
  . . .
}
class ComplexTypeInstance : public TypeInstance {
  void startEncoding();
  TypeEncoder* encodeContent(const XMLCh *);
  TypeEncoder* encodeAttribute(const XMLCh *);
  void endEncoding();
  void endContentEncoding();
  void endAttributeEncoding();
  void startDecoding(BitToBitDataInputStream *);
  TypeEncoder* decode(BitToBitDataInputStream *);
  void endDecoding();
. . .
}
```

Instance manager는 스키마 문서를 FSA를 사용하여 압축하는 모듈로서, TypeInstance의 하위 클래스의 객체가 startEncoding() 메소드를 호출함으로서 압축 과정을 시작한다. 복합 타입의 경우에는 내용 모델과 애트리뷰트를 압축하기 위해 각각 encodeContent()와 encodeAttribute() 메소드를 호출한다. 각 메소드는 FSA의 시작 상태에서 끝 상태로 이동하면서 타입 정의에 따라 TypeInstance 클래스의 encodeTypeInfo() 메소드를 수행하여, 압축된 이진 스트링을 Chunk 객체로 저장한다. 이와 같은 반복적인 과정을 수행한 후에, FSA의 끝 상태에 도달하면 endEncoding() 메소드를 호출하고, chunk writer는 생성된 Chunk 객체들을 전달 받아 이진 파일 형태(*.bin)로 출력한다.

Binary manger는 FSA를 사용하여 이진 스트링을 인스턴스 문서로 복원하는 모듈이다. 이진 파일을 읽어 BitToBitDataInputStream 객체를 생성하고, TypeInstance 클래스의 하위 클래스의 객체가 startDecoding() 메소드를 호출하면서 복원이 시작된다. 복합 타입의 경우에는 decode() 메소드를 호출하여 애트리뷰트와 내용 모델을 FSA를 사용하여 복원한다. 즉, BitToBitDataInputStream 객체의 이진 스트링의 값에 따라 FSA의 시작 상태에서 끝 상태로 이동하면서, TypeInstance 클래스의 decodeTypeInfo() 메소드를 실행하여 타입의 정의를 DOM tree 형태로 재구성한다. 동일한 반복 과정 후에, FSA의 끝 상태에 도달하여 endDecoding() 메소드가 호출되면, instance writer가 구조화된 DOM tree 구조를 인스턴스 문서(*.xml)로 출력한다.

4. 결 론

본 논문은 스키마 문서에 유효한 인스턴스 문서를 MPEG-7 BiM 표준에 맞게 이진 압축하는 BiM 압축기, 그리고 이진 복원된 입력 파일에서 역으로 인스턴스 문서를 얻는 BiM 복원기의 구조를 설계하고, 설계된 압축기와 복원기를 구현하였다. 구현한 BiM 압축/복원기는 MPEG 인스턴스 문서 뿐만 아니라 XML 스키마로 표현되는 스키마 문서에 따르는 어떤 인스턴스 문서도 압축할 수 있는 범용 XML 문서 압축/복원기이므로, XML을 활용하는 모든 분야에서 활용될 수 있다. 특히, 대역폭의 제약이 심한 방송 분야에서 메타데이터 전송을 위한 핵심 모듈로서 활용 가치가 매우 높을 것으로 예상된다.

현재 구현된 시스템은 아직 최적화 되어 있지 않다. 특히 복원기는 매우 작은 메모리와 낮은 성능의 CPU가 장착된 휴대형 단말기에서 실행되는 경우가 많을 것이므로, 최적화에 특히 많은 노력이 필요하다. 앞으로는 이를 위한 연구가 필요할 것이다.

참고문헌

[1] Liefke, H., Suciu D., " Xmill:an efficient compressor for XML data" , Proceedings of SIGMOD Conference, 2000

[2] TV-Anytime Forum, " TV-Anytime Phase 1, Part 3 : Metadata, Sub-part2 : System aspects in a uni-directional environment" , ETSI Standard, ETSI TS 102 822-3-2, V.1.3.1 Jan. 2006

[3] Henry S. Thompson, David Beech, Murray Maloney, Noah Mendelsohn, " XML Schema Part 1 : Structures Second Edition" , Oct. 2004, Available as http://www.w3.org/TR/xmlschema-1/

[4] 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 (m7673)