

Java기반의 MPEG-4 시스템 구현

강 기 정[†] · 홍 충 선^{††} · 이 대 영^{†††}

요 약

본 논문에서는 무선 랜 기반의 유무선 통합서비스 사업을 위해 동영상 멀티미디어 콘텐츠를 포함한 멀티미디어 메시징 서비스를 제공하기 위해 자바와 자바 3차원 기반의 MPEG-4 표준 규약에 충실한 MPEG-4 시스템 구현에 관한 것이며, 이를 위해 MPEG-4 시스템 구현에 필요한 기술, 유무선 통합 멀티미디어 서비스의 정의, DMIF 구현 및 mp4 파일 파싱 등의 구체적인 MPEG-4 시스템 구현 방법 등을 기술한다.

An Implementation of Java based MPEG-4 System

Ki Joung Kang[†] · Choong Seon Hong^{††} · Dae Young Lee^{†††}

ABSTRACT

In this paper, an implementation example of Java based MPEG-4 system that follows MPEG-4 standard protocols in order to provide multimedia-messaging service is introduced. The multimedia-messaging service is a wireless LAN based wired and wireless service that delivers multimedia contents including video and audio information. Detailed Methods to develop a MPEG-4 system like technology of MPEG-4 system implementation, definition for wired and wireless multimedia service, DMIF implementation, and mp4 file parsing are described.

키워드 : 자바(Java), MPEG-4시스템(MPEG-4 System), 유무선(Wired and Wireless), 멀티미디어 서비스(Multimedia Services), PDA

1. 서 론

인터넷 및 무선 통신이 보편화되어 감에 따라 Desktop-PC, PDA, Hand-Held PC 등과 같은 다양한 단말기를 통해 멀티미디어 데이터 서비스를 받고자 하는 사용자의 요구가 증가되고 있다. 또한 멀티미디어 콘텐츠의 대화형 서비스에 대한 요구가 새로운 이슈로 부각되고 있다. 멀티미디어 데이터에 사용자 상호작용이 추가된 형태의 미디어를 보통 리치(Rich) 미디어라고 부른다. 이러한 리치미디어 데이터를 효율적으로 제작, 관리, 전송하고 지원해야 할 필요성에 의해 개발된 것이 MPEG-4 표준이다.

MPEG-4(ISO/IEC 국제표준 14496)는 오디오, 비디오, 합성 오디오, 그리고 그래픽스 요소(material)로 구성된 복잡한 장면(scene)을 표현하고, 이를 통신라인을 통해 사용자와 상호작용이 가능하도록 해주는 멀티미디어 시스템을 정의하는 표준규약을 말한다. MPEG-4 시스템은 이러한 표준규약을 구현한 멀티미디어 시스템이다[1, 2].

MPEG-4에서 가장 중요한 요소 중 하나로서 QoS(Quality

of Service)를 들 수 있는데, 이는 다양한 통신 플랫폼이나 단말기에 맞는 서비스를 제공하여 그 응용분야를 확장하게 한다. 이를 규정하기 위해 다양한 프로파일(Profile)들을 제시하고 있는데, 이는 비디오, 오디오, 그래픽스등의 각 매체에 프로파일 레벨을 정의하고 해당 통신 플랫폼이나 단말기에 맞는 압축 방식이나 장면 구성 방법을 정의하고 있다 [1-3, 6, 7].

또 하나의 중요한 요소는 사용자 상호작용의 지원이다. 이는 기존의 멀티미디어 서비스가 주로 단 방향의 매체를 지원하고 있는데 반해, 쌍방향의 통신이 가능도록 하여 사용자의 욕구가 수시로 반영될 수 있도록 하는 것이다. 위의 두 요소를 만족시키게 되면 디지털 방송, 게임, 애니메이션 등 아주 광범위한 응용이 MPEG-4시스템을 통해 구현된다.

현재 개발되어 있는 MPEG-4시스템을 살펴보면 먼저 IBM을 들 수 있다. MPEG-4시스템에서는 비디오 인코더를 제외한 전체를 자바기반으로 구현하여 웹에서 공개하였으나 MPEG-4 컨소시엄과의 라이선스 문제로 인하여 비공개하기에 이르렀다. 현재는 IBM[3]이 구현한 시스템의 성능이나 기능이 어느 정도까지 인지는 정확하게 파악이 되지 않고 있다. 그 다음으로 Envivio[4]를 들 수 있다. 여기에서는 QuickTime의 플러그-인(Plug-in)형태로 MPEG-4시스템을 개발하였는

† 정 회 원 : 경희대학교 대학원 전자공학과
 †† 종신회원 : 경희대학교 전자정보학부 교수
 ††† 정 회 원 : 경희대학교 전자정보학부 교수
 논문접수 : 2002년 7월 24일, 심사완료 : 2002년 10월 22일

데 주로 2D만 구현되어 있다. 스트리밍(Streaming) 서버와 저작도구도 함께 개발하였다. Blaxxun[5]은 2D, 3D등을 모두 구현하였는데 비디오 데이터는 플레이가 되지 않는 등 완성도가 떨어진다. 그밖에 Philips등이 있는데 이들은 주로 비디오와 오디오를 지원하는데 지나지 않고, BIFS(Binary Format for Scene)를 지원하지 못하는 등 아직은 MPEG-4 표준을 따르지 못한다.

본 논문에서는 유무선 통합 환경에서 응용성이 뛰어난 국제 표준 규격인 MPEG-4시스템의 구현을 자바 언어를 사용했다. 주지하는 바와 같이 Java는 플랫폼과는 독립적인 언어로써 이식성이 매우 높다. 또한 Java2D와 Java3D는 BIFS에 기술되는 장면 그래픽을 구현하는 아주 효율적인 그래픽스 API이기 때문이다.

2. MPEG-4 시스템

2.1 MPEG-4

MPEG-4는 1993년 7월 표준화가 시작되어 1999년 11월에 Committee Draft가 나왔고, 현재 버전 1은 거의 표준화가 종결되었다. MPEG-4는 기존의 MPEG 표준들을 결합하고 다음과 같은 3가지 요구를 반영하는 새로운 표준이다. 첫째 저작자의 요구로써, 디지털 TV나 애니메이션, WWW 등과 같은 개별적인 기술보다는 이들을 통합하여 적용적이고 재사용도가 높은 콘텐츠의 제작에 대한 요구이다. 물론 여기에는 기존의 방식보다 콘텐츠에 대한 저작권의 보호와 관리가 용이하게 되어야 한다.

둘째로는 네트워크 서비스 제공자의 요구로써, 정확한 정보의 제공이다. QoS와 같은 네트워크나 단말기 특성에 따라서 서비스의 품질을 조정하는 방법도 포함한다.

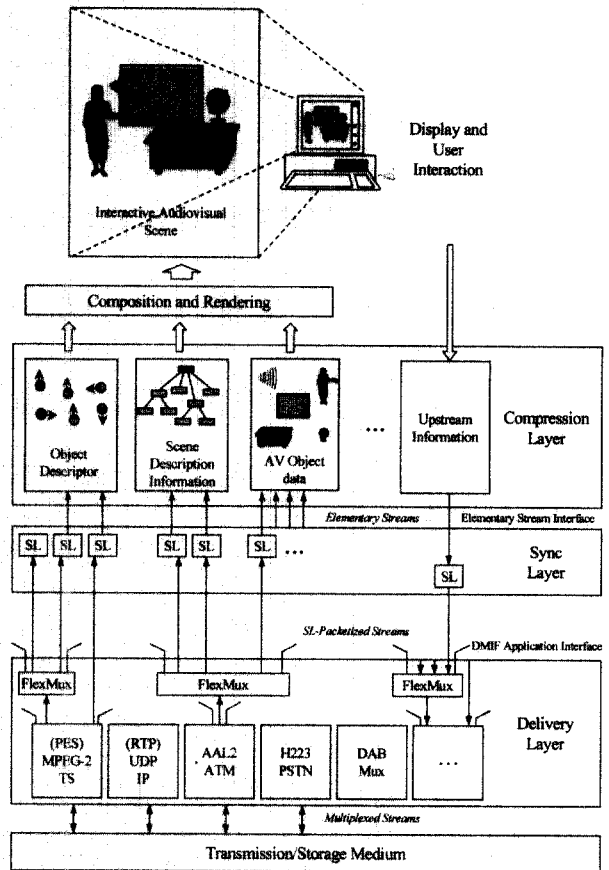
셋째로는 사용자의 요구로써, 하나의 단말기를 통해 다양한 기능에의 접근과 콘텐츠와의 높은 상호작용에 대한 요구이다.

위와 같은 요구를 만족시키기 위해서 MPEG-4에서는 Coding, Composition, Multiplex, Interaction에 대한 제안을 기술하고 있다[6].

2.2 MPEG-4 시스템

MPEG-4시스템은 복합적으로 구성된 각 개체들을 저장, 전송 및 표현하기 위한 시스템으로서 (그림 1)과 같이 구성되어 있다. 로컬 스토리지(storage)나 통신선로를 통해 데이터가 수신되면 전송 계층(Delivery Layer)에서는 패킷화 되어 전송되어온 된 데이터를 각종 미디어 타입별로 분류해서 동기 계층(Sync Layer)에 건네주게 된다. 동기 계층에서는 여기에 타임-스탬프(Time-Stamp)를 붙여 합성시 참고할 수 있도록 SL-packet형태로 만들어서 압축 계층(Compression Layer)에 건네주고 복호화를 할 수 있도록 한다.

합성 계층(Composition Layer)에서는 객체 디스크립터(Object Descriptor), 장면 디스크립션(Scene Description), AV 데이터등 각 스트림별로 해당 복호화나 파서를 통해서 복호화를 하게 되고 이를 각각에 붙어있는 타임-스탬프를 고려해서 적절한 시간에 합성을 함으로써 전체적인 장면이 구성되게 된다[7, 8].



(그림 1) MPEG-4 시스템 구조도

MPEG-4시스템은 크게 3부분으로 나눌 수 있는데, DMIF (Delivery Multimedia Integration Framework), Decoder, Composition과 Rendering부분 등이다. <표 1>에서는 제안된 시스템과 현재 구현된 여러 시스템간의 2D, 3D 및 스트리밍 처리 여부에 대한 비교를 해 보았다. 표에서 보는 바와 같이 다른 시스템들과 비교해 볼 때 제안된 시스템의 성능이 매우 양호한 것으로 분석이 되고 있다[9].

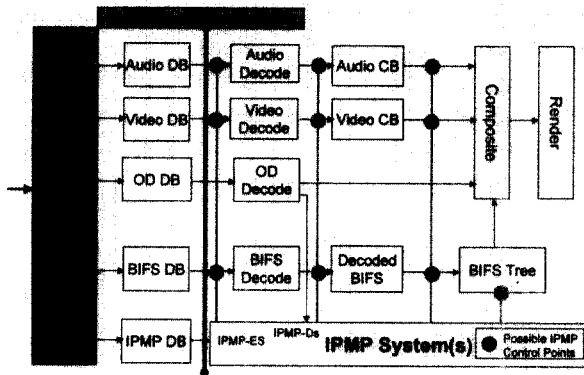
<표 1> MPEG-4 시스템 비교

분류	2D 그래픽스	3D 그래픽스	스트리밍
Envivio	●	×	●
Blaxxun	●	●	×
IBM	●	●	●
제안된 시스템	●	●	●

(● 지원, × 지원하지 않음, ● 일부만 지원)

2.2.1 DMIF

DMIF는 로컬 하드 디스크나 네트워크상의 멀티미디어 데이터를 읽어 들이는 곳으로 가장 중요한 개념은 정확성을 보장하는 것이다. 즉 전송계층에서 모든 데이터가 하나의 스트림 형태로 보이도록 보장해 주는 것이다. 로컬 파일인 경우 Quick Time포맷을 채용한 MP4 파일을 파싱하고, 이를 스트림 형태로 만들어서 위의 복호화 버퍼(Decoder Buffer)에 넣어주어야 한다. 그리고 네트워크를 통해서 오는 데이터는 네트워크 프로토콜을 통해 데이터를 받고, 이를 미디어 타입별로 분류해서 복호화 버퍼에 넣어 전송 계층에서 복호화할 수 있도록 한다[10].



(그림 2) DMIF, 복호화 및 합성

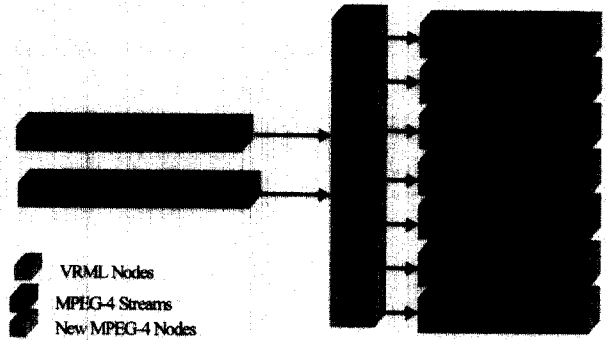
2.2.2 Decoder

복호화는 미디어 타입별로 하게 되는데 스트리밍을 해야 할 경우 데이터가 지연되는 경우가 발생하기 때문에 이를 보완하기 위해서 복호화 버퍼를 두고 이를 완충하여 복호화를 하게 된다. 복호화에는 비디오, 오디오, BIFS, 그리고 객체 디스크립터에 대한 복호화들이 있다. 또한 ImageTexture 등과 같은 노드에서 필요한 JPEG 이미지를 위한 복호화 등도 필요하다.

2.2.3 Composition과 Rendering

장면은 BIFS(Binary Format for Scene)에 의해서 기술된다. 합성은 BIFS에 의해서 기술된 장면을 파싱하고 렌더링하는 과정이라 할 수 있는데, 여기에 비디오와 오디오 그리고 그래픽 객체를 시간 축에 대해서 결합하는 기술이 들어있다. 따라서 BIFS를 파싱하고 이의 문법에 맞게 장면을 구성하고 이를 렌더링하는 과정이 합성의 과정이라 할 수 있다.

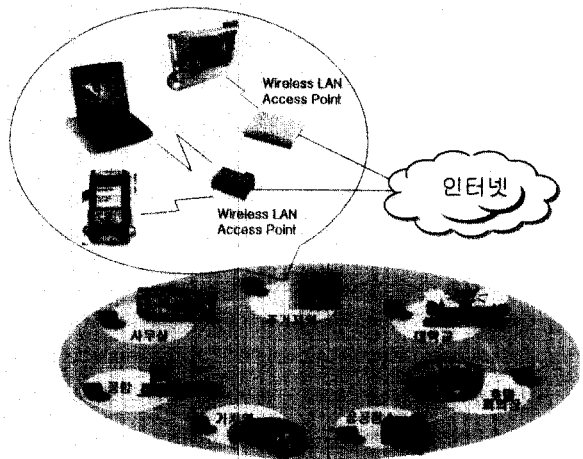
BIFS의 문법은 인터넷 3차원 가상현실 표준인 VRML에 따른다[5]. 물론 VRML 문법 외에 여러 개체들을 포함하고 있고, 여기에 추가적으로 2D그래픽스를 위한 노드들이 추가되어 있다. 현재 BIFS에 기술된 노드들은 (그림 3)에서 보는 바와 같이 VRML2.0에 들어있는 노드 외에 2D Nodes, Audio Nodes, FBA 등 전체적으로 114개의 노드가 있다[11, 12].



(그림 3) BIFS 노드

3. 유무선 통합 멀티미디어 서비스

유무선 통합서비스는 무선 LAN이 구축된 잠재적 이용빈도가 높은 지역에서 무선 LAN기반의 PC, PDA를 이용하여 인터넷 접속 및 유무선 PDA포탈 사이트에서 제공되어지는 서비스이다. PDA는 CDMA 및 GPS기반의 서비스를 받을 수 있으므로, 핫 스팟 이외의 지역에서도 LBS서비스 및 저속 데이터 서비스를 제공 받을 수 있다. 이밖에도 무선 메시징 서비스와 다양한 콘텐츠를 즐길 수 있는 PDA 전용 포털 사이트를 구축하여 PDA단말기를 통해 인터넷 접속, 일정관리, 메신저, 주소록, 채팅, 멀티미디어 화상 통화 서비스, SYNC 및 뉴스, 교육, 게임 등 콘텐츠 서비스를 제공한다[13-15].



(그림 4) 유무선 통합 망 구성도

유무선 통합 멀티미디어 서비스는 (그림 4)에서 보듯이 공항터미날, 주거단지, 학교, 공공건물 등 인구 밀집지역에 무선랜 접속장치(Wireless LAN Access Point)를 설치하고 노트북, PDA, Hand-held PC 등 같은 단말기를 통해 초고속 인터넷에 접속할 수 있는 서비스를 제공하는 것을 목적으로 한다. 이를 활성화시키기 위해 MMS(Multimedia Messenger Service), VOD 등과 같은 고품질의 멀티미디어 부가

서비스의 제공도 추진하고 있다. 무선 LAN은 현재 11Mbps 제공하는 802.11b 규격에서 54 Mbps를 제공하는 802.11a로 발전해 가고 있으며, 서비스는 무선 인터넷 상에서도 멀티 미디어를 제공하는 방향으로 진화하고 있다. 영상메일 서비스의 근간이 되는 MMS는 MPEG-4 표준을 채용하고 있기 때문에 MPEG-4 시스템의 구현이 필요하다.

4. 구현 및 고찰

본 장에서는 주로 mp4 파일을 파싱하여 미디어 스트림을 추출하는 DMIF와 그리고 BIFS를 파싱하여 이로부터 비디오, 오디오, 그리고 그래픽 객체를 합성하여 전체적인 Scene을 구성하는 과정, 그리고 구현 시 고려해야 할 중요한 점들에 대해서 기술한다.

구현된 시스템의 전체적인 구조는 (그림 5)와 같다. MPEG-4 콘텐츠가 비트 스트림(Bit Stream)형태로 DMIF 인터페이스를 통해서 들어오면 MP4 파서가 이를 파싱하여 각 미디어 타입별로 분류한다. 제일 먼저 IOD(Initial Object Descriptor)를 파싱하여 복호화에 필요한 정보와 합성에 필요한 정보를 얻는다. 그 다음 BIFS 트랙을 찾아서 장면 구성에 필요한 BIFS 데이터를 얻어서 파싱을 하여 장면 그래프를 구성하게 된다. 이때 BIFS의 노드 형태로 기술되는 MovieTexture, AudioSource, ImageTexture 노드가 들어가 있을 경우, 비디오, 오디오, 이미지 각각의 복호화를 통해서 해당 데이터를 얻는다. 이들 데이터는 전체적인 장면 합성을 담당하는 장면 합성기의 제어 하에서 장면 그래프에 반영된다. 장면 그래프의 업데이트 시간은 시스템 클럭(System Clock)이나 마우스, 키보드등에서 발생하는 이벤트, 혹은 TimeSensor, TouchSensor, ProximitySensor, PlaneSensor 등과 같은 센서들에서 발생한 이벤트에 따라서 결정된다.

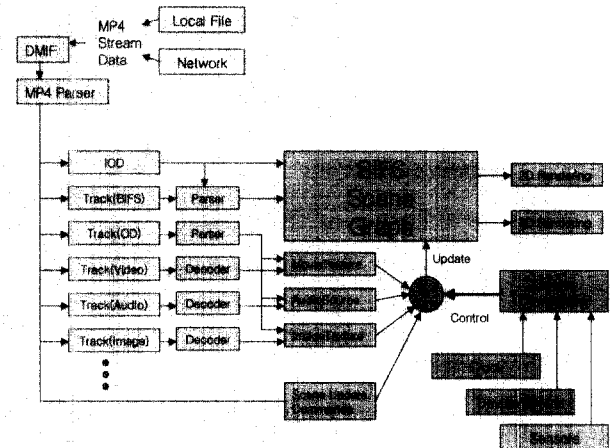
장면 업데이트 명령어는 조건노드나 스트리밍 과정에서 직접 도달할 수 있는 것으로 기존에 구성된 장면의 일부를 업데이트 하거나 전부를 업데이트 한다. 장면 업데이트도 이벤트에 따라서 그 업데이트 시기가 결정된다.

구성된 장면은 주기적으로 렌더링 과정을 통해서 모니터에 디스플레이 되는데 본 시스템에서는 2D와 3D를 구별하여 렌더링 하도록 하였다. 2D의 경우 Java2D를 사용하게 되고, 3D의 경우는 Java3D를 사용하게 된다. 3D의 경우 BIFS 장면 그래프를 Java3D의 장면 그래프로 바꾸는 과정이 필요하고, 2D 렌더링의 경우 원래 구성된 BIFS 장면 그래프를 사용하게 된다.

렌더링 시기는 BIFS 노드를 파싱 할 때 결정된다. 즉 BIFS 장면 그래프의 루트 노드가 그룹이나 Layer3D 노드로 시작하면 3D 문법에 따라 모든 노드의 필드를 해석하고 3D 그래픽스 렌더링을 통해서 장면을 디스플레이하고, OrderedGroup이나 Layer2D 노드로 시작되면, 2D 문법에 따라

서 장면 그래프를 해석하고 렌더링 해야 한다.

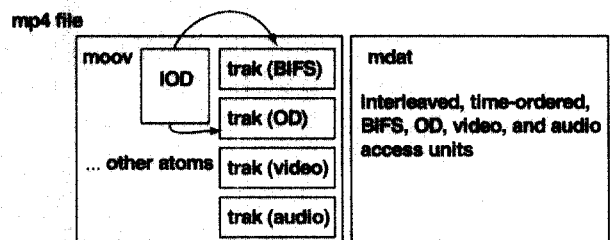
BIFS 장면 그래프에 2D 노드와 3D 노드가 혼합되어 있으면, 2D와 3D 렌더링을 동시에 해야 한다. 이 경우 2D와 3D 렌더링 과정이 개념적으로 많은 차이를 보이고 있어 동일한 방법으로 렌더링을 할 수 없기 때문에 렌더링 속도가 늦어지는 문제가 발생하지만 본 논문에서는 이러한 문제점을 해결하였다.



(그림 5) 구현된 MPEG-4 시스템 구조

4.1 DMIF 구현 및 mp4 파일 파싱

기본적으로 MP4 파일은 (그림 6)에서 보는 바와 같이 Apple사에서 개발한 QuickTime format을 채용하고 있다. MP4 파일은 Atom과 객체 디스크립터로 구성되어 되어 있다. 모든 정보는 Atom이라 불리는 단위 컨테이너에 들어간다. Atom은 데이터의 관련된 헤더 정보를 가지는 moov(Movie Atom)과 실제 미디어 데이터가 들어가는 mdat(Media Data Atom) 등 두 개로 나누어 볼 수 있다. moov에는 IOD(Initial Object Descriptor)라는 헤더 정보가 있는데, 이는 mp4 파일 전체를 파싱하는데 필요한 정보들을 가지고 있다. 그 다음 트랙이 오는데, 이는 각 미디어 형태별로 하나의 트랙이 오게 된다, 그림에서는 BIFS, OD, Video, Audio가 미디어로 올 경우 4개의 트랙이 존재함을 보여준다.



(그림 6) mp4 파일 구조

다음은 mp4 샘플파일에 대해서 파싱한 결과를 보여주고 있는데, Atom들의 계층을 자세히 보여주고 있다.

```

Mp4MovieAtom(moov) {
  Mp4MovieHeaderAtom(mvhd) { }
  Mp4ObjectDescriptorAtom(iods) {
    MP4_IOD {
      Es_Id_IncDescriptor { }
      Es_Id_IncDescriptor { }
    }
  }
  Mp4TrackAtom(trak) -----
  Mp4TrackHeaderAtom(tkhd) { }
  Mp4MediaAtom(mdia){
    Mp4MediaHeaderAtom(mdhd) { }
    Mp4MediaInformationAtom(minf) {
      Mp4DataInformationAtom(dinf) {
        Mp4DataReferenceAtom(dref) {
          Mp4DataEntryURLAtom(url) { }
        }
      }
    }
    Mp4SampleTableAtom(stbl) {
      Mp4TimeToSampleAtom(stts) { }
      Mp4SampleDescriptionAtom(stsd) {
        Mp4VisualSampleEntryAtom(mp4v) {
          Mp4EsdAtom(esds) {
            ES_Descriptor {
              DecoderConfigDescriptor {
                DecSpecificInfoDescriptor { }
                I got decoderSpecificInfo
              }
              SLConfigDescriptor { }
            } // end of es_descriptor
          }
        }
      } // end of sample table atom
      Mp4SampleSizeAtom(stsz) { }
      Mp4SampleToChunkAtom(stsc) { }
      Mp4ChunkOffsetAtom(stco) { }
      Mp4CompositionOffsetAtom(ctts) { }
      Mp4SyncSampleAtom(stss) { }
    }
    Mp4VideoMediaHeaderAtom(vmdh) { }
  }
  Mp4HandlerAtom(hdlr) { }
}
} // end of moov

```

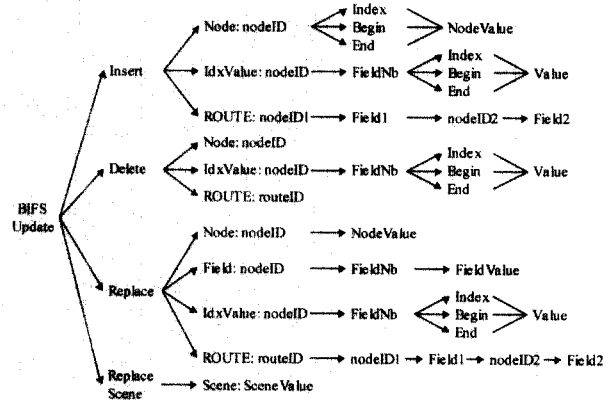
moov에 이어 실제 미디어 데이터인 mdat가 뒤따라 온다. 미디어 데이터가 어디에 있으며 어떻게 복호화해야 하는가 각 트랙에 딸린 ES_Descriptor에서 기술한다. 각 트랙에 실린 해당 미디어를 복호화 할 때 필요한 정보는 DecoderConfigDescriptor에 실어서 보내게 되는데 여기에는 사이즈, 포맷 헤더등과 같은 정보들이 실려서 오게 된다. 예를 들면 BIFS의 경우 캔버스 너비 및 높이, 노드 코딩 비트수 같은 정보가 들어 있어서, BIFS 파서와 장면 합성기에서 사용할 수 있도록 되어있고, 오디오나 비디오 등의 경우는 각각의 복호화를 초기화하는데 사용할 수 있다.

4.2 BIFS 파서의 구현

기존의 VRML(Virtual Reality Modeling Language) 파일이 utf8로 인코딩 된 텍스트 파일인데 비해 BIFS는 이진(binary)으로 코딩되어 있다. 따라서 기존의 Lex나 Yacc를 써

서 데이터를 토큰화 하는 방식은 불가능하다. 비트 단위로 데이터를 읽어서 파싱을 해야 하기 때문에 case by case 형태의 파싱 만이 가능하다.

BIFS는 (그림 7)에서 보는 바와같이 4가지 명령을 통해서 장면이 구성되고 업데이트가 이루어진다. 특히 초기의 장면은 SceneReplacementCommand에 의해서 구성된다.



(그림 7) BIFS의 명령구조

다음은 구현한 파서의 대략적인 흐름을 나타내고 있다.

```

void parseScene() {
  do {
    int code = decoder.parseInt(2);
    switch(code) {
      case 0:
        parseInsertionCommand(); break;
      case 1:
        parseDelectionCommand(); break;
      case 2:
        parse ReplacementCommand(); break;
      case 3:
        sceneReplacementCommand(); break;
    }
    cont = decoder.parseInt(1);
  } while( cont == 1);
}

```

여기서 parseInt(int n) 함수는 BIFS 스트림에서 데이터를 비트단위로 읽는 것으로 n은 읽어야 할 비트수를 나타내고 그 결과값은 정수로 읽어 들인다.

```

void sceneReplacementCommand() {
  int reserved = decoder.parseInt(6);
  useNames = decoder.parseInt(1);
  parseProtoList();
  parseSFNode(NT_TOPNODE);
  int hasRoutes = decoder.parseInt(1);
  if(hasRoutes == 1) {
    parseRoutes();
  }
}

```

위의 루틴은 초기에 BIFS 장면을 구성할 때 쓰이는 루틴이다. useName은 VRML의 USE를 파일에서 사용하고 있

는지를 나타내는 전역변수로 여기서 한번 읽으면 본 루틴이 끝날 때까지 적용이 된다. 그리고 parseProtoList()는 프로토로 선언된 노드들을 파싱하는 루틴이고 parseSFNode()는 SingleField 노드를 파싱하는 것이다. 따라서 NT_TOP-NODE의 의미는 현재 SceneRoot부터 파싱함을 나타낸다. 여기서 VRML이 BIFS와 다른 점이 발견되는데, 바로 TOPNODE의 선언이다. VRML은 텍스트형태로 파일에 들어가기 때문에 그대로 노드 이름이 들어가면 되지만, BIFS의 경우 노드를 그냥 이름으로 넣는 게 아니고 몇 개의 비트로 인코딩해서 넣게 된다. 이를 위하여 BIFS에서는 Node-DataType(NDT) 정의하고 이에 대한 테이블을 사용하고 있는데, 같은 노드라도 NDT가 다르면 다른 비트 수로 코딩이 된다. 현재 BIFS에는 SF2DNode, SF3DNode, SFTopNode 등 31개의 NDT를 정의하고 있는데, 위의 경우에서 NDT가 TopNode 이므로 표준문서에 기술된 테이블을 보면 3비트로 노드를 코딩하고 있음을 알 수 있다.

다음은 각 노드를 파싱하는 루틴으로 장면 루트를 시작으로 각 BIFS 노드와 그 필드를 파싱하게 된다. 여기서 노드는 child로 또 다른 노드를 가질 수 있으므로 순환(recursion)이 발생하게 된다.

```

BaseNode parseSFNode(int parentNDT) {
    BaseNode node = null;
    int isReused = decoder.parseInt(1);
    if (isReused == 1) { // this node refers another node
        int nodeID =
            decoder.parseInt(bifsConfig.nodeIDbits);
    }
    else {
        int nbBits = getNDTnbBits(parentNDT);
        int childNodeType = decoder.parseInt(nbBits);
        int nodeType =
            getNodeType(parentNDT, childNodeType);
        int isUpdateable = decoder.parseInt(1);
        if (isUpdateable == 1) {
            int nodeID =
                decoder.parseInt((bifsConfig.nodeIDbits));
            if (useNames == 1) {
                parseString();
            }
        }
    }
    if (childNodeType == 0) { // extension case
    }
    node = createNode(new Integer(nodeType));
    node.setID(-1);
    bifsScene.nodes.add(node);
    if (root == null) {
        root = node;
        node.bifsScene.setSceneRoot(root);
    }
    int isUpdateable = decoder.parseInt(1);
    if (isUpdateable == 1) { // equivalent to DEF
        node.nodeID =
            decoder.parseInt(bifsConfig.nodeIDbits);
        if (useNames == 1) {
            node.defName = decoder.parseString();
        }
    }
}
    
```

```

int maskDesc = decoder.parseInt(1);
if (maskDesc == 1) { // mask node description
    parseMaskNodeDescription(node);
}
else { // list node description
    parseListNodeDescription(node);
}
}
return node;
}
    
```

위의 코드에서 isReused는 VRML의 USE와 같은 의미로 앞서서 DEF가 선언된 노드를 참조하는 경우에 해당한다. BIFS에서 DEF선언된 노드는 DefName이나 NodeId를 부여해서 사용하고 있는데, 여기서는 USE 문을 만났을 때 NodeId를 가지고 DEF가 선언된 노드를 찾게 된다. 앞서서 기술한 바와 같이 NDT에 따라서 자식 노드를 코딩하는 비트 수가 달라지게 되므로 부모 NDT 정보를 이용해서 파싱해야 할 자식의 코딩된 비트 수를 얻게 된다. 따라서 getNDTnbBits() 함수는 부모 NDT를 주고 자식의 파싱시 읽어야 할 비트 수를 얻어오는 함수이다. bifsConfig는 각 노드, 루트, 프로토에 대해서 코딩 할 때 사용하고 있는 비트 수에 대한 정보를 가지고 있는데, 이는 DMIF의 DecoderConfig-Descriptor 내에 있는 decoderSpecificInfo로부터 그 값을 읽어 들인다. isUpdateable은 VRML에서 DEF 선언문에 대응하는데 선언된 노드에 NodeId를 부여하게 되고, 만일 스트링까지 부여하게 되면 useNames 필드가 세트되어 있고 defName을 스트링형태로 읽으면 된다. 여기까지의 파싱 과정을 VRML 형태로 적어보면 아래와 같다.

```

DEF NodeID defName Group { fields }
USE nodeID defName
    
```

다음은 실제 BIFS 노드를 만드는 곳으로 createNode()에서 하게 된다. 여기서는 해당 nodeType을 주면 미리 해쉬 테이블에 만들어진 각 클래스의 생성자를 불러서 해당 노드 클래스를 만들게 된다.

각 BIFS 노드 클래스에 대한 생성자를 정확하게 만드는 것이 BIFS 파서에 있어서 중요한 일이다. 실제 문서에는 각 노드들에 대해서 필드들만 정해져 있고 이들 필드들이 NDT에 따라서 몇 비트로 코딩되는지 정보만 들어있다. 따라서 이를 참조해서 제대로 된 노드 클래스 생성자를 만드는 것이 중요하다.

다음은 예로써 그룹노드의 클래스 정의와 생성자를 보여 주고 있다.

```

public class BIFSGroup extends BaseNode {
    MFNode addChildren; // eventIn
    MFNode removeChildren; // eventIn
    MFNode children; // exposedField
    int numFields = 3;
    public BIFSGroup(Scene s) {
    
```

```

super(s);
field = new BaseField[numFields];
ndt = NT_3D;
addChildren = new MFNode(NT_3D);
removeChildren = new MFNode(NT_3D);
children = new MFNode(NT_3D);
field[0] = addChildren;
field[1] = removeChildren;
field[2] = children;
// from node coding table
numALLfields = numFields;
numDEFfields = 0;
numDYNfields = 0;
nDEFbits = 0;
nINbits = 2;
nOUTbits = 0;
def2all = new int[numFields];
in2all = new int[numFields];
dyn2all = new int[numFields];
def2all[0] = 2; def2all[1] = 0; def2all[2] = 0;
in2all[0] = 0; in2all[1] = 1; in2all[2] = 2;
dyn2all[0] = 0; dyn2all[1] = 0; dyn2all[2] = 0;
    }
}
    
```

여기서 주의해야 할 것은 MFNode로 정의된 addChildren, emoveChildren, children의 필드를 만드는 것이다. 여기에는 실제 SFNode들이 들어가기 때문에 이들을 파싱 하고자 할 때 몇 비트로 노드들이 코딩되어 있는지를 알아야 하기 때문이다. 이때 표준문서의 노드 코딩 테이블이 사용된다.

BIFS에서 사용하는 fieldID는 5가지 모드를 갖는다. 즉 ALL, DEF, IN, OUT, DYN 등으로 ALL은 보통 fieldID와 매핑 되고 DEF, IN, OUT, DYN은 각각 모드에서 사용하고 있는 필드와 코드를 노드 코딩 테이블에서 기술하고 있다. 위의 노드 생성자에서 def2all, in2all, dyn2all은 각각의 모드에서 해당 필드를 찾고자 할 때 쓰인다. 이에 대한 좀 더 구체적인 설명을 다음에서 한다.

각 노드의 필드들은 벡터나 리스트 형태로 들어갈 수 있다. 여기서는 리스트 형태로 들어간 것을 기준으로 파서의 구현을 설명한다.

```

void parseListNodeDescription (BaseNode node) {
int endFlag = decoder.parseInt(1);
while (endFlag == 0) {
if (node.proto != null) { // proto case
// parse proto
}
else { // non-proto case
int fieldRef;
fieldRef = decoder.parseInt (node.nDEFbits);
int selfField = node.def2all [ fieldRef ];
BaseField f = node.field [ selfField ];
this.parseField (f);
}
endFlag = decoder.parseInt (1);
}
}
    
```

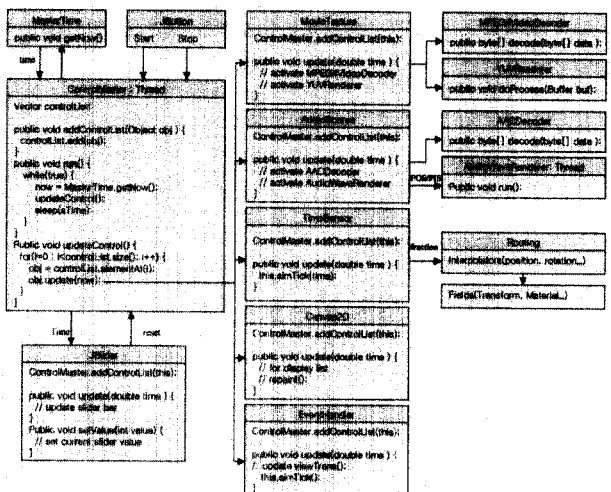
위의 함수를 보면 endFlag가 1이 될 때까지 필드들을 계속 읽게 되는데, endFlag는 VRML의 “)”에 대응된다. 그리고 필드를 읽을 때 node.nDEFbits 만큼 읽어서 각 노드 생성자에서 넣어준 값인 def2all을 통해서 어떤 필드 인지를 찾고 나서 필드 파싱을 시작하게 된다. 이밖에도 각 노드의 필드라든가 루트등의 파싱이 있는데 여기서는 생략한다.

파싱을 하고 나면 Java, Java2D, Java3D API를 사용해서 장면 그래프를 구성하게 된다. VRML이나 BIFS의 장면 그래프는 Java3D에서 제공하는 장면 그래프 구조와 비슷하고 노드들도 일대 일로 매핑 되는 것이 많아 노드들을 Java3D 노드로 구현하는데 용이하다.

(그림 8)은 시간 축에 기반 하여 구현된 MPEG-4시스템의 이벤트 처리의 흐름도를 나타내고 있다. 전체적인 컨트롤을 담당하는 ControlMaster는 쓰레드로 처리되면서 주기적으로 시스템 클럭역할을 하는 MasterTime으로부터 현재 시간을 얻어온다. 그리고 나서 현재 컨트롤 리스트에 들어가는 클래스들에게 이 시간 값을 주면서 update() 하도록 한다. Control List에 등록되는 것은 MovieTexture, AudioSource, TimeSensor, Canvas2D, EventHandler 등이다.

이들은 controlmaster가 건네준 시간 값으로 자신들이 그 시간에 해야 할 일을 결정하고 처리한다. 예를 들면 MovieTexture의 경우 비디오 데이터가 렌더링 되고 있다면 각 비디오 샘플에 붙은 타임 스탬프를 가져와 어떤 비디오 샘플이 그 시간에 렌더링해야 하는지를 결정하여 해당 비디오를 렌더링하게 된다.

(그림 8)에서 Slider는 시간이 지남에 따라 미디어 플레이 진행상황을 나타내는 것으로 Forward, Backward 그리고 random access 등이 가능토록 되어있다.



(그림 8) 이벤트 핸들링 흐름도

mp4 파일의 예제를 파싱한 결과 BIFS 장면 그래프를 보여주고 있는데, 이는 비디오를 사각형에 텍스트로 매핑하고

이를 시간이 지남에 따라 위치 변경과 회전을 행하는 장면이다. 이때 이벤트의 발생은 TimeSensor에서 하고, PositionInterpolator와 ScalarInterpolator가 위치와 회전각도에 대한 값을 만들어내고, Transform2D의 translation과 rotation-Angle값을 바꾸어줌으로써 애니메이션이 이루어진다. 이와 같은 과정은 BFIS 장면 그래프의 맨 끝에 기술된 루트문에 의해서 기술되어 있다. 이와 같은 장면 그래프를 이용하여 합성과 렌더링을 하면 애니메이션이 되는데 (그림 9)는 비디오가 회전하는 장면 중 한 스냅 샷을 보여주고 있다.

(그림 10)은 ImageTexture 노드와 TouchSensor를 사용하여 사용자 상호작용이 가능 하도록 구성된 MPEG-4 컨텐츠로, 마우스를 화면의 가장자리에 있는 꽃 영상들 중 하나로 움직이면 그 영상이 선택되었다는 것을 나타내기 위하여 영상의 크기가 변하고 바로 옆에 있는 영상의 크기도 조금 변한다. 그리고 마우스를 클릭하면 선택된 영상의 확대영상이 화면의 중앙에 디스플레이 되도록 되어 있다. 현재 화면은 아래쪽 가운데 영상을 클릭한 결과로 그 해당영상의 확대된 영상이 중앙에 보이고 있다.

(그림 11)은 2D 그래픽스 노드만 들어있는 장면을 지원한 것으로 PositionInterpolator와 CoordinateInterpolator를 통해서 나온 결과를 점들의 좌표를 바꾸는 방법으로 애니메이션을 하고 있다. 이 예제는 Flash Animation과 같은 것들도 MPEG-4의 BIFS 노드들의 조합을 사용해서 구현할 수 있음을 잘 보여주고 있다.

```

OrderedGroup (
  children [
    DEF 0 Transform2D {
      translation -160.0 120.0
      scale 1.0 1.0
      rotationAngle 0.0
      children [
        Shape {
          appearance Appearance {
            material DEF 1 Material2D {
              emissiveColor 0.0 0.0 0.0
              filled true
              transparency 0.0
            }
            texture MovieTexture {
              url [
                url null OD ID 1
              ]
              speed 1.0
              loop true
              startTime 0.0
              stopTime 6.73
              repeats true
              repeat true
            }
          }
        }
        geometry Rectangle {
          size 320.0 240.0
        }
      ]
    } // end of shape
  ]
)

```

```

DEF 2 PositionInterpolator2D (
  key [ 0.0 1.0 ]
  keyValue [ (-160.0 120.0) (118.0 -77.0) ]
)
DEF 3 ScalarInterpolator (
  key [ 0.0 1.0 ]
  keyValue [ 0.0 6.28 ]
)
] // end of children
) // end of DEF 0 Transform2D
Transform2D (
  translation 180.0 -210.0
  scale 1.0 1.0
  rotationAngle 0.0
  children [
    Shape (
      appearance Appearance {
        material Material2D {
          emissiveColor 0.5 0.5 0.5
          filled true
          transparency 0.0
        }
      }
    )
    geometry DEF 4 Text {
      string [ AuthorI/NetCODEC ]
      fontStyle FontStyle {
        family [
          Tahoma
          SERIF
          SANS
          TYPEWRITER
        ]
        horizontal true
        justify [ BEGIN BEGIN ]
        language = (null)
        leftToRight true
        size 16.0
        spacing 1.0
        style = PLAIN
        topToBottom true
      }
    }
  ]
) // end of Transform2D
Transform2D (
  translation 181.0 -210.0
  scale 1.0 1.0
  rotationAngle 0.0
  children [
    Anchor {
      url [
        url http://www.netcodec.com OD ID 0
      ]
    }
    children [
      Shape {
        appearance Appearance {
          material Material2D {
            emissiveColor 0.75 0.75 0.75
            filled true
            transparency 0.0
          }
        }
      }
      geometry USE 4
    ] // end of Shape
  ]
)

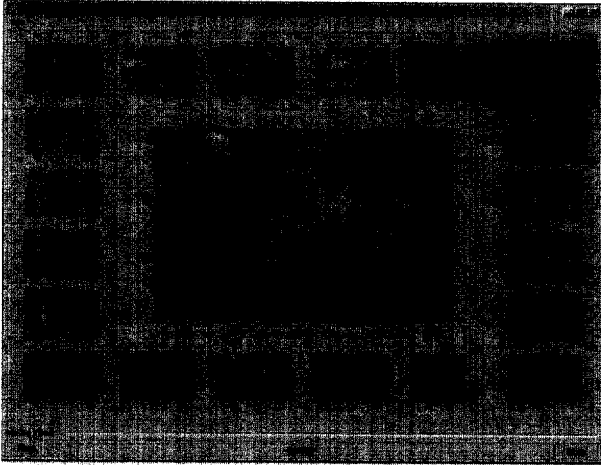
```



```

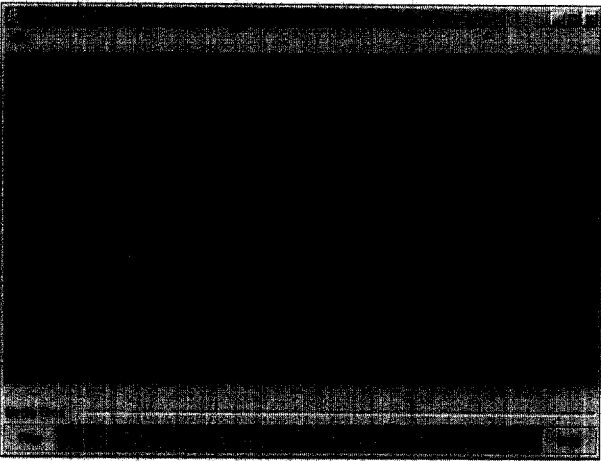
    } // end of Anchor
  ]
} // end of Transform2D
DEF 5 TimeSensor {
  cycleInterval 4.0
  enabled true
  loop true
  startTime 0.0
  stopTime 6.73
}
]
} // end of OrderedGroup
Route 5.fraction_changed to 2.set_fraction
Route 2.value_changed to 0.translation
Route 5.fraction_changed to 3.set_fraction
Route 3.value_changed to 0.rotationAngle

```



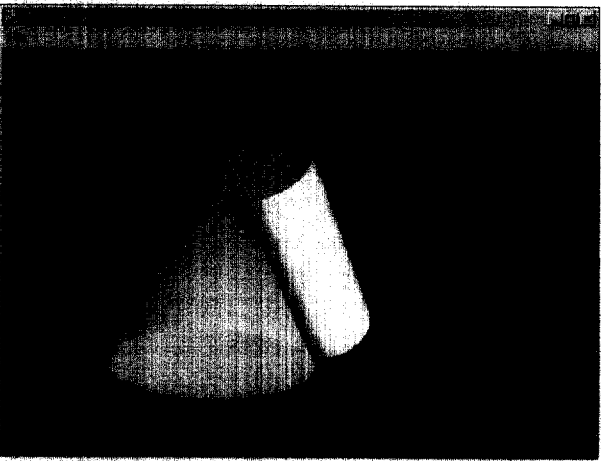
(그림 10) ImageTexture 와 사용자 상호작용의 예

지금까지 MPEG-4 시스템의 구현에 대해서 고찰하였다. 오디오의 경우 AAC Low Complexity Profile을, 비디오의 경우 Simple Profile을 만족시키고 있다. MPEG-4 시스템은 Java, java2D등을 사용하여 구현하였는데, 이는 MPEG-4에서 추구하고 있는 목표와 자바에서 추구하고 있는 목표가 서로 부합되는 점이 많기 때문에 아주 적절한 선택이라 할 수 있다. MPEG-4에서는 이식성을 높이고 지원되는 단말기 성능에 맞는 서비스를 제공하여 아주 광범위한 응용분야에서 쓰일 수 있는 표준을 만드는 것이다. 자바에서도 One source-multiple use등과 같은 철학을 가지고 있고, JMF(Java Media Framework)와 같은 그룹에서는 MPEG-4에서 하고 있는 것과 거의 유사한 목적을 가지고 자바 API를 개발하고 있다.



(그림 11) 2D 그래픽스 애니메이션

3D의 경우 Java3D를 쓰게 되는데, 이는 장면 그래프구조가 VRML구조와 아주 잘 매칭이 되어서 실제 구현과 수행 효율면에서 많은 장점을 가지고 있다. (그림 12)는 Java3D를 이용한 플레이어에서 3차원 MPEG-4 콘텐츠를 플레이한 장면을 보여주고 있다.



(그림 12) 3D 그래픽스 애니메이션



(그림 9) 비디오 회전의 예

5. 결 론

본 논문에서는 MPEG-4 시스템의 구현에 있어서 중요한 컴포넌트 들인 DMIF와 BIFS 파서를 Java와 Java2D, Java

3D기반으로 구현하는 방안에 대하여 기술하였다. 실제 C++과 같은 언어를 써서 DMIF나 BIFS 파서 구현한다 하여도 커다란 차이는 없다. 단지 비디오, 오디오 코덱 등과 같은 성능에 민감한 부분들은 C++가 더 효율적이므로 API형태의 DLL로 만들어 JNI(Java native interface)를 통해서 자바에서 불러 쓰도록 하였다. 현재 MPEG-4 시스템을 구현한 곳은 Envivio와 IBM그리고 Blaxxun등을 들 수 있다. Envivio와 Blaxxun은 C, C++을 써서 DirectX를 사용하고 있으며, 본 시스템과 IBM은 Java기반으로 OpenGL을 사용하고 있다. Envivio는 2D 그래픽스만 구현되어 있는 상태이며, 3D는 지원하지 못하는 반면, 3D 그래픽스 전문 회사이었던 Blaxxun는 3D는 구현이 완료되어 있는 상태이며, 오디오, 비디오 및 2D 그래픽스는 구현이 되어 있지 않다. 이와 같이 MPEG-4시스템에 대한 완성도가 떨어지고 또한 이를 위한 저작도구가 미비한 상태에서 다양한 콘텐츠를 개발하는 것이 어려운 것이 현실이다. 국제 표준 규격을 만족시키며 개발된 본 MPEG-4 시스템은 2D 및 3D가 모두 구현은 물론 MPEG-4시스템간의 상호 호환성이 매우 뛰어난 점이 구현된 여러 시스템과의 상호 연동시험을 통해 알 수 있었다. 현재 고속의 데이터 통신이 가능한 유무선 통합 멀티미디어 서비스 시스템으로 활용하고 있다.

참 고 문 헌

[1] MPEG-4 official site : <http://www.cseit.it/mpeg/>.
 [2] MPEG-4 System site : <http://garuda.imag.fr/MPEG4/>.
 [3] IBM : <http://www.alphaworks.ibm.com/tech/mpeg-4>.
 [4] Envivio : <http://www.envivio.com>.
 [5] Blaxxun song : <http://www.octaga.com/SoNG-Web/>.
 [6] ISO/IEC 14496-1, Coding Of Audio-Visual Objects : Systems, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2501, October, 1998.
 [7] ISO/IEC 14496-2, Coding Of Audio-Visual Objects : Visual, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2502, October, 1998.
 [8] ISO/IEC 14496-3, Coding Of Audio-Visual Objects : Audio, Final Draft International Standard, ISO/IEC JTC1/SC29/WG11 N2503, October, 1998.
 [9] MPEG-4 Part 1 : Systems(ISO 14496-1), doc, N2501, Atlantic City, N. J., USA, Oct., 1998.
 [10] MPEG-4.Part 6 : DMIF(ISO 14496-6), doc, N2506, Atlantic City, N. J., USA, Oct., 1998.
 [11] VRML(ISO 14772-1), "Virtual Reality Modeling Language," April, 1997.

[12] Julien Signes, "Binary Format for Scene(BIFS) : Combining MPEG-4 media to build rich multimedia services," <http://www.cseit.it/mpeg/documents/koener/signes.zip>.
 [13] 3G TS 22.105 V3.9.0, "Service and Service Capabilities," 3GPP, 2000.
 [14] 임영권, "기술적 쟁점, 멀티미디어 스트리밍", 마이크로소프트웨어, pp.212-219, 2001.
 [15] 이재용, "스트리밍에 활력 불어넣는 MPEG-4 의 힘", 마이크로소프트웨어, pp.238-244, 2001.



강 기 정

e-mail : kjkang@kt.co.kr

1990년 대전대학교 전자계산학과(공학사)
 1993년 경희대학교 전자공학과(공학석사)
 1993년~현재 KT서비스개발연구소 선임보
 연구원
 1999년~현재 경희대학교 전자공학과 박사
 과정

관심분야 : MMS, 영상처리, VOD/AOD, 유무선통합 멀티미디어 스트리밍 서비스 등



홍 충 선

e-mail : cshong@khu.ac.kr

1983년 경희대학교 전자공학과졸업(학사)
 1985년 경희대학교 전자공학과(공학석사)
 1988년~1999년 한국통신 통신망연구소
 선임연구원/네트워킹연구실장
 1996년 Keio University, Department of
 Information and Computer
 Science (공학박사)

1999년~현재 경희대학교 전자정보학부 조교수
 관심분야 : 인터넷 서비스 및 망 관리구조, 분산컴포넌트관리,
 IP 멀티캐스트, 멀티미디어스트리밍 등

이 대 영

e-mail : dylee@khu.ac.kr

1984년 서울대 물리학과 졸업(학사)
 1971년 캘리포니아 주립대학원 컴퓨터학과(공학석사)
 1979년 연세대학교 전자공학과(공학박사)
 1990년~1993년 경희대학교 산업정보대학원 대학원장
 1999년~2000년 한국통신학회 회장
 1971년~현재 경희대학교 전자정보학부 교수
 관심분야 : 영상처리, 컴퓨터 네트워크, 컴퓨터 시스템등