석 사 학 위 논 문

리눅스 GStreamer를 기반으로 한 모바일 멀티미디어 프레임워크 설계

Design of a Mobile Multimedia Framework based on Linux GStreamer

신 현 정

한 양 대 학 교 대 학 원

2007년 2월

석 사 학 위 논 문

리눅스 GStreamer를 기반으로 한 모바일 멀티미디어 프레임워크 설계

Design of a Mobile Multimedia Framework based on Linux GStreamer

지도교수 정 제 창

이 논문을 공학석사학위 논문으로 제출합니다.

2007년 2월

한 양 대 학 교 대 학 원

전자통신컴퓨터 공학과

신 현 정

이 논문을 신현정의 석사학위 논문으로 인준함.

2007년 2월

심사위원장 김 동 규 (인)

심사위원 정 제 창 (인)

심사위원 천 방 훈 (인)

한 양 대 학 교 대 학 원

국 문 요 약

최근의 모바일 기기에 대한 요구 사항의 발전 추이로 볼 때 3~4년 이내에 고성능의 멀티미디어 처리, 고속의 끊김없는 높은 대역폭의 네트워크 연결성, 다중 태스크 실행 등과 같이 현재 PC에서만 기대할 수 있는 기능들이 모바일 기기 에서도 실현 가능할 것으로 예측된다. 모바일 플랫폼이란 모바일 기기에 탑재되어 해당 기기에 다양한 어플리케이션을 실행할 수 있도록 하는 하드웨어 및 소프트웨어에 대한 프레임워크를 기술하는 것이다. 하드웨어 측면으로는 멀티미디어 어플리케이션을 구동하기 위한 어플리케이션 프로세서 칩셋과 무선 네트워크 연결을 처리하는 모뎀 칩셋이 주요 기술 요소이며 소프트웨어 측면에서는 커널부터 미들웨어까지 그리고 어플리케이션 프레임워크를 포함하는 소프트웨어 솔루션인 모바일 소프트웨어 플랫폼을 들 수 있다. 모바일 기기의 구조를 개발하기 위해서는 이 세 가지 주요 기술 요소들이 모두 설계, 구현되어 서로 연동되어야 한다.

본 논문에서는 리눅스 기반의 모바일 소프트웨어 플랫폼에서의 GStreamer기반의 멀티미디어 프레임워크를 제안하여 설계하였다. GStreamer는 기본적인 기능을 담당하는 GStreamer core와 멀티미디어 어플리케이션에 필요한 기능들을 플러그인으로 제공함으로써 다양한 모바일 기기에 맞는 구성을 소스 수정없이 쉽게 이룰 수 있도록 해준다. 제안하는 멀티미디어 프레임워크에서는 GStreamer의 구조를 도입하여 어플리케이션이 멀티미디어 각 기능에 따른 파이프라인을 손쉽게 구성할 수 있도록 하였으며 실험를 통해 플러그인을 변경하여 다양한 파이프라인의 구성이 가능함을 보였다. 또한 GStreamer 자체의 최적화를 통해 초기화 시간를 최대 72.45%, 코드 크기를 66.27% 정도 감소시켰다.

제안하는 멀티미디어 프레임워크의 다른 특징 중의 하나는 OpenMAX

API(Application Programing Interface)의 도입이다. 하드웨어 환경에 의존적인 코덱들을 OpenMAX IL(Intergration Layer) 컴포넌트로 구성하여 멀티미디어 프레임워크에서는 OpenMAX IL API를 통해 하드웨어의 변경 및 특성과 상관없이 일관성 있는 코덱 API들을 사용할 수 있다. OpenMAX IL 컴포넌트를 GStreamer 플러그인으로 통합하여 두 구조의 장점을 모두 취하였으며 실험 결과 통합에 따른 소요시간의 증가는 코덱 종류나 데이터 종류에 관계없이 1 sec 정도의 적은 수준으로 일정하게 나타났으며 프레임 길이 증가에 대해서는 0.1~0.2 sec정도의 무시할 만한 소요시간이 측정되었다.

목 차

그 림 목 차

표 목 차

제 1 장 서 론

1.1 연구의 필요성

최근의 모바일 기기에 대한 요구 사항의 발전 추이로 볼 때 3~4년 이내에 고성능의 멀티미디어 처리, 고속의 끊김 없는 높은 대역폭의 네트워크 연결성, 다중 타스크 실행 등과 같이 현재 PC에서만 기대할 수 있는 기능들이 모바일 기기 에서도 실현 가능할 것으로 예측된다. 모바일 플랫폼이란 모바일 기기에 탑재되어 해당 기기에 다양한 어플리케이션을 실행할 수 있도록 하는 하드웨어 및 소프트웨어에 대한 프레임워크를 기술하는 것이다. 하드웨어 측면으로는 멀티미디어 어플리케이션을 구동하기 위한 어플리케이션 프로세서 칩셋과 무선 네트워크 연결을 처리하는 모뎀 칩셋이 주요 기술 요소이며 소프트웨어 측면에서는 커널부터 미들웨어까지 그리고 어플리케이션 프레임워크를 포함하는 소프트웨어 솔루션인 모바일 소프트웨어 플랫폼을 들 수 있다. 모바일 기기의 구조를 개발하기 위해서는 이 세 가지 주요 기술 요소들이 모두 설계, 구현되어 서로 연동되어야 한다.

그 중에서도 모바일 기기에 있어 멀티미디어 기능에 대한 요구 사항은 특히 갈수록 높아지고 있으며 동시에 다양한 멀티미디어에 특화된 모바일 기기들이 생산되고 있다. 이러한 시장 상황에서 대응하기 위해서는 모바일 소프트웨어 플랫폼내의 유연성 있고 안정성 있는 멀티미디어 프레임워크의 설계가 필수적이다.

1.2 연구의 목표 및 내용

본 연구를 위하여 먼저 모바일 기기에서의 리눅스 도입에 대한 현황을 파악하도록 한다. 현재 기존 상용 모바일 플랫폼의 로열티 문제 및 모바일 기기의 빠른 성장에 따른 개발 기간의 단축등의 이유로 모바일 기기에서의

리눅스의 도입은 전폭적으로 늘어나고 있어 2010년에는 그 비중이 다른 기존 상용 모바일 플랫폼과 동등한 위치가 될 것으로 예상된다[1].

리눅스 기반의 GStreamer을 모바일 플랫폼에서의 멀티미디어 프레임워크에 도입하기 위해 GStreamer의 기본적인 개념과 구조등을 살펴 보고 그 적합성을 검증하도록 한다. 그리고 모바일 환경에서의 다양한 멀티미디어 프레임워크를 단일화하려는 목적으로 만들어진 OpenMAX API들을 파악하여 GStreamer로의 적용에 대해 살펴본다.

본 논문에서는 리눅스 GStreamer를 기반으로 멀티미디어 프레임워크를 설계, 구축하고 멀티미디어 프레임워크를 위한 코덱구조에 OpenMAX API를 적용함으로써 컨텐츠 실행 성능과 하드웨어로부터의 독립성을 확보하는 것을 목표로 한다. 제안한 멀티미디어 프레임워크는 각 구조의 성능과 특성을 다양한 시나리오와 동영상을 통해서 검증하도록 한다.

본 논문의 구성은 다음과 같다. 2장에서는 무선 단말기 시장에서의 리눅스의 성장에 대한 그 배경 및 도입 현황에 대한 내용을 기술한다. 3장에서는 본 논문의 주 관심 분야인 GStreamer의 개념과 구조에 대해 구체적으로 기술한다. 4장에서는 OpenMAX API의 구조와 개념에 대해 기술하고 GStreamer로의 적용 방법에 대해 기술한다. 5장에서는 제안하는 모바일 플랫폼에서의 멀티미디어 프레임워크의 구조를 설계하고 그 특징과 장점을 기술한다. 멀티미디어 어플리케이션 중 플레이어와 레코더를 예로 제안한 멀티미디어 프레임워크의 구조를 어떻게 사용하여 동작할 수 있는지를 기술한다. 6장에서는 다양한 시나리오와 동영상을 이용한 실험 결과를 보이고 결과에 대해 고찰한다. 마지막으로 7장에서 결론을 맺는다.

제 2 장 리눅스 모바일

2.1 무선 단말기 시장에서의 리눅스의 부상

Gartner의 보고서에 따르면 2009년이 되면 전세계적으로 26억개의 단말기가 사용될 거라고 한다. 또한 IDC의 보고서에 따르면 스마트폰 시장은 해마다 85%의 증가율로 성장할 것이라고 예측된다. 또한 스마트폰에 자리잡고 있는 리눅스는 해마다 400%의 시장점유율을 보일 것이라고 예측되고 있다. 2004년에서 2005년 사이에 리눅스가 채택된 단말기 모델은 36개를 넘어가고 있으며 2006-2007년 사이에는 더욱 증가될 것으로 전망되고 있다[1].

플랫폼들의 OEM(Original Equipment Manufacturer)은 다양한 OS와 스택, 툴들을 지원하고 있으며 모델별, 지역별, 네트워크종류별로 그 구성을 달리하며 분기되고 이다. 이에 플랫폼에 대한 교육, 지원, 전문적 기술을 가진 인력들을 통합하는 것이 필요하다. 또한 다양한 플랫폼들은 각 회사 이익에 맞게 독자적으로 설계되어 온 것도 사실이다. 단말기의 기능이 다양해지고 복잡해짐에 따라 코드량은 해마다 거의 두배씩 증가하고 있으며 이를 감당할 수 있는 운영체제와 플랫폼의 역량에 대한 기대도 따라서 증가하고 있다.

리눅스는 워낙 광범위하게 사용자에 의해 발전된 형태여서 리눅스 플랫폼과 CPU에 대해 다양한 선택 항이 존재하며 그래픽과 미들웨어에 대해서도 선택할 수 있는 범위가 넓다. 또한 기존 어플리케이션을 자유롭게 사용할 수 있으며 상품화되었거나 혹은 자유롭게 가져다 사용할 수 있는 공개된 모듈이 많은 것도 장점이다.

리눅스는 개발 비용을 획기적으로 줄일 수 있다. 이는 리눅스가 로열티 없이 무료로 가져다 사용할 수 있는 미들웨어와 어플리케이션이 다양하게 존재하기 때문이다. 이는 치열한 경쟁 시장에서 낮은 소비자 가격으로 높은 이윤을 낼 수 있는 구조를 가능하게 한다.

리눅스는 윈도우 모바일을 제작한 마이크로소프트보다 보다 친밀한 브랜드 이미지를 가지고 있으면서도 채택한 단말기에서 OEM을 이용해 자사 브랜드, 스킨, 혹은 자체 플랫폼을 적용할 수 있는 여지를 제공한다. 리눅스 대신 SymbianOS등의 상용 플랫폼을 사용한다면 높은 라이센스 비용 때문에 성능 대비 가격이 높아져 시장 진입이 더욱 어려울 것이다.

2.2 리눅스 모바일 플랫폼

리눅스를 채택한 단말기내 구조는 그림 1과 같다.

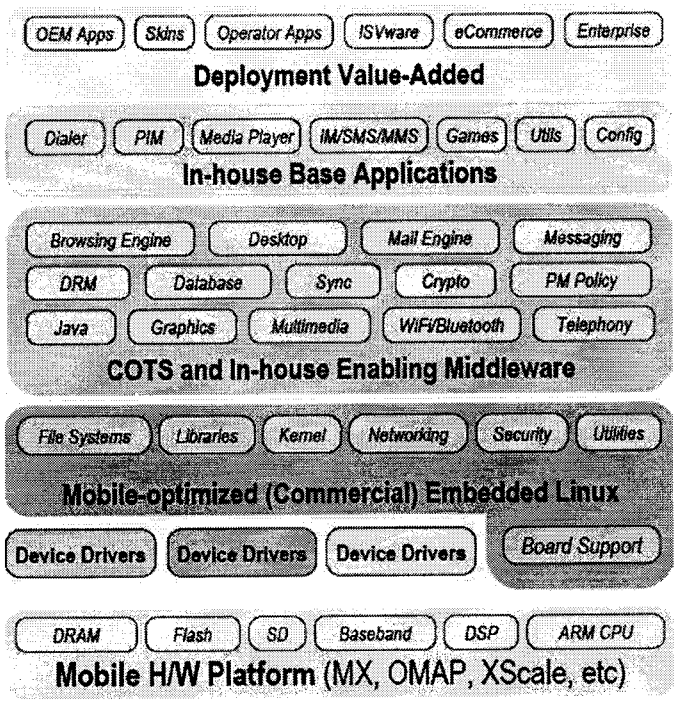

그림 1. 리눅스가 채택된 단말기 구조

■ 모바일 리눅스의 강점

모바일 리눅스의 강점은 플랫폼에 대해서 사업자와 제조사의 직접 수정이 용이하다는 것이다. 즉 다시 말해서 공개 소스들을 통합하여 원하는 사양의 컴

포넌트를 만드는 것이 쉽다는 것이다. 모듈성이 뛰어나 모바일의 메모리 크기와 기능에 맞는 모듈들을 골라 구성할 수 있다. 무엇보다 리눅스는 광범위하고 강력한 커뮤니티가 형성되어 있다. 기존 상용 플랫폼의 경우 다음 버전이 발표될 때까지 기다려야 하고 또한 원하는 기능이 포함시키기도 어려우나 리눅스의 경우 새로운 기능 추가가 빨리 일어나며 제조사가 직접 수정한 부분을 적용시킬 수도 있다[2].

이러한 리눅스의 강점으로 점점 리눅스를 채택하는 단말기가 늘어나고 있으며 그림 2에서 보듯이 향후 2010년에는 스마트폰 시장 점유율에서 SymbianOS를 앞지를 것으로 보인다. 2004년에는 리눅스의 스마트폰 시장점유율은 불과 5%에 불과 했다[3].

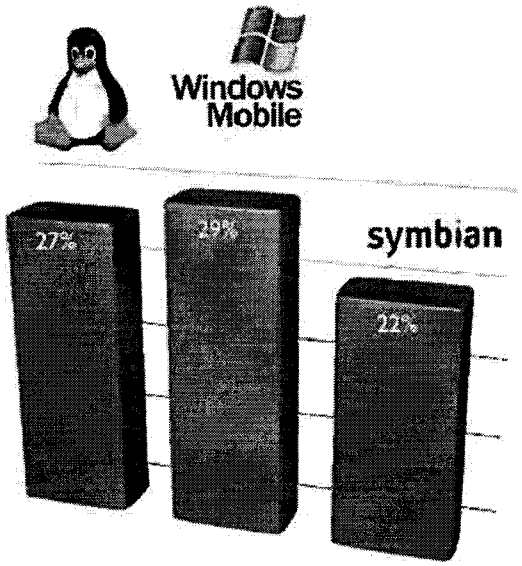

그림 2. 2010년 스마폰 시장의 점유율 예측

제 3 장 GStreamer 멀티미디어 프레임워크

3.1 GStreamer

GStreamer는 멀티미디어 어플리케이션의 작성을 위한 프레임워크이다. 기본적인 디자인은 OGI(Oregon Graduate Institute) 의 비디오 파이프라인에서 가져왔으며 일부는 DirectShow에 기반을 두기도 했다[4].

GStreamer 프레임워크에서는 모든 종류의 스트리밍 멀티미디어 어플리케이션의 작성이 가능하다. GStreamer 프레임워크는 오디오와 비디오를 각각 또는 동시에 다루는 어플리케이션을 쉽게 만들 수 있도록 설계되었다.

GStreamer의 가장 큰 어플리케이션은 미디어 플레이어를 만드는 것이다. GStreamer는 이미 MP3, Ogg/Vorbis, MPEG-1/2, AVI, Quicktime, mod등의 매우 다양한 포맷을 지원하는 어플리케이션을 만들 수 있는 컴포넌트를 가지고 있다. 그러나 GStreamer는 또 하나의 미디어 플레이어 이상이다. GStreamer의 가장 큰 장점은 pluggable 컴포넌트들이 임의의 파이프라인 안으로 섞여 들어갈 수 있어서 독립적인 비디오/오디어 편집 어플리케이션도 만드는 것이 가능하다.

프레임워크는 다양한 코덱과 기능을 제공하는 플러그인에 기반을 두고 있다. 플러그인들은 하나의 파이프라인 안에서 서로 연결되어 정렬된다. 파이프라인은 데이터의 흐름을 말하며 GUI 편집기로 구성하거나 XML (eXtensible Markup Language)로 저장될 수 있어 파이프라인 라이브러리들은 최소한의 노력으로 만들어 질 수 있다.

GStreamer의 핵심적인 기능은 플러그인, 데이터 흐름, 미디어 타입 처리를 위한 프레임워크를 제공하는 것이다. 또한 다양한 플러그인들을 이용한 어플리케이션을 작성하기 위한 API도 제공한다. GStreamer의 가장 기본적인 개념은 다음에 설명할 엘리먼트, 빈, 파이프라인, 패드로 대변될 수 있다[5].

3.2 엘리먼트 (Element)

엘리먼트는 GStreamer에서 가장 중요한 개념이다. 사용자는 기본적으로 엘리먼트들을 생성하여 연결 고리를 만들고 데이터가 이 연결 고리들을 통과하도록 설계한다. 각각의 엘리먼트는 파일에서 데이터를 읽어오거나 읽어온 데이터를 디코딩하거나 해당 데이터를 사운드 카드로 내보내는 등의 어느 특정한 기능을 구현한 내용을 담고 있다. 사용자는 이러한 엘리먼트들을 여러 개 연결함으로써 파이프라인을 만들고 해당 파이프라인을 통해 미디어 파일을 실행하거나 녹음하는 등의 특정한 기능을 수행할 수 있다. GStreamer는 다양한 멀티미디어 어플리케이션을 개발할 수 있도록 광범위한 양과 종류의 엘리먼트들을 포함하고 있으며 필요하다면 새로운 엘리먼트들을 직접 개발할 수도 있다.

모든 다양한 고수준의 컴포넌트들은 GstElement로부터 상속받아 사용한다. 모든 디코더, 인코더, 디멀티플렉서, 비디오 또는 오디오 출력도 모두 GstElement이다. 어플리케이션 작성자에게 엘리먼트는 블랙박스로 보인다. 사용자가 한쪽으로 입력을 넣으면 다른 한쪽으로 출력이 나올 뿐인 것이다. 디코더 엘리먼트를 예를 들면 사용자가 인코딩된 데이터를 넣으면 그 엘리먼트는 결과물로 디코딩된 데이터를 내보낸다.

■ **소스 엘리먼트 (Source Element)**

소스 엘리먼트는 디스크로부터 읽거나 혹은 사운드 카드로부터 읽거나해서 파이프라인에 의해 사용될 데이터를 생성해낸다. 그림 3은 소스 엘리먼트를 시각화한 그림이다. 항상 엘리먼트의 오른쪽에 소스 패드를 그린다.

소스 엘리먼트는 데이터를 받아들이지 않고 단지 만들기만 할 뿐이다. 그림 3에서 해당 엘리먼트가 소스 패드만 가지고 있는 것을 볼 수 있는데 소스 패드는 데이터를 생성해내기만 한다.

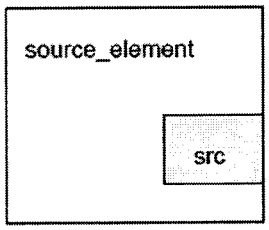

그림 3. 소스 엘리먼트

■ **필터 (Filter), 컨버터 (Converter), 디멀티플렉서 (Demuxer), 멀티플렉서 (Muxer), 코덱 (Codec)**

필터와 필터 유사 엘리먼트들은 입력과 출력 패드를 모두 가지고 있다. 해당 엘리먼트들은 입력(싱크) 패드로부터 데이터를 받고 출력(소스) 패드를 통해 데이터를 제공한다. 이러한 엘리먼트들의 예로는 볼륨 엘리먼트(필터), 비디오 스케일러(컨버터), Ogg 디멀티플렉서, Vorbis 디코더등이 있다.

필터 유사 엘리먼트들은 소스나 싱크 패드의 개수를 임의로 가질 수 있다. 예를 들어 비디오 디멀티플렉서의 경우에는 하나의 데이터 스트림에 대해서 싱크 패드와 여러 개의 (1-N) 소스 패드를 가진다. 반면 디코더는 소스와 싱크 패드를 하나만 갖는다.

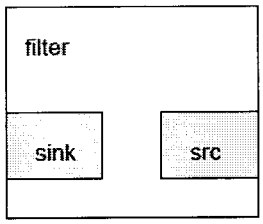

그림 4. 필터 엘리먼트

그림 4는 필터 유사 엘리먼트를 어떻게 시각화할 수 있는지 보여준다. 이러한 엘리먼트는 하나의 소스와 하나의 싱크 패드만 가지고 있다. 입력 데이터를

받는 싱크 패드는 일반적으로 엘리먼트의 왼쪽에 그린다.

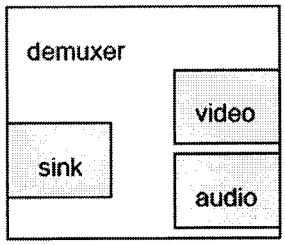

그림 5. 하나 이상의 출력 패드를 갖는 필터 엘리먼트

그림 5는 또 다른 필터 유사 엘리먼트를 보여준다. 그림에서 디멀티플렉서 엘리먼트는 하나 이상의 출력(소스) 패드를 가지고 있음을 보여주고 잇는데 이러한 예로는 오디오와 비디오를 모두 포함하고 있는 Ogg 스트림에 대한 Ogg 디멀티플렉서를 들 수 있다. 한 소스 패드는 기본적인 비디오 스트림을 다른 하나는 기본적인 오디오 스트림을 담게 된다.

■ **싱크 엘리먼트 (Sink Element)**

싱크 엘리먼트는 미디어 파이프라인에서 마지막에 위치한다. 해당 엘리먼트는 데이터를 받아들이기만 할 뿐 생성하지 않는다. 디스크 쓰기, 사운드 카드 실행, 비디오 출력들은 모두 싱크 엘리먼트로 구현될 수 있다.

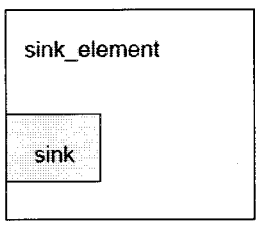

그림 6. 싱크 엘리먼트

■ **엘리먼트 연결하기**

소스 엘리먼트와 필터 유사 엘리먼트, 그리고 싱크 엘리먼트들을 연결함으로써 미디어 파이프라인을 설정할 수 있다. 이것은 GStreamer에서의 미디어 핸들링을 다루는 기본 개념이다.

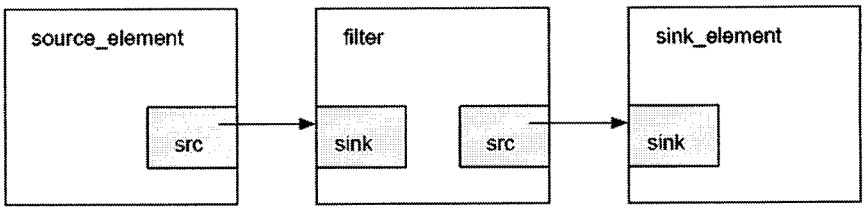

그림 7. 연결된 엘리먼트들

이 세 개의 엘리먼트들을 연결함으로써 가장 간단한 엘리먼트들의 연결고리를 만들게 된 것이다. 소스 엘리먼트는 필터 유사 엘리먼트의 입력으로 동작하고 필터 유사 엘리먼트는 데이터를 특정 동작을 수행한 뒤 그 결과를 마지막 싱크 엘리먼트에게 전송한다. 간단한 Ogg/Vorbis 디코더에 이 그래프를 적용해 보자. 소스는 디스크로부터 파일을 읽고 가져온 디스크 데이터 소스이고 두번째 엘리먼트는 Ogg/Vorbis 오디오 디코더가 될 것이다. 싱크 엘리먼트는 디코딩된 오디오 데이터를 출력하는 사운드 카드가 될 것이다.

■ **엘리먼트 상태 변경**

엘리먼트를 만든다고 해서 바로 어떤 동작을 수행하지는 않는다. 엘리먼트를 수행시키기 위해서는 해당 엘리먼트의 상태를 변경해주어야 한다. GStreamer는 4개의 엘리먼트 상태를 가지고 있으며 각각 고유한 의미를 가진다.

- GST_STATE_NULL: 초기 상태. 엘리먼트에 의한 어떠한 자원의 할당도 일어나지 않은 상태이다.

- GST_STATE_READY: 이 상태에서 엘리먼트는 자신에게 필요한 자원를 할당한다. 장치를 열고 필요한 버퍼를 할당하는 단계라고 보면 된다. 그러나 이 상

태에서 스트림은 아직 열리지 않은 상태로 스트림의 위치는 자동적으로 0이 된다. 이전에 스트림이 열려 있었다면 이 상태에서는 자동적으로 닫히고 위치나 특성등이 다시 초기화된다.

- GST_STATE_PAUSED: 이 상태에서 엘리먼트는 스트림을 열지만 처리을 시작하지는 않은 상태이다. 엘리먼트는 스트림의 위치를 수정하고 읽고 조작할 수 있는 권한을 얻은 상태이고 스트림이 실행될 준비를 마친 상태이다. 클럭이 아직 수행되기 전이라는 것이 PLAYING 상태와 다른 유일한 점이다. PAUSED 상태로 간 엘리먼트는 최대한 빨리 PLAYING으로 옮겨갈 준비를 마쳐야 한다. 예를 들어 비디오나 오디오 출력은 상태 변경을 하자마자 수행될 수 있도록 이미 도착해서 큐에 들어가 있어야 한다. 또한 비디오 싱크는 첫번째 프레임을 그릴 수 있어야 한다. Autoplugger들은 파이프라인에서의 엘리먼트들이 동시에 수행될 수 있도록 같은 상태 변경이 이루어지도록 할 수 있다. 코덱이나 필터 같은 대부분의 다른 엘리먼트에서도 이 상태에서는 특별히 할 일은 없다.

- GST_STATE_PLAYING: 이 상태는 엘리먼트는 클럭이 수행되기 시작했다는 것외에는 PAUSE와 완전히 같다.

사용자는 gst_element_set ()를 통해 상태변경을 할 수 있다. 한 엘리먼트의 상태를 변경하면 GStreamer는 내부적으로는 중간에 필요한 모든 상태 변경을 거치도록 한다. 사용자가 NULL에서 PLAYING으로 상태를 변경하면 내부적으로 READY와 PAUSED가 그 사이에 설정되게 된다.

GST_STATE_PLAYING으로 상태가 변경되면 파이프라인은 자동적으로 데이터를 처리하기 시작한다. 내부적으로 GStreamer는 이를 수행할 쓰레드를 시작하고 파이프라인의 쓰레드와 어플리케이션 자체 쓰레드 사이의 메시지를 전달하게 된다. 이 때 메시지는 GstBus을 사용한다.

3.1 빈(Bin)과 파이프라인 (Pipeline)

빈은 엘리먼트들의 집합을 담고 있는 객체를 말한다. 위에서 언급한 엘리먼트들로 연결된 파이프라인 역시 빈의 특정한 형태로 자신이 포함하고 있는 엘리먼트들의 수행하도록 하는 기능을 가지고 있는 것이다. 빈 자체는 엘리먼트의 서브 클래스이므로 사용자는 엘리먼트들을 다루듯이 빈을 동작시킬 수 있을 뿐만 아니라 많은 엘리먼트들을 구성해야 함으로써 오는 복잡성을 어플리케이션에게 노출하지 않을 수 있다. 예를 들어 빈에 포함된 모든 엘리먼트들의 상태를 변경할 때 빈 자체의 상태만 변경해주면 그 안의 모든 엘리먼트들의 상태도 따라서 자동적으로 변경된다. 또한, 빈은 자신이 포함하고 있는 엘리먼트들로부터 발생하는 에러 메시지나 태그 메세지 같은 버스 메시지들을 통합적으로 관리하여 전달할 수 있다.

파이프라인은 최상위 레벨의 빈을 말한다. 사용자가 파이프라인의 상태를 PAUSED나 PLAYING등으로 변경할 때 데이터의 흐름도 그에 따라 시작되고 미디어 프로세싱도 발생하게 된다. 한번 시작되면 파이프라인은 사용자가 정지시키거나 더 이상 다룰 데이터가 없을 때 까지 독립된 쓰레드안에서 수행된다.

■ **빈**

빈은 엘리먼트 컨테이너이다. 사용자는 빈에 엘리먼트를 추가할 수 있다. 빈 자체도 엘리먼트이므로 빈은 엘리먼트와 같은 방식으로 처리될 수 있다. 빈은 사용자가 서로 연결된 엘리먼트들의 그룹을 하나의 논리적인 엘리먼트에 결합시킨다. 빈은 파이프라인을 더 작은 규모로 나눌 수 있게 하기 때문에 복잡한 파이프라인을 구성하는 데 강력한 역할을 한다. 빈은 또한 그 안에 포함된 엘리먼트들을 관리하는 데 효과적이다. 빈 안에서 데이터를 어떻게 전달할 것인지 데이터 흐름에 가장 최적화된 plan을 생성할 수 있게 한다. Plan 생성은 GStreamer에서 가장 복잡한 과정중의 하나로 스케쥴링이라고 불린다.

■ **파이프라인**

파이프라인은 가장 특화된 빈 타입 중 하나이다. 파이프라인은 포함된 엘리먼트들의 스케쥴링할 수 있는 일반적인 컨테이너이다. 가장 상위 레벨의 빈은 하나의 파이프라인이 되어야 하므로 모든 어플리케이션은 최소한 하나이상의 파이프라인이 필요하다. 파이프라인은 시작될때 백그라운드 쓰레드에서 자체적으로 자동적으로 수행된다.

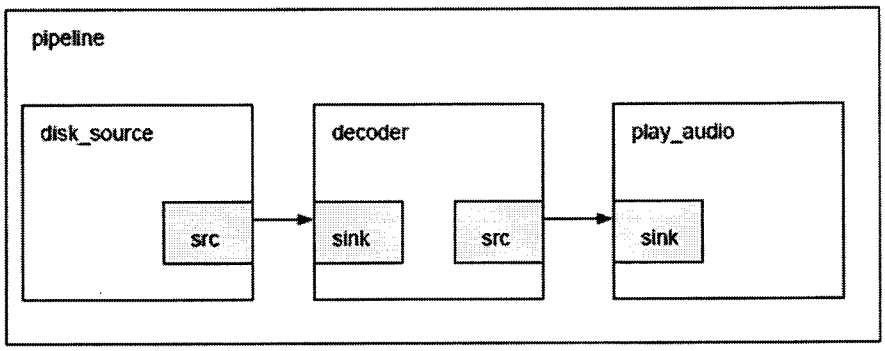

그림 8. 파이프라인/빈 구조

3.2 버스 (Bus)

버스는 자체 쓰레드 컨텍스트내에서 파이프라인 쓰레드에서 어플리케이션로 메시지를 전달하는 일을 관장하는 시스템을 말한다. 버스의 장점은 GStreamer자체가 복잡한 쓰레드를 갖고 있더라도 어플리케이션이 버스를 사용하면 GStreamer내의 쓰레드에 대해 신경쓰지 않아도 된다는 점이다.

모든 파이프라인은 버스를 가지고 있으므로 어플리케이션을 따로 버스를 생성하지 않아도 된다. 어플리케이션이 해야할 유일한 일은 버스에 메시지 핸들러를 설정하는 것이다. 메인 루프가 수행되면 버스는 주기적으로 새로운 메시지가 있는지 확인하고 메시지가 있으면 callback 함수를 호출한다.

버스를 사용하는 데는 크게 두 가지 방법이 있다. 하나는 GLib/Gtk+ 메인

루프를 수행시키고 버스에 watch를 추가시킨다. 이 방법은 GLib 메인 루프가 새로운 메시지가 있는지 확인하고 있다면 사용자에게 알려주는 방식이다. 이 경우에는 gst_bus_add_watch () 나 gst_bus_add_signal_watch ()을 사용한다. 버스를 사용하기 위해서 gst_bus_add_watch ()을 사용해 파이프라인의 버스에 메시지 핸들러를 설정한다. 이 핸들러는 파이프라인이 버스에서 메시지를 제거할 때마다 불린다. 이 핸들러에서는 신호 타입을 체크해서 타입에 따라 처리한다. 버스에서 메시지를 제거해야 하므로 핸들러의 리턴 타입은 늘 TRUE여야 한다. 다른 방법은 버스 자체의 메시지를 체크하는 것이다. 이것은 gst_bus_peek () 이나 gst_bus_poll ()을 사용해서 가능하다.

메인루프의 쓰레드 컨텍스트 내에서 핸들러가 불리워진다는 것은 중요한 정보이다. 이것은 버스상의 파이프라인과 어플리케이션 사이의 통신이 비동기적이라는 것을 의미하므로 오디오 트랙 사이에서 일어나는 cross-fading이나 비디오 효과같은 실시간 어플리케이션에는 적합하지 않을 수 있다. 모든 일들이 파이프라인 컨텍스트내에서 이루어지게 하기 위한 가장 쉬운 방법은 GStreamer 플러그인을 작성하는 것이다. 이 방법은 메시지를 파이프라인에서 어플리케이션으로 전달한다는 기본 목적에는 가장 유용하다. 이러한 접근방법의 장점은 GStreamer의 모든 쓰레드들이 어플리케이션에게는 숨겨지므로 어플리케이션 개발자들은 쓰레드 문제에 대해서 전혀 신경쓰지 않아도 된다는 것이다.

사용자가 GLib 메인 루프를 사용한다면 watch를 붙이는 대신에 버스상에 “message” signal을 연결시킬 수 있다. 이 방법은 모든 메시지에 대해서 switch()를 하지 않아도 된다. 관심있는 signal에 대해서만 “message::<type>” 형식으로 연결하기만 하면 된다.

3.3 패드 (Pads)

엘리먼트에서 기술했듯이 패드는 엘리먼트와 외부를 연결해주는 인터페이스

이다. 데이터는 엘리먼트의 소스 패드로부터 나와서 다른 엘리먼트의 싱크 패드로 전달된다. 해당 엘리먼트가 처리할 수 있는 미디어의 특정 타입은 패드의 capabilities를 보면 알 수 있다.

한 패드는 direction과 availability 두 속성으로 정의될 수 있다. GStreamer는 소스 패드와 싱크 패드 두 가지의 direction으로 정의된다. 이 용어는 엘리먼트 내부의 시각에서 정의된 것이다. 엘리먼트는 싱크 패드에서 데이터를 받고 소스 패드에서 데이터를 생성하여 내보낸다. 도식적으로 싱크 패드는 엘리먼트의 왼쪽에 그리고 소스 패드는 엘리먼트의 오른쪽에 그린다. 그러한 그래프에서 데이터는 왼쪽에서 오른쪽으로 전달된다.

패드의 direction은 availability와 비교하면 매우 간단한 것이다. 한 패드는 always, sometimes, on-request의 세 종류의 availability중 하나를 가질 수 있다. 그 이름처럼 의미하는 바도 명확하다. Always 패드는 언제나 존재한다는 것을 의미하고 sometimes 패드는 특정한 경우에만 패드가 존재한다는 것을 의미한다. 마지막으로 on-request 패드는 어플리케이션으로부터 요구가 있을 때만 나타나는 것을 의미한다.

■ **Capabilities**

패드는 엘리먼트가 외부에 어떤 모습으로 보일 것인가에 중요한 역할을 하기 때문에 메카니즘은 데이터가 capability를 통해서 전달될 수 있는지 아니면 현재 전달되고 있는지를 기술하도록 구현된다.

Capabilites는 패드 템플릿과 패드에 추가된다. 패드 템플릿에서는 해당 템플릿에서 생성된 패드를 통해 전달될 미디어 타입을 기술하고 패드에서는 패드가 아직 협상되지 않은 경우에는 가능한 caps의 리스트를 제공하고 이미 협상이 끝난 패드의 경우에는 현재 해당 패드에 전달되고 있는 미디어 타입에 대해 기술한다.

패드의 capability는 GstCaps 객체로 기술된다. 내부적으로 GstCaps는 미디어 타입을 기술하는 GstStructure를 하나 이상 포함하고 있다. 협상이 완료된

패드는 단지 하나의 GstStructure만을 가지고 있고 고정된 값을 가지고 있게 되지만 이러한 제한은 협상이 안된 패드나 패드 템플릿에는 적용되지 않는다.

예를 들어 다음은 gst-inspect vorbisdec을 수행하여 얻은 결과로 "vorbisdec" 의 capability를 보여주고 있다. 여기에 소스와 싱크 패드가 각각 하나씩 있고 이 두 패드 모두 항상 가능하며 각각에 capability를 가지고 있다. 이 중 싱크 패드는 vorbis-encoded 오디오 데이터를 받아들일 수 있으며 "audio/x-vorbis" 의 MIME 타입을 가지고 있고 소스 패드는 디코딩된 오디오 데이터를 보낼 수 있으며 "audio/x-raw-int" 의 MIME 타입을 가지고 잇다. 이 소스 패드는 또한 오디오 샘플 레이트와 채널 수에 대한 속성도 가지고 있다.

```
Pad Templates:
        SRC template: ' src'
                Availability: Always
                Capabilities:
                        audio/x-raw-float
                                rate: [ 8000, 50000 ]
                                channels: [ 1, 2 ]
                                endianness: 1234
                                width: 32
                                buffer-frames: 0
        SINK template: ' sink'
                Availability: Always
                Capabilities:
                        audio/x-vorbis
```

■ **Capability의 사용**

Capability(caps)는 두 패드 사이에 전달되는 데이터의 타입에 대해 나타내

거나 특정 패드에서 지원하는 데이터의 타입에 대해 나타낸다. 이는 다음과 같은 다양한 목적에 사용된다.

- Autoplugging: capability에 근거하여 패드에 연결될 엘리먼트를 자동적으로 찾을 수 있다. 모든 autoplugger들은 이 방법을 사용한다.

- Compatibility detection: 두 패드가 연결되었을 때 GStreamer는 같은 미디어 타입에 대해서 두 패드가 모두 지원하는지 검증할 수 있다. 두 패드가 서로 호환 가능한지 검증하고 서로 연결하는 과정을 "caps negotiation" 이라고 한다.

- Metadata: 어플리케이션은 패드로부터 capability를 읽어내어 패드를 통해 전달될 미디어의 타입에 대한 정보를 제공할 수 있다.

- Filtering: 어플리케이션은 두 패드 사이에 전달되는 미디어 타입에 대해 제한을 두기 위해 capability를 이용한다. 예를 들어 어플리케이션은 두 패드 사이에 전달될 비디오 크기를 정하기 위해서 "filtered caps" 를 사용한다.

3.4 버퍼(Buffer)와 이벤트(Event)

파이프라인을 통한 데이터 흐름은 버퍼들과 이벤트들의 조합으로 이루어진다. 버퍼들은 실제적인 파이프라인 데이터들을 가지고 있다. 이벤트들은 파일내의 데이터 위치 정보와 EOS(end-of-stream) 알람같은 컨트롤 정보를 가지고 있다. 이 모든 것들은 실행이 시작되면 파이프라인을 통해서 자동적으로 이루어진다.

■ 버퍼

버퍼는 사용자가 생성한 파이프라인 내에 전달되는 데이터를 담고 있다. 일반적으로 소스 엘리먼트는 새 버퍼를 생성하고 연결 고리에 있는 다음 엘리먼트에 있는 패드를 통해 전달한다. 미디어 파이프라인을 만들기 위해서 GStreamer 구조를 사용한다면 해당 엘리먼트가 버퍼를 알아서 제어할 것이므로 사용자는

신경쓸 필요가 없다. 버퍼는 다음과 같은 구성 요소들로 이루어져 있다.

- 메모리에 대한 포인터
- 메모리 크기
- 버퍼에 대한 타임스탬프
- 얼마나 많은 엘리먼트들이 이 버퍼를 사용하고 있는지에 대한 refcount.

이 refcount는 어떤 엘리먼트도 해당 버퍼를 참조하고 있지 않게 되면 버퍼를 삭제하는데 사용한다.

가장 간단한 경우는 버퍼가 생성되고 메모리가 할당되면 데이터가 버퍼로 복사되고 다음 엘리먼트로 전달되는 것이다. 해당 엘리먼트는 버퍼로부터 데이터를 읽은 후 버퍼의 레퍼런스를 해제한다. 이것은 데이터에 대한 메모리를 해제하고 버퍼를 삭제할 수 있도록 한다. 일반적인 비디오나 오디오 디코더들이 이와 같은 방식으로 동작된다.

조금 더 복잡한 시나리오는 엘리먼트가 새로운 버퍼를 할당하지 않고 버퍼를 수정하는 것이다. 뿐만 아니라 엘리먼트는 비디오 캡쳐 소스나 XShm을 사용한 X-server로부터의 메모리 할당처럼 하드웨어 메모리에 쓸 수 있으며 또한 read-only로만 동작하게 할 수도 있다.

■ **이벤트**

이벤트는 데이터가 파이프라인 안에서 버퍼에 의해 움직일때 발생한다. Downstream 이벤트는 스트림의 상태에 대해 다음 엘리먼트에게 알려준다. Upstream 이벤트는 어플리케이션과 엘리먼트 사이의 인터랙션 뿐 아니라 이벤트-이벤트 인터랙션 사이에도 사용된다. 어플리케이션에게는 upstream 이벤트만이 중요하며 downstream 이벤트는 단지 데이터에 대한 개념에 대한 완전한 그림을 얻기 위해 설명된다.

3.5 GStreamer의 구성

GStreamer는 공개 소스 프로젝트이므로 웹 사이트[4]에 소스와 문서를 공개하고 있으며 누구나 참여할 수 있도록 하고 있다. 다음에는 제공되고 있는 소스의 각 모듈의 특성에 대해 기술하였다.

■ **gstreamer**

GStreamer내의 다른 모듈이 반드시 포함되어야 하는 core 라이브러리이다. 기본 기능과 라이브러리, 여러 필수 엘리먼트들과 문서와 테스트 코드를 포함하고 있다.

■ **gst-plugins-base**

검증된 GStreamer의 플러그인과 엘리먼트들을 포함하고 있다. 고정적으로 사용되는 플러그인들의 집합을 가지고 있으며 광범위한 타입의 엘리먼트들을 포함하고 있다. Core 라이브러리와 같이 지속적으로 업데이트되고 있다.

- 다른 사용자들은 이들 플러그인들을 안심하고 사용할 수 있다.
- 엘리먼트들을 만드는 사용자들은 이들 엘리먼트들을 기반으로한 코드를 프로그래밍해야 한다
- 이들 엘리먼트들은 예제, 문서, 테스트 코드와 같이 제공된다.

■ **gst-plugins-good**

바로 상품화해서 사용할 수 있는 정도의 품질을 유지하고 있는 플러그인들을 포함하고 있다. 코드도 잘 정비되어 있으며 라이센스 문제도 모두 해결된 코드들로 문서화와 테스트가 잘되어 있다. 자체 플러그인을 만들어 사용할 때 참조 코드로 사용될 수 있다.

- 사용자들은 안심하고 이들 플러그인들을 사용할 수 있다.
- 엘리먼트를 만드는 사용자들은 이들 엘리먼트를 기반으로 한 코드를 프로그래밍해야 한다.

■ **gst-plugins-ugly**

높은 품질과 정확한 기능을 제공하지만 이 소스를 배포하는 데는 문제가 있

을 수 있다. 플러그인들이나 라이브러리들에 대한 라이센스가 명확하지 않으므로 이후에 특허권 문제를 일으킬 수 있다.

- 사용자는 이들 플러그인들을 사용하려면 그전에 미리 라이센스등을 각각 체크한 후에 사용하여야 한다.

- 엘리먼트를 만드는 사용자는 이들 엘리먼트들을 기반으로 코드를 프로그래밍해야 한다.

■ **gst-plugins-bad**

다른 모듈과 비교하여 완성도가 떨어지는 플러그인들의 집합으로 이루어져 있다. 코드 자체나 문서화, 테스트, 또는 업데이트등에 문제가 있을 수 있다. 이 모듈에 포함된 엘리먼트들은 업그레이드되면 gst-plugins-good이나 gst-plugins-ugly에 포함될 수 있다.

- 플러그인이 문제를 일으켜도 사용자는 불평할 수 없다. 대신 문제점을 고쳐서 코드가 업그레이드 되도록 할 수 있다.

- 새로 추가되는 코드들은 이 모듈에 포함되는 것으로 이 공개 소스 프로젝트에 참여할 수 있다.

3.6 GStreamer 어플리케이션

현재 GStreamer 웹사이트에는 GStreamer 기반으로 제작된 많은 어플리케이션들이 공개되어 있다. 그림 9은 공개된 어플리케이션들을 실행시킨 화면들이다. 대부분의 어플리케이션들은 미디어 플레이어를 구현하고 있다.

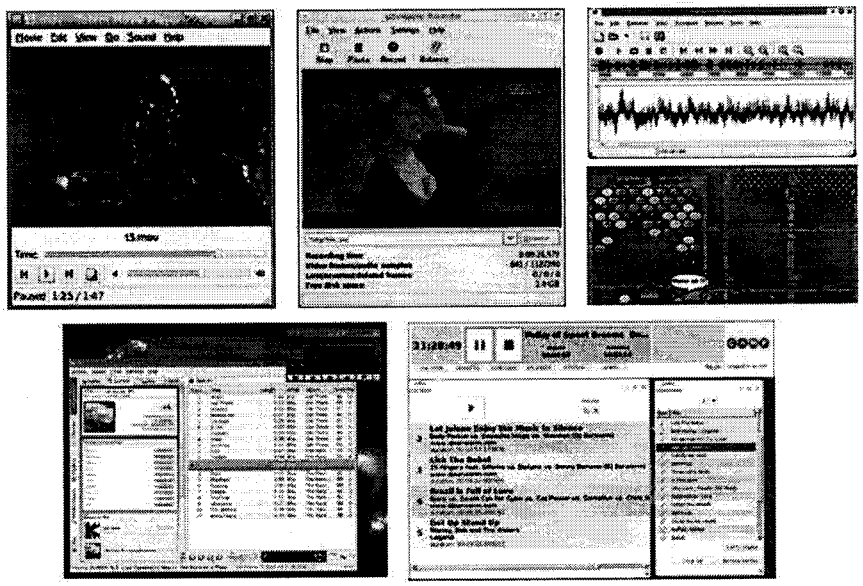

그림 9. GStreamer 기반의 미디어 플레이어 어플리케이션들

제 4 장 OpenMAX

4.1 OpenMAX

OpenMAX는 크로노스 그룹에서 만들어진 로열티가 없고 다양한 플랫폼에 호환성을 가지는 API이다[6]. 최적화된 멀티미디어 컴포넌트들을 다양한 운영 체제과 실리콘 플랫폼 상에 개발되고 통합될 수 있도록 광범위한 미디어 코덱과 어플리케이션에게 이식성(portability)을 제공한다. OpenMAX API는 라이브러리와 코덱 개발자들이 하드웨어 구조와 상관없이 새로운 프로세서의 가속화기능을 가장 효율적으로 사용할 수 있도록 해준다.

사용자가 스마트폰, 오디오/비디오 미디어 플레이어, 게임 콘솔 같은 다양한 플랫폼상에서 좀 더 나은 멀티미디어 어플리케이션 기능을 요구함으로써 멀티미디어 하드웨어 플랫폼의 개발은 점점 가속화되어 가고 있다. 일반적으로 이런 종류의 제품은 고성능와 빠른 데이터 처리량을 요구한다. 결과적으로 다양한 솔루션이 개발될 수록 그에 맞은 멀티미디어 어플리케이션의 가속화를 위한 구조의 설계가 요구되고 있다.

이러한 구조적 다양화의 가장 큰 이슈는 구조에 효율적인 코드를 개발하는 것이다. 하이레벨 언어로 이러한 구조의 잠재력을 모두 구현하는 것은 불가능하기 때문에 이러한 어플리케이션의 대부분은 하드웨어 플랫폼에 최적화된 어셈블리 코드로 구현된다. 결국 다양한 멀티미디어 하드웨어 솔루션을 도입한다는 것은 소프트웨어가 전부 새로 작성되어야 하며 새로운 플랫폼에 대해 다시 최적화해야 한다는 것이다. 구현에 있어서의 이러한 비효율성은 새로운 제품의 출하를 지연시키며 개발 비용을 증가시키고 제품의 품질을 저하시킨다.

■ 새로운 공개 표준 수립

위에서 언급한 문제점들을 공유하고 해결하기 위해서 크로노스 그룹내에 OpenMAX 워킹 그룹이 형성되어 멀티미디어 어플리케이션을 위한 Open API의 규

격을 정의하고 있다. 표준 수립의 목적은 새로운 프로세서와 구조에 멀티미디어 소프트웨어를 포팅하는데 필요한 비용과 복잡도를 줄이는 데 있다.

멀티미디어 코덱, 그래픽 라이브러리, 비디오, 오디오, 이미지, 음성 코덱 등 미들웨어들을 공통된 규격으로 정의함으로써 같은 기능을 재구현하는 것을 반복하는 대신 각 제품별 특화에 집중할 수 있도록 한다. 새로운 제품은 시장에 좀 더 빨리 소개될 수 있으며 좀 더 많은 범위의 하드웨어 플랫폼에 대한 지원이 가능해 진다. 그림 10은 OpenMAX을 기반으로 한 멀티미디어 스택 구조를 보여준다[7].

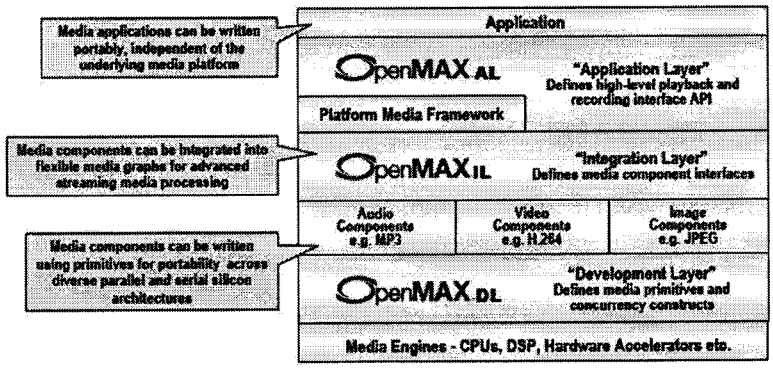

그림 10. OpenMAX를 기반으로 한 멀티미디어 스택의 구조

4.2 OpenMAX AL (Application Layer)

OpenMAX AL API는 어플리케이션과 멀티미디어 미들웨어 사이에 규격화된 인터페이스를 제공한다. 멀티미디어 미들웨어는 멀티미디어를 구동시키기 위한 모든 서비스들을 AL API로 제공함으로써 어플리케이션에게 멀티미디어 인터페이스 관련한 이식성을 제공한다.

OpenMAX AL은 아직 발표되지 않았지만 현재 논의되고 있는 기능들은 다음과 같다.

- Playback: 비디오 파일 실행, 음악 파일 실행, 이미지 파일 출력

- Recording: 비디오 파일 녹화, 오디오 파일 녹음, 이미지 파일 저장

이와 같은 기본적인 동작 외에도 조작을 위한 play, pause, stop, FF, RW같은 기능들을 제공하게 된다. 또한 설정관련해서도 오디오 출력에 대해 크기와 채널에 대해 설정할 수 있으며 비디오 출력에 대해서는 비디오 크기을 설정할 수 있게 된다. 이외에도 스트림으로부터 메타 데이터를 추출하거나 추가할 수 있는 기능도 추가된다. 현재 구조와 API를 명확히 하기 위한 작업이 추진중이며 최초의 규격 문서는 빠르면 2006년 4Q내에 발표될 계획이다.

4.3 OpenMAX IL (Integration Layer)

OpenMAX IL은 모바일 기기에 내장된 오디오, 비디오, 이미지 코덱등을 위한 인터페이스를 제공한다[8]. 이는 어플리케이션과 미디어 프레임워크가 멀티미디어 코덱과 이를 지원하는 모듈들과의 단일화된 인터페이스를 통해 사용할 수 있도록 한다. 코덱 자체는 하드웨어와 소프트웨어의 조합으로 구현되어 있을 수 있으나 사용자에게는 완벽하게 가려질 수 있다. 이러한 특성들의 규격화된 인터페이스가 없다면 코덱 개발자들은 각 모바일 기기마다 정의된 고정되거나 자체 인터페이스를 통해 코덱을 구현해야 한다. IL의 기본적인 목적은 코덱에게 시스템 구조를 추상화함으로써 매우 광범위한 미디어 시스템로의 이식성 문제를 해결하는 것이다.

OpenMAX IL은 Symbian MDF, GStreamer, DirectShow, MMAPI, OpenML등의 기존의 미디어 프레임워크들을 재사용이 가능하도록 통합할 수 있다. 규격에서 지원하지 않는 기존 미디어 프레임워크의 API를 그래도 사용할 수 있도록 확장 가능한 API를 제공하고 미디어 그래프로 만든 유스 케이스를 그대로 재사용 가능하도록 한다. 그림 11은 OpenMAX IL 로 만든 그래프의 예제이다.

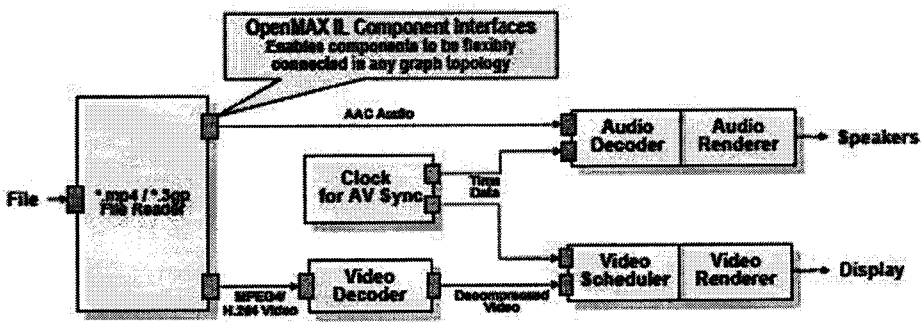

그림 11. MPEG-4 비디오와 AAC 오디오의 동기화

OpenMAX IL은 두 가지 방법의 프로파일을 이용해 컴포넌트를 구성하여 테스트 할 수 있다. 첫째는 base 프로파일로 컴포넌트의 기본 동작에 대해 테스트하는 것이고 두번째는 interop 프로파일은 컴포넌트 간의 상호 동작에 대해 테스트하는 것이다.

OpenMAX IL에서 API로 제공되는 기능들은 다음과 같다.

- Objectives & Profiles
- Component Registration
- Component States
- In-Context/Out-Context Behavior
- Buffer Allocation & Sharing
- Port Reconnection
- Buffer Queue Flush
- Buffer Marking
- Buffer Payload
- Buffer Flags
- Synchronization
- Rate Control

- Resource Management

4.4 OpenMAX DL (Development Layer)

OpenMAX DL API는 오디오, 비디오, 신호 처리에 대한 광범위한 API를 가지고 있다[9]. 해당 API는 다양한 CPU와 하드웨어 엔진을 위해 개발되고 최적화되어 가속화된 코덱 기능을 어플리케이션에게 제공하기 위해 사용된다. API 함수들은 H.264, MPEG-4, AAC, MP3, JPEG등에서의 주요 알고리즘에 대한 것들이다. 이러한 함수들은 코덱 처리에 약 80%를 차지한다. 새로운 하드웨어 플랫폼으로의 코덱 포팅은 새로운 OpenMAX DL 라이브러리로 대체해서 컴파일하는 것으로 간단히 해결할 수 있다.

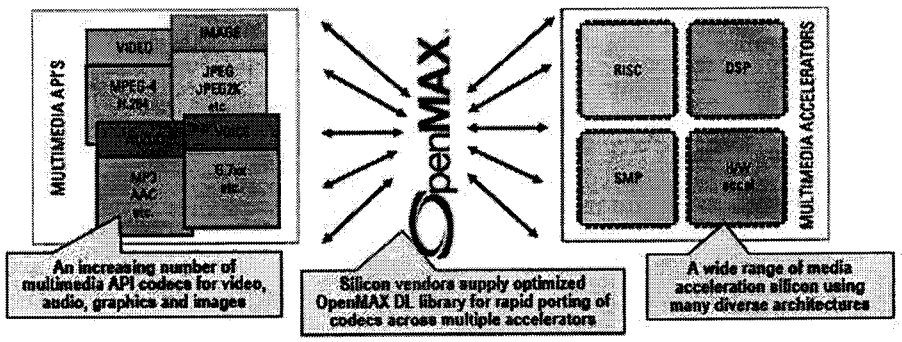

그림 12. OpenMAX DL API의 적용

OpenMAX DL은 코덱 종류에 따라 다음과 같이 여러 도메인으로 나누어져 있다.

■ **비디오 도메인**

블록 단위의 기본적인 비디오 처리에 대한 API이다. 비디오 기본 함수로 다음과 같은 API를 제공한다.

- 8x8 Add, Sub & 16x16 Add, Sub

- 8x8 DCT+Q+Scan & 8x8 IDCT+Q+InvScan

- MPEG-4 VLD (Variable Length Decode)

또한 좀 더 나은 성능을 위해 움직임 예측(Mostion Estimation), 움직임 보상(Motion Compensation), 디블러킹(Deblocking) 같은 한단계 위 레벨의 API도 제공한다.

■ **이미지 도메인**

블록단위의 기본적인 영상 처리에 대한 함수를 제공한다.

- JPEG: encode & decode, 8x8 DCT & 8x8 IDCT, Quantization, Merged DCT & quatization, Huffman encoding & decoding

- Image processing: color space conversion & packing/unpacking, de-blocking/de-ringing filtering, Filtering, Moments, Block copy, rotation, mirroring & scaling

■ **오디오 도메인**

블록 단위의 기본적인 신호 처리 함수를 제공한다.

- Audio API: Unpacking of headers and bit-streams, Huffman decode, IMDCT & MDCT Polyphase filter, TNS & PNS processing

- Singal Processing: FFT & IFFT, FIR, IIR & Median filters, Dot product, Block exponent

OpenMAX DL API는 기본적으로 동기식으로 동작하나 특정한 하드웨어나 여러 개의 프로세서의 동작을 위해 비동기식으로 동작해야 할 필요가 있을 때를 대비하여 aDL (Asynchronous DL) API도 제공한다. 또한 OpenMAX iDL도 제공하는 데 이는 OpenMAX IL 구조를 활용한 것으로 OpenMAX aDL과 같은 효과가 있다.

OpenMAX는 기본적으로 멀티미디어 코덱, 게임 엔진, 그래픽 라이브러리를 구현하는 미들웨어 개발자들을 대상으로 하고 있다. OpenMAX는 스마트폰, 게임

콘솔, 디지탈 텔레비젼, 셋탑 박스 같은 멀티미디어 성능이 중요한 모든 어플리케이션들에 적용 가능하다.

4.5 OpenMAX의 적용

OpenMAX IL은 여러 회사에서 적용할 계획이 있으며 이와 관련해 구현된 소소를 크로노스 그룹에 공개하고 있다. ST에서 ffmpeg기반의 ALSA와 MP3 IL 컴포넌트를 구현하여 공개하였다. TI에서는 리눅스를 기반으로 H.263, Narrow Band AMR, Baseline JPEG 코덱들의 OpenMAX IL 예제 코드들을 공개했다.

■ **ARM OpenMAX DL 라이브러리**

OpenMAX DL 적용의 가장 큰 동기는 ARM에서 지원하기 시작했다는 것이다. ARM에서는 구현 샘플로 C 버전의 OpenMAX DL 라이브러리를 무료로 배포하고 있다. 또한 어셈블리로 최적화된 ARM용 OpenMAX DL 라이브러리를 작성하고 있다. ARM에서 현재 OpenMAX DL 라이브러리를 최적화 하고 있는 칩셋은 다음과 같다[10].

- ARMv6 SIMD: ARM11 계열의 프로세서들의 ARMv6 구조를 활용하여 최적화 될 것이다.

- ARMv7/NEON: Cortex-A8 프로세서의 강력한 NEON 신호 처리 기술을 활용해서 최적화될 것이다[11].

4.6 GStreamer 플러그인으로의 OpenMAX IL

크로노스 그룹의 OpenMAX 사이트에서는 GStreamer와 OpenMAX IL 구조의 유사점을 들어 두 프레임워크가 하나의 멀티미디어 프레임워크로 쉽게 통합될 수 있음을 보여주고 있다[12]. GStreamer안에서 OpenMAX IL 컴포넌트를 GStreamer 플러그인 형태로 사용하는 구조이다. 그림 13은 GStreamer 플러그인 엘리먼트가 OpenMAX IL 컴포넌트를 OpenMAX core와 OpenMAX IL API를 통해 사용할 수 있음

을 보여주고 있다.

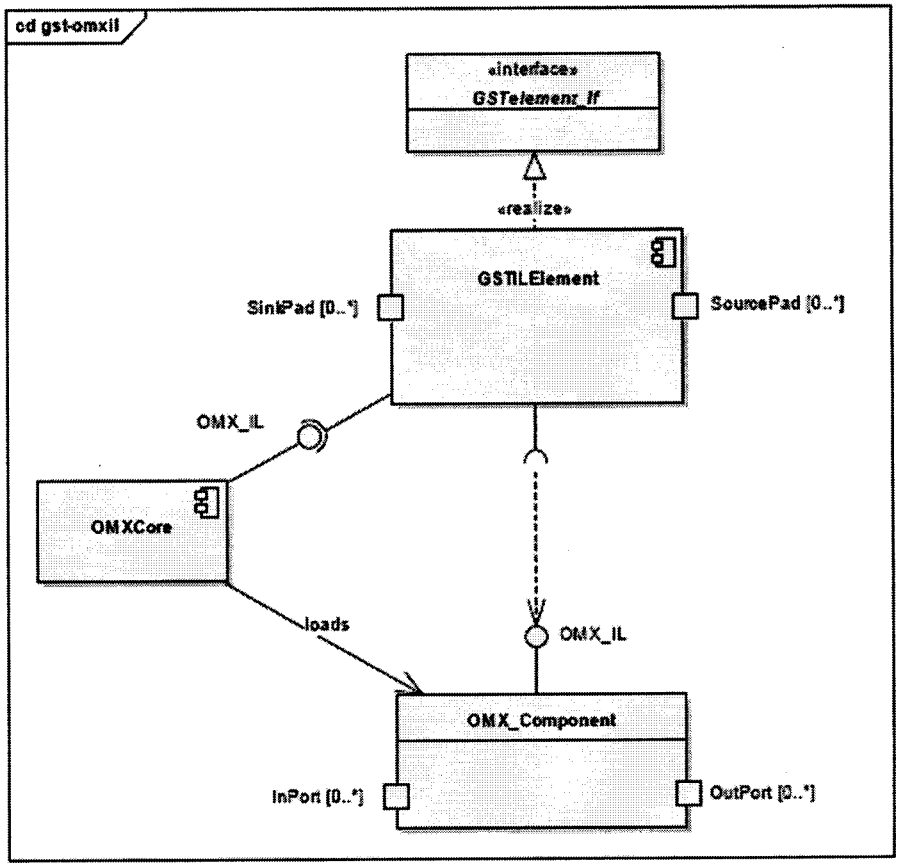

그림 13. GStreamer 엘리먼트와 OpenMAX IL 컴포넌트

OpenMAX IL API를 통해 GStreamer 엘리먼트는 OpenMAX IL 컴포넌트내의 데이터 버퍼의 할당과 사용을 관리할 수 있을 뿐더라 파라미터를 통해 컴포넌트의 설정을 수정할 수 있다. OpenMAX IL 구조에 따르면 GStreamer 엘리먼트는 OpenMAX IL 클라이언트로 동작하며 GStreamer 엘리먼트의 패드는 OpenMAX 컴포넌트의 포트로 동작할 수 있다. 패드와 포트의 동작에 있어 가장 큰 차이점은 패드가 GStreamer 엘리먼트에 능동적으로 추가/삭제될 수 있지만 포트는 고정적이며 실행중일 때 사용할 것인지 아닌지만을 결정할 수 있다.

GStreamer와 OpenMAX IL 간의 기능과 용어에 대한 차이점을 표 1에 나타내었다.

표 1. GStreamer와 OpenMAX IL의 기능 비교

GStreamer	OpenMAX IL	Remarks
element	component	기본 인터페이스 블록
sink pad	input port	데이터를 받아들이는 인터페이스
source pad	output port	데이터를 내보내는 인터페이스
property	param & config	속성
bin, pipeline	-	OpenMAX에는 없는 연결 고리에 대한 개념
plugin	-	GStreamer 엘리먼트의 컨테이너
event	event	GStreamer는 이벤트 전달 기능 제공 OpenMAX는 IL 클라이언트에 구현
buffer	buffer	데이터를 담고 있는 단위
caps	port definition	가능한 데이터 포맷에 대한 범위

GStreamer 엘리먼트와 OpenMAX IL 컴포넌트의 상태는 매우 비슷하다. 이에 대한 비교를 표 2에 나타내었다.

OpenMAX IL은 GStreamer에는 정의되어 있지 않은 OMX_STATEINVALID, OMX_STATEWAITFORRESOURCES 두 상태를 가지고 있다. 그 중 첫번째는 IL 클라이언트에 의해 OpenMAX 컴포넌트가 언로딩되거나 복구할 수 없는 에러가 발생했을 때의 상태를 말하며 두번째는 하드웨어 자원이 서로 충돌되었을 때의 컴포넌트의 상태를 말한다.

표 2. GStreamer와 OpenMAX IL의 상태 비교

GStreamer 엘리먼트 상태	OpenMAX 컴포넌트 상태
GST_STATE_VOID_PENDING	
GST_STATE_NULL	OMX_STATELOADED
GST_STATE_READY	OMX_STATEIDLE
GST_STATE_PLAYING	OMX_STATEEXECUTING
GST_STATE_PAUSED	OMX_STATEPAUSE
	OMX_STATEWAITFORRESOURCES
	OMX_STATEINVALID

OpenMAX와 GStreamer 상태 사이의 가장 큰 차이는 상태 변경에 있다. GStreamer 어플리케이션은 엘리먼트의 상태를 NULL에서 PLAYING으로 변경할 것을 요청할 수 있다. 그러면 GStreamer 프레임워크는 상태 변이에 따른 중간 매개체들을 관리한다. 그러나 OpenMAX에서는 IL 클라이언트가 컴포넌트의 상태 변경를 요청할 수 없다.

표 3. GStreamer와 OpenMAX IL의 함수 비교

목적	GStreamer 함수	OpenMAX IL 함수
설정 초기화	gst_init()	OMX_Init()
엘리먼트 생성	gst_element_factory_make()	OMX_GetHandle()
상태 변경	gst_element_set_state()	OMX_CommandSendCommand()
엘리먼트 연결	gst_element_link() gst_element_link_pads()	OMX_SetupTunnel()
초기화시 속성 Set/Get	g_object_set() g_object_get()	OMX_SetParameter() OMX_GetParameter()
실행시 속성 Set/Get	g_object_set() g_object_get()	OMX_SetConfig() OMX_GetConfig()
버퍼 할당	gst_buffer_new()	OMX_UseBuffer()

	gst_buffer_new_and_alloc()	OMX_AllocateBuffer()
버퍼 해제	gst_buffer_unref()	OMX_FreeBuffer()

표 3에 GStreamer 어플리케이션과 IL 클라이언트에서 같은 목적으로 사용될 수 있는 함수들에 대해 비교하였다. 앞에서 언급한 대로 OpenMAX IL API를 사용하는 GStreamer 엘리먼트는 IL 클라이언트로 동작한다. 그래서 엘리먼트내의 GStreamer 함수는 관련된 OpenMAX IL 함수를 부르는 것으로 구현될 수 있다.

위의 API간의 비교를 통해 알아본 것처럼 GStreamer와 OpenMAX IL 함수를 서로 비교하여 같은 역할을 하는 함수를 호출했을 때의 GStreamer 플러그인에서 OpenMAX IL 컴포넌트로의 데이터 흐름을 그림 14과 같이 나타내었다. 다음에 각 단계에 대한 설명을 기술하였다.

1.0: 프레임워크는 엘리먼트 상태를 NULL에서 READY로 변경하도록 요청한다.

1.1: OpenMAX를 초기화한다.

1.2: 엘리먼트에서 OpenMAX 컴포넌트를 생성한다.

1.3: 생성된 컴포넌트의 파라미터들을 설정한다.

1.4: 컴포넌트를 IDLE 상태로 변경한다.

1.5: 입력 포트를 위해 버퍼가 할당된다. 이때 버퍼는 GStreamer에 의해 이미 할당된 상태이다.

1.6: GStreamer 엘리먼트는 컴포넌트에게 출력 포트를 위한 버퍼를 할당하도록 요청한다.

1.8: 엘리먼트의 상태가 READY에서 PAUSED로 변경된다.

1.9: 엘리먼트는 OpenMAX 컴포넌트가 IDLE로 변경되어 모든 필요한 자원이 할당될 때까지 기다린다.

1.12: PAUSED에서 PLAYING으로 상태가 변경된다.

1.13: 몇 개의 비어있는 버퍼가 출력 포트로 보내진다.

1.14: 컴포넌트가 EXECUTING상태가 될때까지 기다린다.

2.0: 프레임워크가 데이터 처리를 위해 버퍼를 보낸다.

2.1: 버퍼가 OpenMAX 컴포넌트로 전달된다.

2.2: 컴포넌트는 사용할 수 있는 출력 버퍼를 가진 상태가 된다.

2.3: 버퍼는 다시 프레임워크로 돌아가고 이것은 파이프라인에 있는 다음 엘리번트로 보내질 것이다.

2.4: gst_pad_push ()가 리턴되면 버퍼는 재사용될 수 있다.

2.6: 입력 버퍼에 대한 처리가 완전히 끝난 상태이다.

2.7: gst_chain () 수행값이 리턴된다.

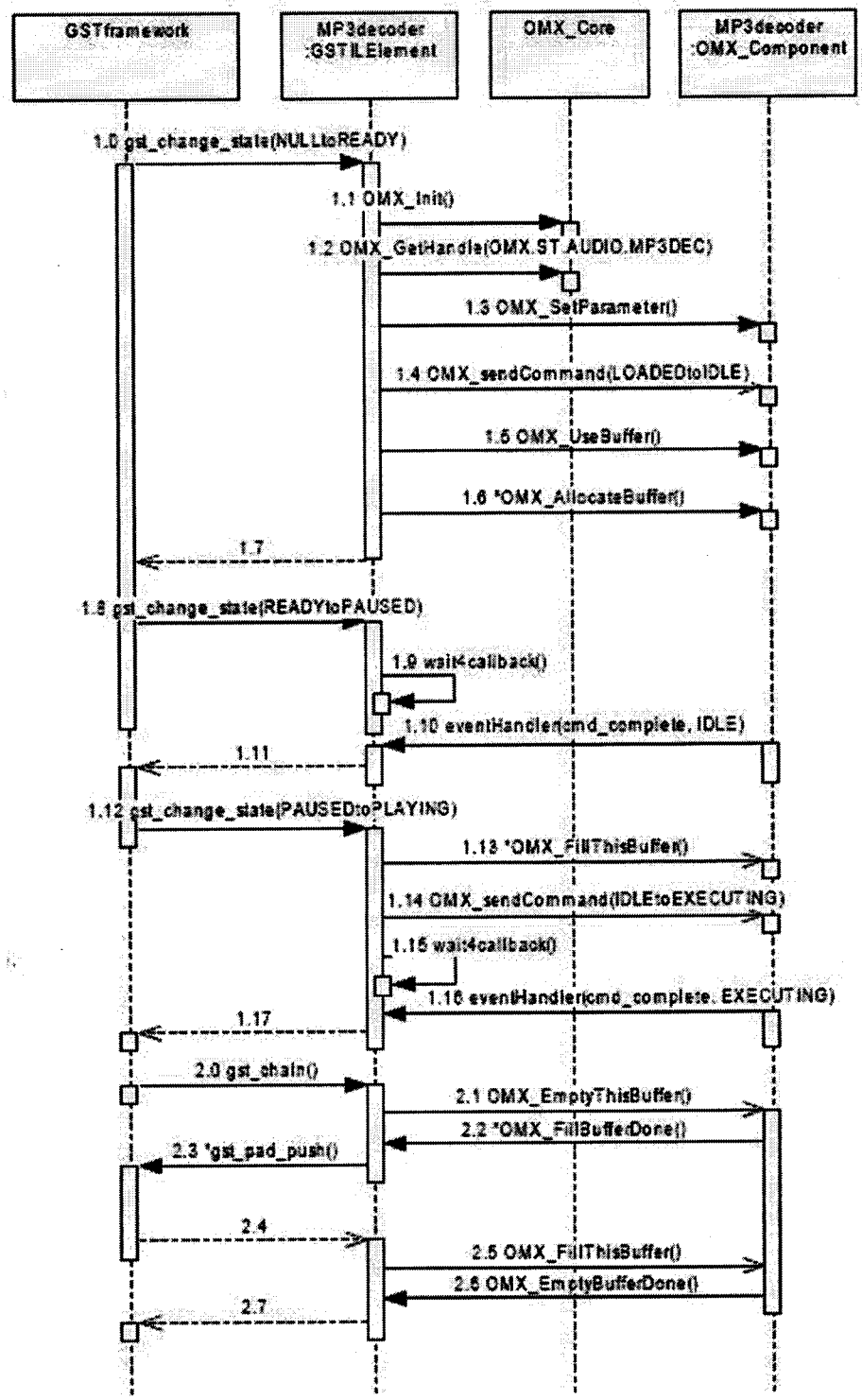

그림 14. GStreamer에서 OpenMAX IL 코덱 컴포넌트의 사용

제 5 장 리눅스 GStreamer 기반의 모바일 멀티미디어 프레임워크

5.1 모바일 소프트웨어 플랫폼

모바일 플랫폼이란 모바일 기기에 탑재되어 해당 기기에 다양한 어플리케이션을 실행할 수 있도록 하는 하드웨어 및 소프트웨어에 대한 프레임워크를 기술하는 것이다. 하드웨어 측면에서의 어플리케이션 프로세서 칩셋과 무선 네트워크 연결을 위한 모뎀 칩셋외에도 이를 활용하여 어플리케이션에게 커널, 미들웨어, 어플리케이션 프레임워크를 제공해 줄 수 있는 모바일 소프트웨어 플랫폼은 모바일 플랫폼 구조 중 핵심이다.

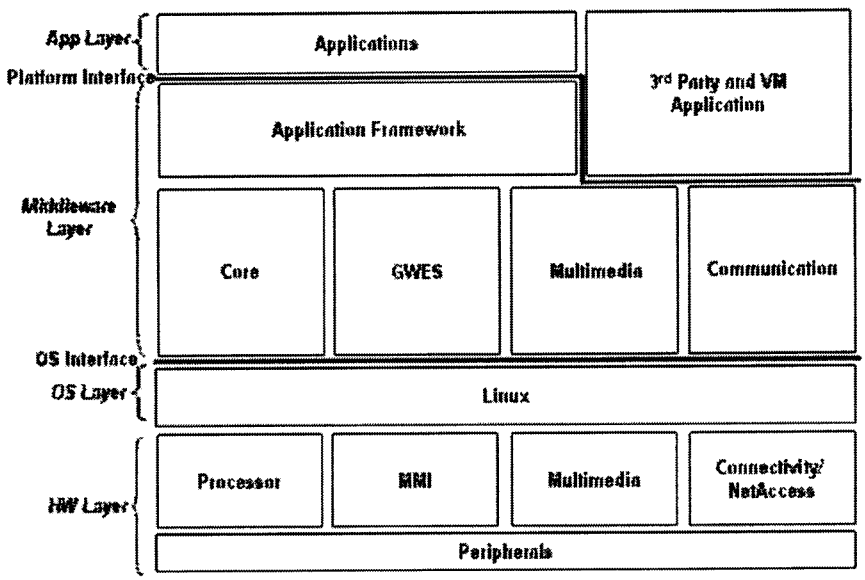

그림 15. 모바일 소프트웨어 플랫폼의 구조

소프트웨어 측면에서의 모바일 기기 구조의 기술 요소인 모바일 소프트웨어 플랫폼은 그림 15과 같은 하위 시스템들로 구성되어 있으며 그림에는 리눅스 기반의 모바일 소프트웨어 플랫폼의 구조를 나타내었다. 모바일 소프트웨어 플랫폼은 다음과 같은 기능을 제공한다.

- OS/Core: 성능, 보안, 안전성을 위한 MPSoc 기반의 시스템 서비스
- Communication: 데이터 커뮤니케이션 프로토콜과 연결 관리 서비스
- GWES: 윈도우 시스템과 2D/Vector/3D 그래픽스
- Multimedia: 멀티미디어 처리와 표현
- Application Framework: UI와 배치를 위한 어플리케이션 서비스/엔진

모바일 소프트웨어 플랫폼내의 서브 시스템들의 주요 기능을 알 수 그림 16에 한 단계 상세한 수준의 모바일 소프트웨어 플랫폼의 구조를 나타내었다.

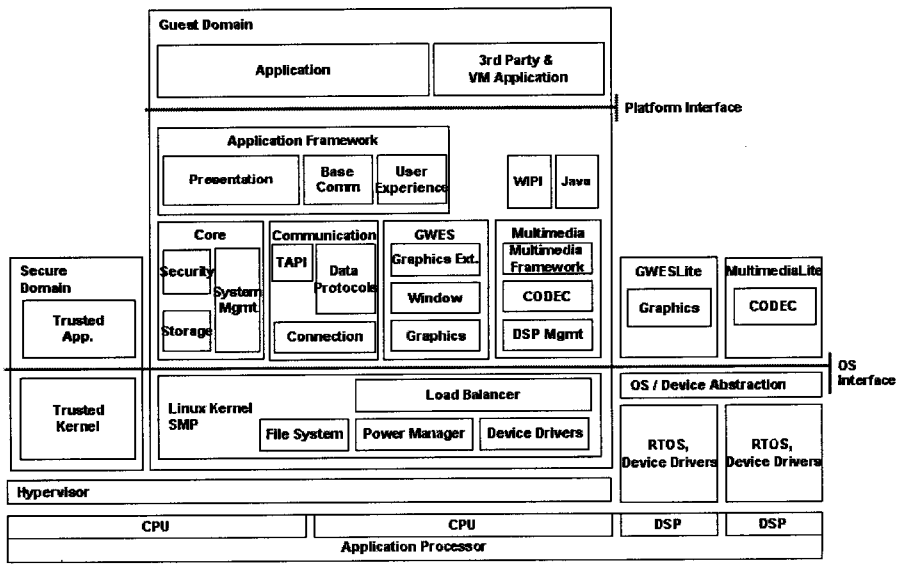

그림 16. 모바일 소프트웨어 플랫폼의 상세 구조

그 중에서도 모바일 기기에 있어 멀티미디어 기능에 대한 요구 사항은 특히 갈수록 높아지고 있으며 동시에 다양한 멀티미디어에 특화된 모바일 기기들이 생산되고 있다. 이러한 시장 상황에서 대응하기 위해서는 모바일 소프트웨어 플랫폼내의 유연성 있고 안정성 있는 멀티미디어 프레임워크의 설계가 필수적이다.

5.2 멀티미디어 프레임워크의 설계 시 고려사항

모바일에서의 지원되어야 하는 멀티미디어 기능의 종류와 복잡성이 점점 증가함에 따라 다양한 멀티미디어 어플리케이션 개발을 위한 안정적이면서도 유연성 있는 개발 플랫폼으로써의 멀티미디어 프레임워크에 대한 요구가 높아지고 있다. 이러한 플랫폼은 개발을 쉽게 빠르게 할 수 있으면서도 소프트웨어의 품질을 향상시킬 수 있는 메커니즘과 기능을 제공할 수 있어야 한다.

모바일에서의 멀티미디어 컨텐츠들의 해상도와 품질이 점점 높아짐에 따라 CPU 혹은 DSP에 최적화된 전용 코덱들이 사용되고 있으며 이에 따라 멀티미디어 프레임워크의 하드웨어에 대한 의존성이 증가되고 있다. 그러나 멀티미디어 어플리케이션이 하드웨어의 구성에 영향을 받는다면 해당 소프트웨어의 모듈성과 재사용성이 떨어지므로 결국 소프트웨어의 수명은 단축될 것이며 하드웨어가 변경될 때마다 일어나야 하는 새로운 소프트웨어의 개발은 비용 증가와 시장 진입의 시기를 놓치는 결과를 초래할 것이다. 그러므로 멀티미디어 어플리케이션에는 하드웨어에 독립적인 개발 환경을 제공해주는 멀티미디어 프레임워크이 필요하다. 이와 동시에 하드웨어 변경이 일어나더라도 이를 쉽게 대처할 수 있는 멀티미디어 프레임워크에서의 포팅 레이어가 구비되어 있어야 한다.

그리고 네트워크 사업자가 상호 운용성의 보장을 내세워 자신들이 운용하는 멀티미디어 서버와 같은 회사의 멀티미디어 클라이언트 솔루션을 채용할 것을 요청하는 일도 빈번하다. 그러므로 멀티미디어 프레임워크은 이러한 상황에

대처하여 모듈화를 하여 쉽게 다른 솔루션으로 변경할 수 있어야 하며 이러한 변경이 멀티미디어 어플리케이션에 끼치는 영향을 최소화 할 수 있어야 한다.

본 논문에서 제안하는 멀티미디어 프레임워크는 다음과 같이 다양한 종류의 모바일 기기에 탑재되는 것으로 가정한다.

- Smartphone: Basic Smart Phone, Enhanced Smart Phone
- Mobile Phone: Basic Phone, Enhanced Phone (Music Phone, Camera Phone, Video Phone, TV Phone, Multi-mode/Multi-band Phone)
- PMD (Personal Multimedia Device): PMP (Portable Multimedia Player), Digital Audio Player (MP3P), Digital Camcorder, Digital Camera, Mobile TV, Portable Came Player
- PDA (Personal Digital Assistant): Telematics (Car Navigation Device, Pedestrian Navigation Device), Protable Internet (Wibro Device, Multiple N/W Access Device)

탑재되는 모바일 기기의 하드웨어은 다음과 같은 사양을 갖출 것이라 가정한다.

- MPSoC hardware: 멀티미디어 프레임워크는 다양한 MPSoC위에 도입될 수 있다. MPSoC는 1-N개의 multi-CPU와 multi-DSP로 구성되어 있는 하드웨어를 뜻한다.
- Core hardware performance
  . Flash Memory: 256MB (실행할 프로그램의 코드 크기)
  . RAM: 228MB (런타임시 사용되는 메모리 크기)
  . CPU: ARM11 동급 또는 이상
  . DSP: StarCore2400 동급 또는 이상
- Screen Resolution: 모바일폰에서 많이 채용하는 QCIF(176x144), QVGA(320x240), 또는 VGA(640x480) 해상도를 사용한다.

- Periperal hardware performance
  . Camera: YUV 4:2:0, VGA(640x480) 이상, 60 fps
  . Speaker, Mic, Ear-phone, HFK, Line-In
  . LCD, TV out
  . TV tuner
- Hardware accelerated CODEC

■ **성능 (Performance)**

멀티미디어 어플리케이션에서 있어 가장 중요시 요구되는 것은 성능이다. 멀티미디어 컨텐츠들의 품질과 해상도가 점점 높아져 가고 있지만 이를 수행하는 데 걸리는 시간에는 지연이 있어서는 안된다.

■ **구성력 (Configurability)**

앞에서 나열한 해당 멀티미디어 프레임워크가 목표로 하는 모바일 기기들의 예에서 보았듯이 지원하는 기기에 따라 다양한 서비스들을 제공하기 위해서는 필요한 모듈 또한 매우 다양하다. 예를 들어, 카메라, 캠코더, 화상 전화등은 스마트폰에는 들어가지만 PMP (Personal Multimedia Player)에는 필수가 아니다. 이러한 다양한 서비스에 맞는 모듈들을 기기마다 손쉽게 재구성할 수 있어야 한다.

또한 각각의 기기들은 서로 다른 LCD 크기와 해상도를 가질 뿐만 아니라 다양한 사운드 출력 사양을 가지고 있다. 멀티미디어 프레임워크는 이러한 하드웨어 사양에 따라 멀티미디어 컨텐츠들을 손쉽게 조작할 수 있도록 해주어야 한다.

■ **신뢰성 (Reliability)**

멀티미디어 어플리케이션이 불법적으로 만들어진 혹은 유통되는 멀티미디어 컨텐츠에 접속하거나 다운받지 못하도록 방어해야 한다.

■ **사용성 (Usability)**

해당 멀티미디어 프레임워크에서 지원하려고 생각하는 어플리케이션은

GStreamer 기반 어플리케이션, Mozilla 기반 어플리케이션, OpenMAX AL 기반 어플리케이션으로 모두 세 가지이다. GStreamer 기반으로 설계되므로 기존의 이미 GStreamer 기반으로 만들어진 어플리케이션들은 기존의 동작을 보장받아야 하며 새로 만들어지는 어플리케이션들이 GStreamer core API를 자유롭게 사용할 수 있도록 열어 주어야한다. Mozilla 기반 어플리케이션들을 위해 Mozilla 플러그인으로 멀티미디어 프레임워크 기능들을 제공해 주어야한다. 마지막으로 OpenMAX AL을 기반으로 한 어플리케이션 작성이 가능하도록 해주어야한다.

■ **이식성 (Portability)**

멀티미디어 어플리케이션은 하드웨어 사양이나 구성에 대한 고려없이 다양한 코덱을 사용할 수 있어야 한다. 이는 코덱뿐 아니라 하드웨어에 포함된 다른 주변 기기에 대해서도 마찬가지로 적용되어야 한다.

■ **확장성 (Extensibiltiy)**

멀티미디어 프레임워크의 개발이 끝나 상품화가 된 경우에라도 새로운 컨텐츠 타입에 대한 요구가 있을 경우 플러그인을 사용하여 새로운 코덱을 추가할 수 있어야 한다. 이는 코덱 뿐 아니라 다른 멀티미디어 기능에 대한 업그레이드나 추가에도 마찬가지로 적용되어야 한다.

5.3 모바일 멀티미디어 프레임워크 설계

■ **시스템 컨텍스트**

본 논문에서 제안하는 멀티미디어 프레임워크의 시스템 컨텍스를 그림 17에 나타내었다. 그림은 모바일 소프트웨어 플랫폼 안에서 멀티미디어 프레임워크과 다른 모듈간의 관계에 대해 나타내고 있다. 멀티미디어 프레임워크는 외부적으로는 GStreamer 기반의 멀티미디어 어플리케이션, GTK+ 기반의 멀티미디어 어플리케이션, Mozilla 기반의 멀티미디어 어플리케이션을 지원하도록 설계하였다. GStreamer core API를 그대로 공개함으로써 기존 공개 소스 형태의 GStreamer

기반의 멀티미디어 어플리케이션들이 수정없이 동작되도록 한다. 또한 모바일 소프트웨어 플랫폼에서는 GTK+ 기반의 어플리케이션을 지원하므로 해당 어플리케이션에서도 멀티미디어 프레임워크를 통해 OpenMAX AL API나 GStreamer core API를 제공하여 멀티미디어 기능을 구현하도록 지원한다. Mozilla 기반의 어플리케이션을 위해서는 Mozilla plugin 형태로 멀티미디어 기능을 제공한다.

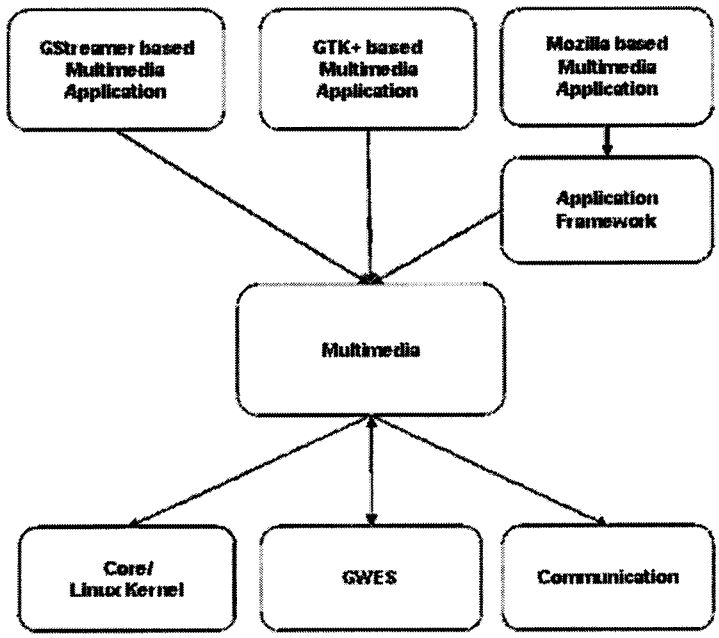

그림 17. 멀티미디어 프레임워크의 시스템 컨텍스트

플랫폼 내부적으로는 파일이나 메모리등의 기본적인 기능을 위해 Core나 리눅스 커널 API를 사용하며 영상 출력등을 위해 윈도우 시스템인 GWES API를 사용한다. 스트리밍이나 화상 전화등의 네트워크를 사용하는 경우 Communication의 프로토콜 API를 사용한다.

■ 제안하는 멀티미디어 프레임워크의 구조

제안하는 멀티미디어 프레임워크는 공개 소스인 GStreamer를 기반으로 설계되었으며 그 구조는 그림 18에 나타내었다.

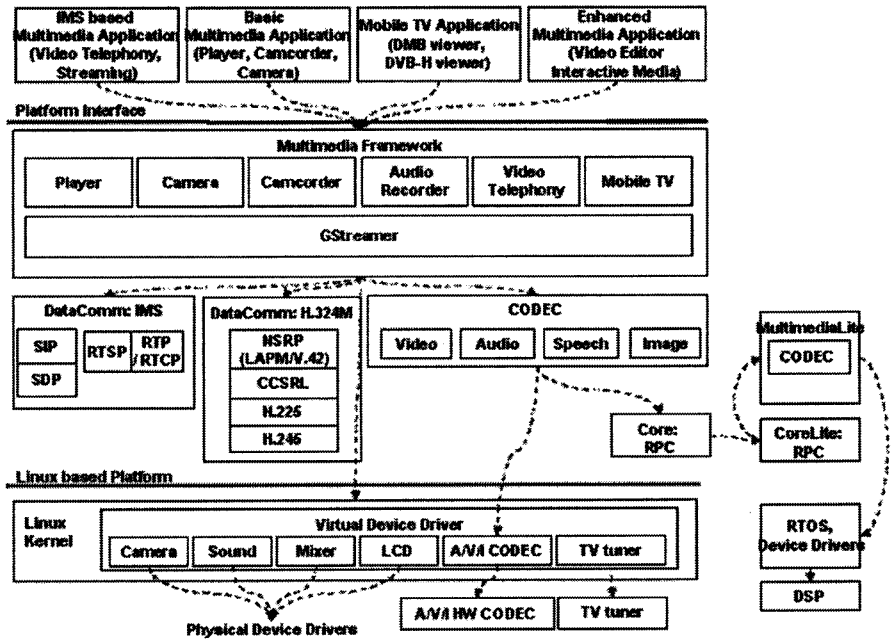

그림 18. 제안하는 멀티미디어 프레임워크의 구조

■ 서비스 및 기능

그림 18에서 멀티미디어 프레임워크 안의 플레이어, 카메라, 캠코더, 오디오 레코더, 화상 전화, 모바일 TV는 멀티미디어 프레임워크의 기능을 나타내고 있는 블록들로 어플리케이션에게 해당 서비스를 위한 API를 제공한다. 플레이어는 로컬 파일 형태의 멀티미디어 컨텐츠들과 스트리밍 서버로부터 실시간으로 받은 멀티미디어 컨텐츠들을 실행할 수 있는 기능을 제공한다. 카메라는 모바일 기기의 카메라 하드웨어를 컨트롤하여 사용자가 사진을 촬영하여 이미지로 저장해주는 기능을 제공한다. 캠코더와 오디오 레코더는 카메라와 마이크를 이용해

비디오로 녹화하거나 목소리를 녹음할 수 있는 기능을 제공한다. 화상 전화는 3G 망에서 SIP 기반의 1:1 화상 전화 기능을 제공하며 모바일 TV는 DMB/DVB-H, IP기반의 스트리밍 TV, 3G 네트워크에서의 MBMS 기반의 방송 등 다양한 전송방식에서의 방송 서비스을 제공한다.

■ **GStremer 기반 멀티미디어 프레임워크**

각 멀티미디어 기능에 대해 제안하는 멀티미디어 프레임워크은 리눅스 기반의 GStreamer 를 채택하여 모바일 플랫폼에 맞게 플러그인들을 선택, 구성하여 최적화된 모듈을 사용하도록 한다.

■ **OpenMAX의 적용**

멀티미디어 컨텐츠의 해상도와 품질에 대한 요구가 높아짐에 따라 이를 사용자가 원하는 수준에 맞게 실행할 수 있는 성능이 요구된다. 멀티미디어 프레임워크에서는 이를 실현하기 위해서 multi-core & multi-DSP 기반의 모바일 플랫폼에서 높은 성능이 요구되는 코덱등을 별도의 DSP로 실행 가능하도록 컨트롤할 수 있다. 또한 코덱을 플러그인 구조로 모바일 기기의 사양에 따라 H/W 코덱과 S/W 코덱을 유연하게 선택할 수 있도록 OpenMAX의 IL과 DL API를 도입하여 코덱 플러그인 구조을 구축한다.

■ **어플리케이션 지원**

멀티미디어 프레임워크의 API는 어플리케이션에 의해 직접 호출될 수 있으며 브라우저나 Mozilla 기반 어플리케이션을 위한 플러그인 형태로 제공될 수 있다. 또한 기존의 GStreamer 기반의 어플리케이션을 지원하기 위해서 GStreamer API를 그대로 제공한다.

■ **코덱**

최근 모바일 기기에 요구되는 코덱의 예는 다음과 같다. 멀티미디어 프레임워크에서 지원할 코덱의 범위와 종류는 타겟 사양에 따라 다르게 구성될 수 있다. 또한 모바일 기기의 하드웨어 사양 에 따라 S/W 혹은 H/W 코덱을 사용할 것인지 여부가 결정될 것이다.

- Speech: AMR, G.711, G.723, G.726, G.729, QCELP, EVRC
- Audio: AAC LC, AAC+SBR, AAC+BASC, MP3, WMA, OGG Vovis
- Image: PNG, GIF, WBMP, Windows BMP, TIFF, JPEG2000
- Video: H.263, MPEG4 SP, DivX, XviD, H.264, WMV10

5.4 멀티미디어 프레임워크의 상세 구조

가장 핵심인 멀티미디어 프레임워크에 대한 상세한 구조를 그림 19에 나타내었다. 가장 상위 레벨은 OpenMAX AL 모듈이다. 4장에서 기술한 대로 OpenMAX AL은 하부 구조에 관계없이 어플리케이션에게 동일한 API를 제공함으로써 이식성을 높여주는 모듈이다. 현재 크로노스 그룹에서 AL은 발표 예정이므로 다른 모듈과 달리 점선으로 표시하였다. AL 혹은 GStreamer 기반 어플리케이션에서 멀티미디어 기능을 쉽게 구현하도록 하기 위해서 GStreamer 빈 구조를 이용하여 각각 기능별 빈을 만들어 구성한다. Playbin은 로컬 파일 실행 및 스트리밍을 위해 파이프라인을 자동적으로 생성하여 수행 해주는 엘리먼트이며 Cambin은 카메라 관련하여 설정하고 스냅샷을 찍어 파일에 저장하는 데 필요한 파이프라인을 자동적으로 만들어 수행해 주는 엘리먼트이다. Recordbin은 마이크나 카메라를 통해 녹음이나 녹화하는 필요한 파이프라인을 구성하여 자동적으로 수행해 주는 엘리먼트이고 VTbin은 비디오 텔레포니를 위한 프로토콜과 코덱들을 파이프라인으로 구성하여 수행해 주는 엘리먼트이다. 마지막으로 MTVbin은 DMB나 DVB-H 같은 모바일 TV의 기능을 위한 파이프라인을 구성하여 수행해 주는 엘리먼트이다. 이러한 빈 엘리먼트들은 사용자가 입력과 출력 형태를 정하면 각각의 빈의 기능에 해당하는 엘리먼트들을 자동적으로 찾아내어서 파이프라인을 형성함으로써 어플리케이션이 일일이 엘리먼트들을 만들어 등록하는 수고를 덜 수 있을 뿐만 아니라 잘못된 엘리먼트를 연결하여 생기는 오류를 방지할 수 있다. 그림의 소스, 코덱/필터, 싱크 플러그인 엘리먼트들은 파이프라인을 구성하는

가장 기본이 되는 블록들이다.

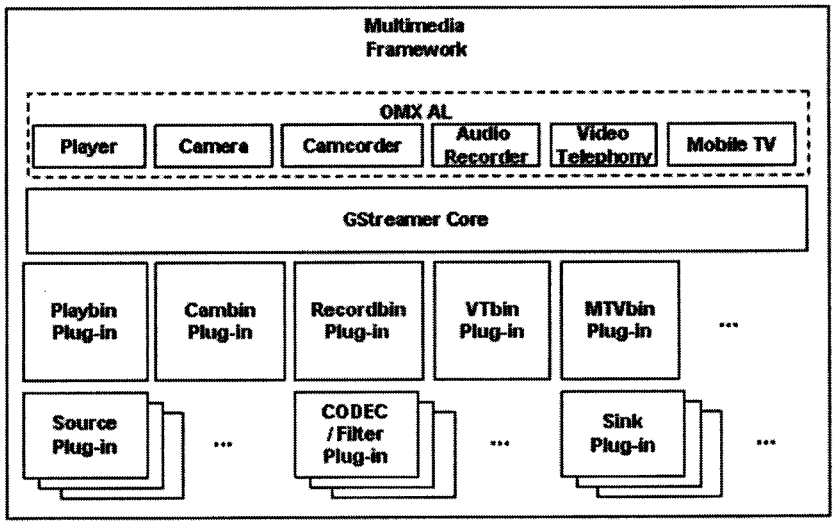

그림 19. 멀티미디어 프레임워크의 상세 구조

그림 20는 멀티미디어 프레임워크에서 사용되는 코덱들의 구조를 종류대로 그린 것이다. 제 4 장에서 언급한 대로 OpenMAX IL 구조를 GStreamer 구조에 통합하는 형태를 가지고 있다. 그림에서 OpenMAX IL Core는 단지 OpenMAX 컴포넌트를 로딩/언로딩하는 역할만 하며 GStreamer core에서 플러그인을 로딩/언로딩하는 함수에 매핑되어 동작한다. GStreamer에서 볼 때 코덱은 하나의 플러그인으로 동작하기 때문에 어플리케이션에서는 아래와 같은 OpenMAX와 통합된 구조임을 알 필요가 없다. GStreamer의 플러그인을 수행하기 위해 불리는 GStreamer 플러그인 API들은 OpenMAX 컴포넌트 API와 매핑되어 결국엔 OpenMAX 컴포넌트로 구현된 코덱을 호출하여 사용하게 된다. OpenMAX DL API는 MPEG-4, H.264, MP3, AAC, JPEG같이 높은 압축율을 가지는 코덱을 위해서 하드웨어 전용 엔진이나 DSP를 사용해야하는 블록 단위의 연산 함수들이다. OpenMAX DL을 적용함으로써 하드웨어가 변경되더라도 API의 변경없이 코덱을 일관성 있게 사용할 수 있다.

그림에서 가장 아랫부분의 블록은 코덱의 구성에 따른 종류를 나타낸 것으로 첫 번째 소프트웨어로만 되어 있는 코덱은 OpenMAX DL API가 정의되어 있지 않은 코덱에 주로 적용될 수 있는 구조로 JPEG을 제외한 이미지 코덱이나 좀 더 간단한 형태의 비디오 코덱이나 오디오 코덱에 해당하는 구조이다. 두번째 OpenMAX DL과 소프트웨어로 이루어진 코덱의 경우에는 특정 CPU(ARM11같은)나 DSP 명령어등의 어셈블리로 최적화된 코덱에 적용될 수 있는 구조이다. 세번째 OpenMAX DL과 하드웨어로 이루어진 코덱의 경우에는 OpenMAX DL의 API들을 하드웨어 IP로 제공되어 처리될 수 있는 코덱의 구조이다. 마지막으로 하드웨어 코덱으로만 되어 있는 코덱은 OpenMAX DL이 정의되어 있는 않은 코덱중 하드웨어 IP로만 되어 있거나 OpenMAX DL이 정의된 코덱이라도 하드웨어에서 코덱의 전체 알고리즘을 지원하여 OpenMAX IL에서 바로 하드웨어 IP를 불러서 사용하는 구조이다.

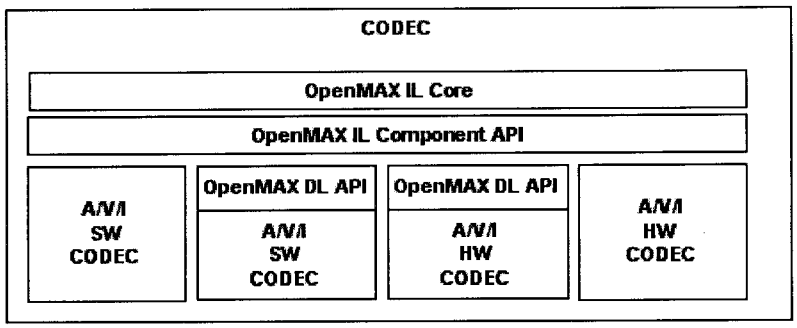

그림 20. 멀티미디어 프레임워크를 위한 코덱의 구조

다음 그림 21에는 멀티미디어 프레임워크에서의 코덱의 동작을 단계별로 나타내고 있다.

1. GStreamer 플러그인 API를 통해 플러그인을 로딩하려고 하면 GStreamer 코덱 플러인에서는 OpenMAX IL core에게 원하는 코덱의 컴포넌트를 로딩하도록 요청한다.

2. OpenMAX IL core에서는 해당 코덱의 컴포넌트를 로딩하여 GStreamer 코덱 플러그인에게 해당 컴포넌트의 핸들을 넘겨준다.
3. GStreamer 코덱 플러그인에서는 넘겨받은 컴포넌트의 핸들을 이용해 인코딩/디코딩 할 데이터를 보내고 관련 파라미터들을 설정한다.
4. 해당 코덱의 OpenMAX 컴포넌트는 각 종류에 따라 OpenMAX DL API를 호출하거나 혹은 core의 RPC 모듈을 통해 DSP의 코덱수행을 요청하는 등의 실제적인 데이터 처리를 시작한다.

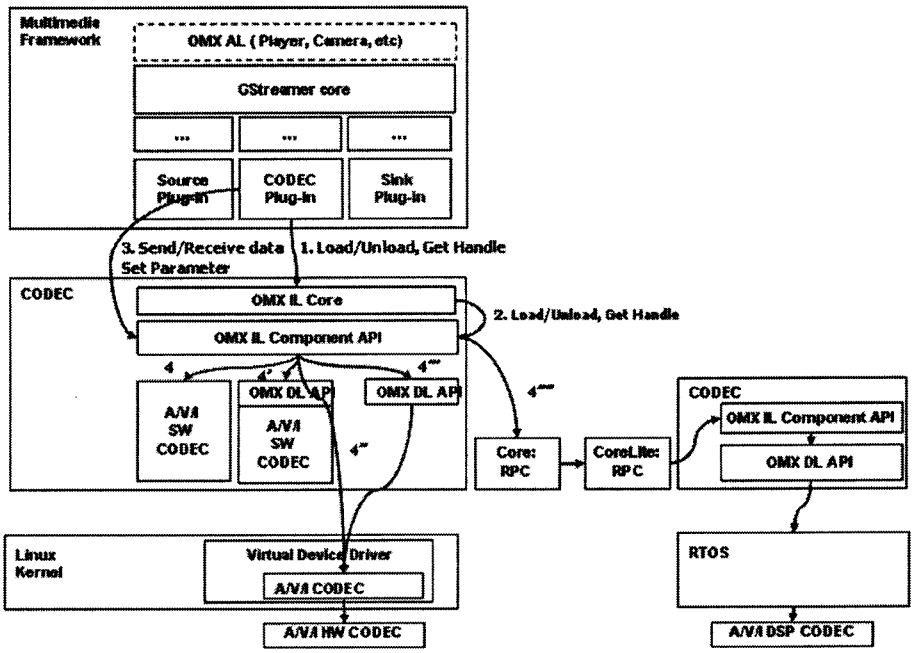

그림 21. 멀티미디어 프레임워크에서 코덱의 동작

그림 22는 멀티미디어 프레임워크를 위한 리눅스 커널과 하드웨어의 구조에 대해 나타내었다. 가장 윗부분의 core는 멀티미디어를 위한 전용 DSP가 존재할 때 멀티미디어 프레임워크에서 요청한 코덱의 수행을 DSP에서 실행하고 그 결과값을 알려주는 역할을 한다. 리눅스 커널에는 멀티미디어 기능을 위해서 사용해

야 하는 각종 주변 기기들에 대한 디바이스 드라이버가 존재하며 그 아랫부분에는 각 디바이스 드라이버에 해당 주변 기기 및 하드웨어 코덱들이 위치한다.

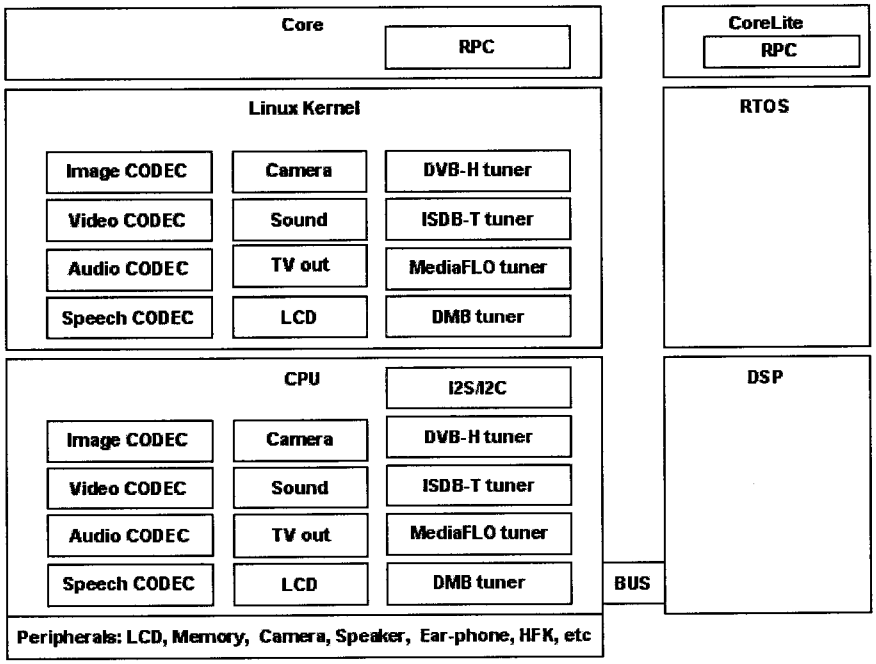

그림 22. 멀티미디어 프레임워크를 위한 Core과 Kernel의 구조

5.5 플레이어를 위한 멀티미디어 프레임워크의 구조 및 동작

그림 23은 멀티미디어 어플리케이션 중 플레이어를 구동했을 때의 멀티미디어 프레임워크의 구조 및 동작에 대해 나타낸 것이다. 각 단계별 간략한 설명은 다음과 같다.

1. 플레이어 어플리케이션에서 OpenMAX AL 혹은 GStreamer core API 를 통해 Playbin 를 생성하고 입력으로 파일 경로나 URI를 보낸다.
2. Playbin에서는 입력에 따라 소스 엘리먼트 플러그인부터 싱크 플러그인까지 자동적으로 파이프라인을 생성한다. 스트리밍을 나타내는 URI인

경우에는 소스 플러그인을 IMS의 RTP등 네트워크 프로토콜을 설정하고 파일 경로인 경우에는 파일 소스를 소스 플러그인으로 설정한다.

3. 소스 플러그인으로부터 받은 데이터를 디코더 플러그인을 이용하여 코덱을 호출하도록 한다.
4. 디코더 플러그인은 OpenMAX IL 컴포넌트 형태의 코덱에게 데이터를 전달한다.
5. 코덱에서 데이터를 디코딩한다. 하드웨어 코덱인 경우와 DSP을 이용한 코덱인 경우를 그림에 나타내었다.
6. 디코딩된 데이터를 코덱으로부터 전달받는다.

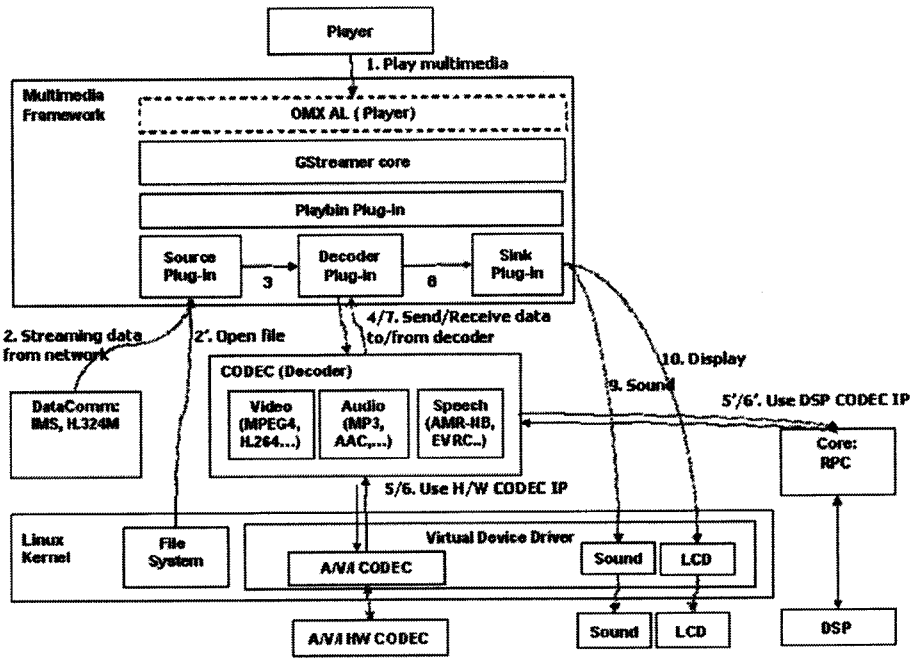

그림 23. 멀티미디어 프레임워크에서의 플레이어 동작

7. 코덱 컴포넌트로부터 디코딩된 데이터를 전달받는다.

8. 싱크 플러그인을 통해 디코딩된 데이터를 출력한다.
9. 사운드는 리눅스 커널의 사운드 디바이스 드라이버를 통해 스피커로 출력한다.
10. 영상인 경우에는 리눅스 커널에 있는 LCD 디바이스 드라이버를 통해 LCD에 출력한다.

다음 그림들은 playbin이 입력으로 들어오는 데이터 타입에 따라 파이프라인을 어떻게 구성하는지 보여주고 있다. 그림 24는 MP3 음악 파일을 실행하기 위한 playbin의 구성이고 그림 25는 Ogg 음악 파일을 실행하기 한 playbin의 구성이다. 둘다 오디오만을 포함한 미디어 파일이지만 데이터 타입에 따라 디멀티플렉서와 디코더가 다르게 선택되어 구성되고 있음을 볼 수 있다. 마지막에 AlsaSink와 OssSink는 사운드 디바이스 드라이버를 선택적으로 변경 가능함을 보여주고 있다.

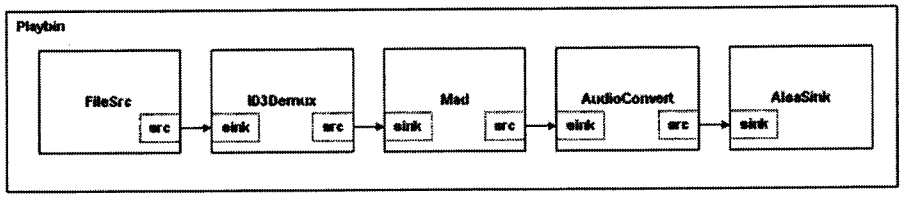

그림 24. MP3 음악 파일 실행을 위한 playbin

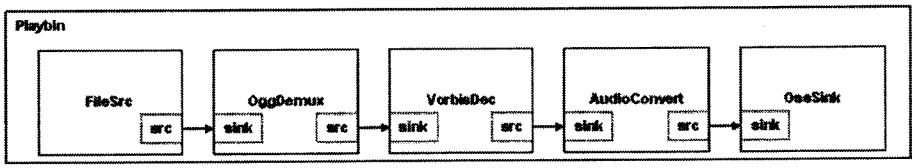

그림 25. Ogg 음악 파일 실행를 위한 playbin

미디어 파일이 오디오와 비디오를 모두 포함한 경우라면 playbin의 구성은

조금 더 복잡해진다. 그림 26은 Xvid 비디오 데이터와 AAC 오디오 데이터를 담고 있는 AVI 미디어 파일 실행을 위한 playbin의 구성을 보여주고 있다.

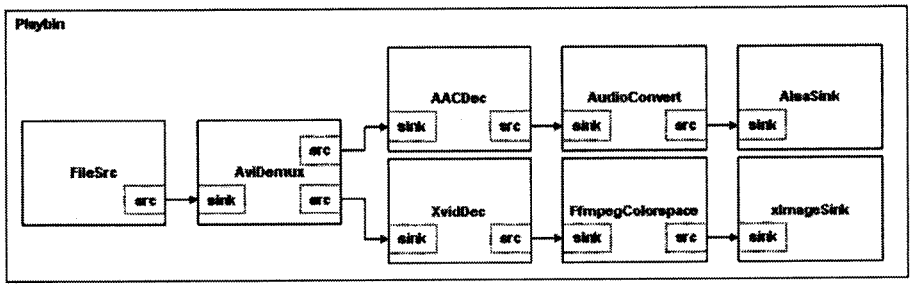

그림 26. AVI 포맷 파일 실행을 위한 playbin

그림 27은 파일 경로가 아닌 URL을 입력으로 받은 경우에 음악 데이터 스트리밍을 위한 playbin의 구성이다. RTSP/RTP와 같은 프로토콜로부터 입력을 받아 구성한다. 그림에서 디코더라고 표현된 부분은 받은 데이터 종류에 따라 달라질 수 있다. 위의 다른 그림에서 보인 대로 디코더외에 AudioConvert등 필요한 플러그인들이 추가로 구성될 수 있다.

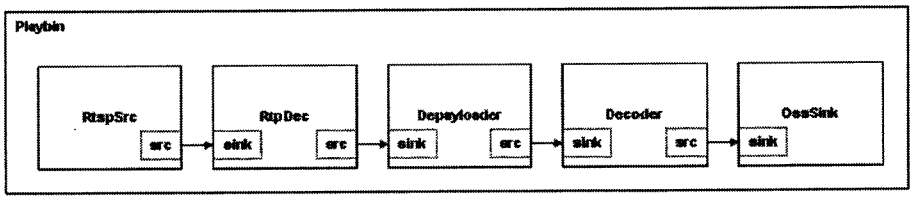

그림 27. 스트리밍을 위한 playbin

그림 28은 플레이어 어플리케이션에서 GStreamer API를 이용하여 playbin을 초기화하는 단계에 대한 시퀀스 다이어그램을 보여주고 있다. 그림에서 어플리케이션은 사용자의 입력을 받거나 출력하는 부분이고 AVMS:AvPlayer 부분은

GStreamer을 이용해 실제적으로 미디어 파일의 실행을 수행하는 부분이다.

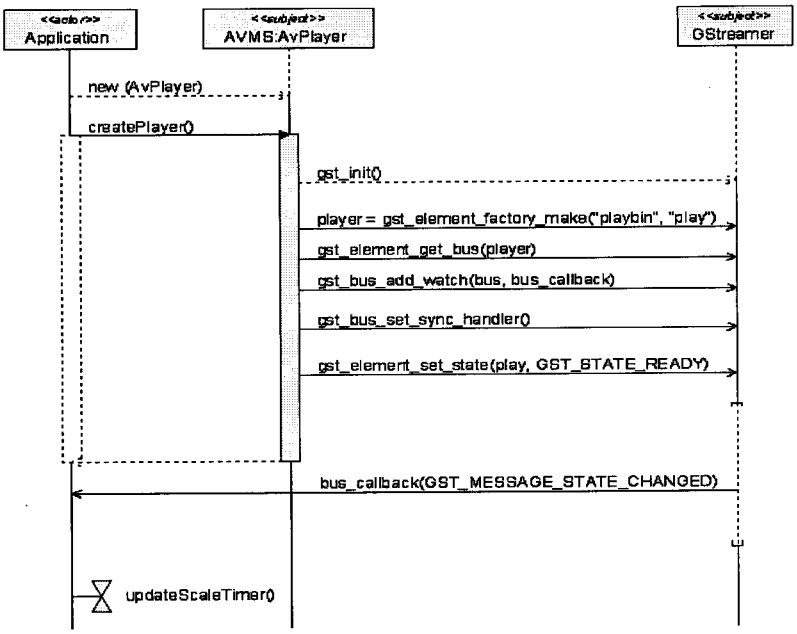

그림 28. 플레이어 시퀀스 다이어그램 - Create

그림 29은 초기화가 끝난후 미디어 파일의 실행을 시작하는 부분이다. "uri" 을 설정하고 gst_element_set_state ()로 상태를 변경하는 것만으로 어플리케이션은 playbin을 구동시켜 음악 파일을 듣거나 동영상을 볼 수 있다.

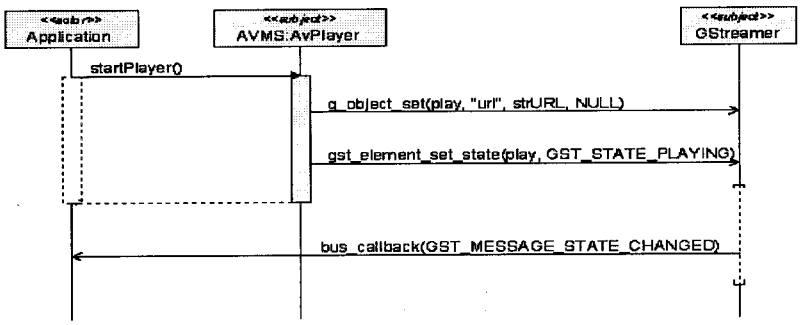

그림 29. 플레이어 시퀀스 다이어그램 - Play

5.6 캠코더를 위한 멀티미디어 프레임워크의 구조와 동작

그림 30은 멀티미디어 어플리케이션 중에 캠코더를 구동했을 때의 멀티미디어 프레임워크 구조 및 동작을 나타낸 것이다.

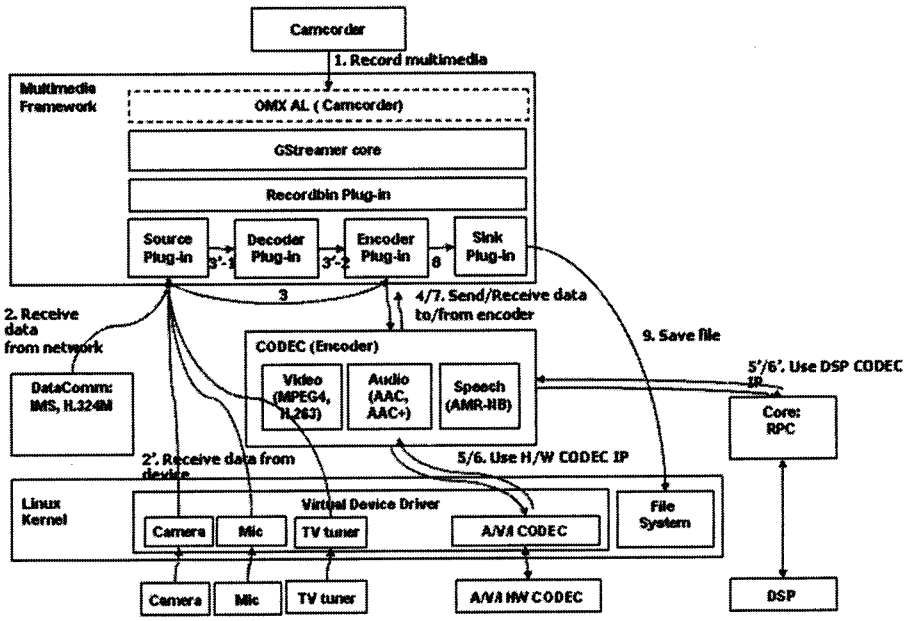

그림 30. 멀티미디어 프레임워크에서의 녹음/녹화 동작

각 단계별 간략한 설명은 다음과 같다.

1. 사용자가 캠코더 어플리케이션를 통해 녹화를 명령하면 캠코더 어플리케이션에서는 OpenMAX AL API나 GStreamer core API를 통해 recordbin을 초기화한다.
2. recordbin에서는 입력으로 지정된 미디어 타입의 종류에 따라 소스 플러그인을 선택한다. 스트리밍으로 전송되는 미디어를 저장하는 경우에는 IMS등의 네트워크 프로토콜로 소스 플러그인으로 선택하고 모바일

기기의 카메라와 마이크로부터 사용자의 입력을 받는 경우에는 카메라와 마이크 디바이스 드라이버를 소스 플러그인으로 선택한다. TV 튜너 디바이스 드라이버를 소스 플러그인으로 선택하는 경우는 사용자가 DMB 등 모바일 TV 방송을 녹화하는 경우이다.

3. 입력받은 소스 중 카메라와 마이크로부터 받은 데이터의 경우에는 바로 인코더 플러그인으로 연결되지만 스트리밍으로 받은 MP3 파일이나 DMB 방송용으로 받은 H.264같은 미디어 타입은 사용자가 원하는 다른 미디어 타입으로 저장될 경우 디코더를 거쳐 인코더 플러그인으로 연결된다. 만약 네트워크으로부터 받은 MP3나 H.264를 그대로 저장하고 싶다면 중간에 디코더와 인코더 플러그인을 생략하고 바로 파일 싱크 플러그인으로 전달하면 된다.
4. 카메라와 마이크로부터 바로 받은 데이터거나 디코딩된 데이터는 GStreamer 플러그인으로부터 사용자가 설정한 미디어 타입대로 OpenMAX IL 컴포넌트로 데이터를 전달한다.
5. 이 경우 코덱은 그 구성에 따라 하드웨어 코덱을 사용하거나 DSP 에서 코덱을 수행할 수도 있다.
6. 인코딩이 끝난 데이터를 코덱으로부터 전달받는다.
7. OpenMAX IL 코덱 컴포넌트는 인코딩된 데이터를 GStreamer 인코더 플러그인에 전달한다.
8. 전달된 오디오와 비디오 데이터는 사용자가 설정한 파일 포맷으로 구성된다.
9. 싱크를 통해 리눅스 커널의 파일 I/O API를 통해 파일로 저장된다.

그림 31은 카메라와 마이크로부터 입력받아 AVI 파일로 저장하는 recordbin의 구조이다.

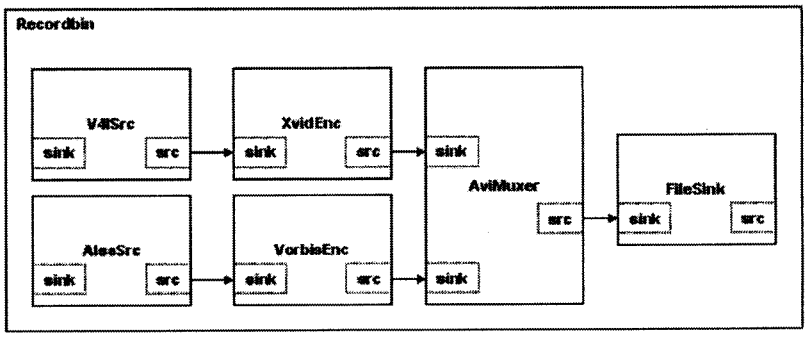

그림 31. AVI 파일 녹화를 위한 recordbin 구조

V4lSrc은 카메라로부터 사용자의 영상을 입력받는 소스 플러그인이고 AlsaSrc은 마이크로부터 사용자의 목소리를 입력받는 소스 플러그인이다. 각각 Xvid 비디오 데이터와 Vorbis 오디오 데이터로 인코딩된 후 AVI 파일 포맷 Muxer로 합성되어 파일로 저장된다. 이 그림은 AVI 파일 저장을 위해 중요 플러그인 위주로 나타낸 것으로 오디오와 비디오 동기를 맞추기 위한 큐 플러그인이나 비디오 레이트나 오디오 볼륨을 조정하기 위한 플러그인이 추가될 수 있다.

플레이어와 마찬가지로 같이 캠코더에서도 간단하게 recordbin을 구동하여 녹화나 녹음을 할 수 있다. 그림 32는 캠코더 어플리케이션에서 recordbin을 초기화하는 시퀀스 다이어그램을 보여주고 있다.

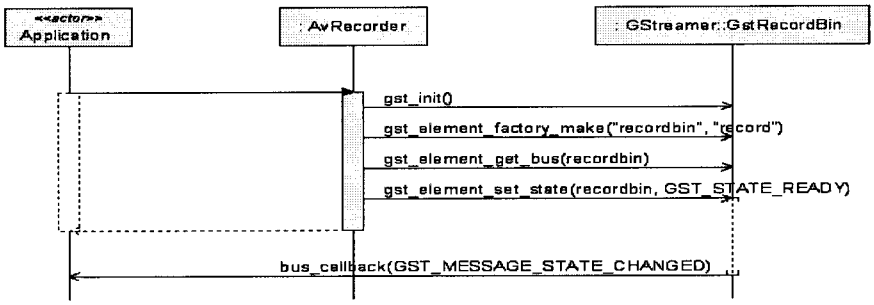

그림 32. 캠코더 시퀀스 다이어그램 - Create

초기화가 끝난 후 캠코더 어플리케이션은 저장할 파일의 이름과 경로를 설정해주고 recordbin의 상태를 PLAYING으로 바꿈으로써 쉽게 녹화/녹음을 수행할 수 있다. 그림 33은 캠코더 어플리케이션에서 녹화/녹음을 시작했을 때 recordbin을 구동시키는 시퀀스 다이어그램을 나타내고 있다.

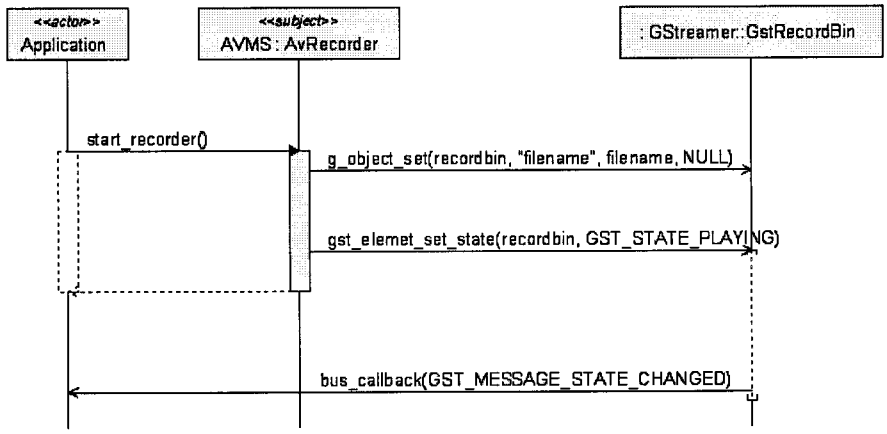

그림 33. 캠코더 시퀀스 다이어그램 – Record

제 6 장 실험 및 결과

6.1 실험 환경 및 방법

타겟 보드로 ARM11 MPCore emulation baseboard을 사용하였다. 타겟 보드에 리눅스 2.6.17 버전[16]을 설치하여 파일 시스템은 네트워크를 통한 리눅스 PC의 파일 시스템을 NFS (Network File System)으로 연결하여 사용하였다. 실제 제품화되는 경우에는 NFS가 아니라 모바일 기기에 내장된 파일 시스템을 사용하게 되므로 성능이 더 나아질 것이라고 예상된다. 컴파일러는 arm-linux-gcc로 4.1.0 버전을 사용하였다.

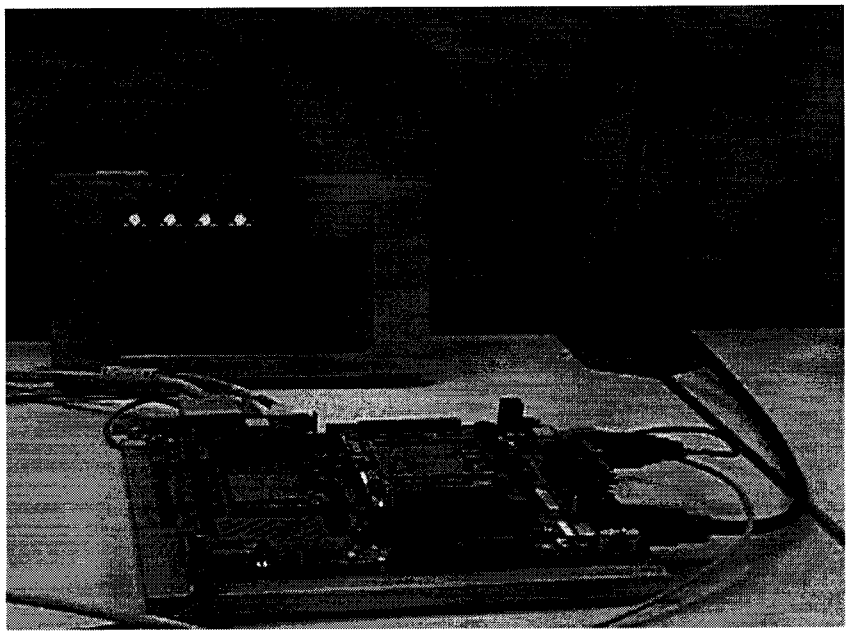

그림 34. 타겟 보드에 리눅스를 구동시킨 모습

멀티미디어 프레임워크를 구성하는데 사용하는 GStreamer의 버전은 다음과 같다[17].

- gstreamer: 0.10.6 version
- gst-plugins-base: 0.10.4 vserion
- gst-plugins-good: 0.10.4
- gst-plugins-ugly: 0.10.4
- gst-plugins-bad: 0.10.3

OpenMAX는 1.0 버전의 API를 적용시켜 구현하였다. OpenMAX IL이 적용된 코덱은 ARM11용 명령어를 이용하여 어셈블리로 최적화하였다.

실험은 간단한 플레이어 어플리케이션을 구현하여 멀티미디어 프레임워크를 검증하였다. 플레이어 어플리케이션은 로컬 파일 실행과 스트리밍을 통한 파일 실행을 그 기능으로 가지고 있다. 실제 모바일 환경에 가깝게 하기 위해서 일반적으로 사용하는 동영상 대신에 실제 광고 동영상들을 사용하였다.

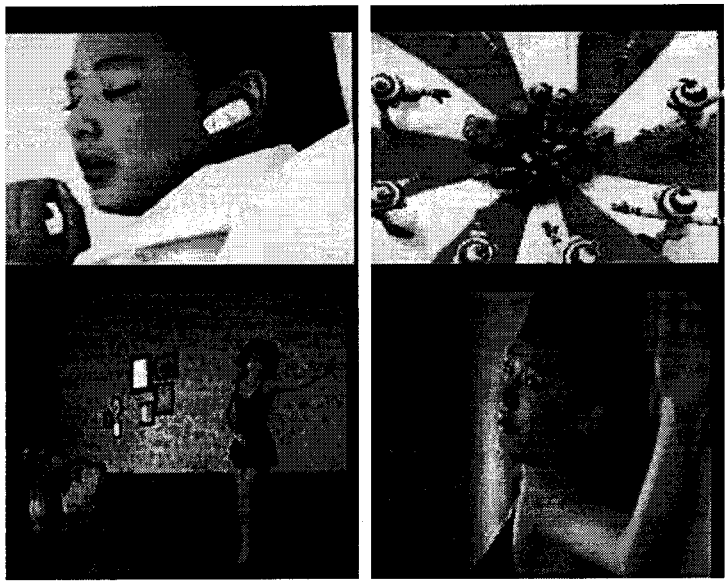

그림 35. 실험에 사용된 동영상들

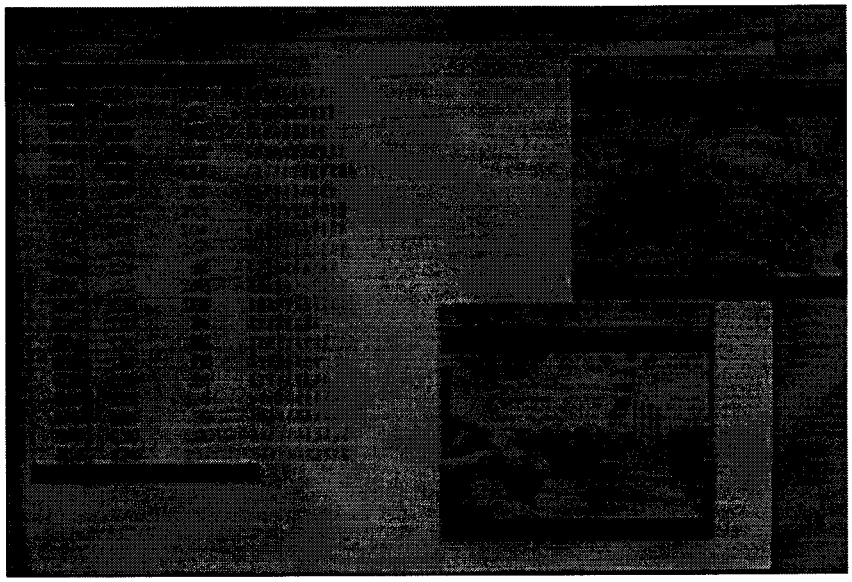

그림 36. 타겟 보드에서 플레이어를 구동시킨 모습

6.2 실험 결과 및 고찰

■ **GStreamer 최적화에 따른 성능 개선**

GStreamer의 빌드 옵션을 수정하여 GStreamer의 초기화 시간을 최적화하였다. 미디어 파일 플레이시에 초기화에 소요 되는 시간은 파일 플레이를 위해 파이프라인을 구성하게 되는 플러그인 파이들(*.so)를 로딩하는 데 소용되는 시간과 GStreamer에서 gst_element_factory_make ("filesrc" , "source")를 호출해서 각 플러그인에 대한 엘리먼트를 생성하는데 소용되는 시간 두 가지로 측정될 수 있다.

실험에 사용한 Xvid 비디오를 포함하고 있는 AVI 파일을 재생하면서 필요한 *.so 파일들은 다음과 같으며 GStreamer에서 생성되는 엘리먼트들은 filesrc, avidemux, xviddec, ffmpegcolorspace, ximagesink이다.

- libgstcoreelements.so
- libgstxvid.so
- libgstffmpegcolorspace.so

- libgstximagesink.so

이에 수정한 빌드 옵션의 내용은 다음과 같다.

--disable-gst-debug --disable-debug

--disable-loadsave --disable-registy

표 4에 각 초기화에 소요되는 시간에 대해서 타겟 보드에서 L2 cache를 on/off했을 때와 각 환경에서 최적화하기 전과 후를 측정한 것을 정리하였다.

표 4. GStreamer 최적화에 따른 초기화 시간 측정

	.so 로딩 시간 (sec)		gst 초기화 시간(sec)	
	L2 cache off	L2 cache on	L2 cache off	L2 cache on
최적화 전	8.127	1.817	4.591	1.372
최적화 후	3.568	0.515	3.987	1.382
차이	-56.10%	-72.45%	-13.16%	0.72%

*.so 파일들을 로딩하는데 소요된 시간에 대해서 살펴보자. 파일 읽기/쓰기에 대한 성능을 올릴 수 있는 L2 cache을 off에서 on으로 변경함으로써 파일을 로딩하는 시간을 획기적으로 줄일 수 있음을 볼 수 있다. 그러나 이러한 타겟 환경 설정의 변경 외에도 GStreamer의 빌드 옵션 최적화를 통해 얻을 수 있는 효과 또한 큼을 알 수 있다. 표에서 보듯이 최적화 이후 L2 cache off의 경우에서는 56.10%와 L2 cache on의 경우에서는 72.45%가 감소되는 효과를 보여 L2 cache와 별개로 최적화가 성공적으로 이루어졌음을 알 수 있다. 감소율뿐 아니라 실제 소요 시간도 0.5 sec 정도로 작은 수치로 나타났다.

반면 gst 초기화 시간은 그에 비해 최적화의 효과가 적은 것으로 나타났다. L2 cache를 on으로 설정한 경우도 *.so 파일 로딩 시간만큼 효과가 적고 빌드

옵션에 따른 최적화에서도 그다지 좋은 효과를 나타내고 있지 않다. 이는 gst 초기화 시간은 사용할 플러그인 엘리먼트에 대한 인스턴스를 생성할 때 메모리로 할당하고 초기화하는 데 걸리는 시간이어서 수정한 빌드 옵션과 관계가 적기 때문으로 나타난다. 그러나 *.so 로딩 시간보다 효과가 적더라도 실제 소요된 시간에서는 처음보다 상당 부분 소요 시간이 줄어들었음을 알 수 있다.

이러한 초기화에 소요되는 시간외에 타겟 보드에 탑재되는 컴파일된 이미지의 크기를 측정하였다. 최적화 측정에 사용되어 컴파일된 파일들은 실험에서 사용한 플레이어 어플리케이션을 구동하는 데 필요한 플러그인들로만 구성하였다.

표 5. GStreamer 최적화에 따른 코드 크기 측정

	Code size (MByte)	
	unstripped	stripped
최적화 전	3.849	2.960
최적화 후	1.872	1.722
차이	-51.34%	-41.83%

플레이어에 필요한 파일들을 처음으로 빌드한 코드 크기는 3.849MB였고 이를 디버깅등에 사용할 심볼 데이터등을 제거하고 빌드 옵션 최적화한 크기는 1.722MB로 66.27%정도의 감소율을 보인다. Unstripped된 상태에서도 빌드 옵션 최적화만을 통해서도 51.34%의 감소율을 보이고 있어 최적화 방법이 효과적임을 나타내고 있다.

■ 싱크 플러그인 변경에 따른 파이프라인 재구성의 성능 측정

GStreamer의 가장 큰 장점중의 하나는 소스 변경없이 환경에 맞는 엘리먼트를 선택/구성이 가능하다는 것이다. 아래 실험은 다양한 미디어 파일 포맷에 따

른 코덱의 구성은 물론 그 외 싱크 플러그인의 종류을 다르게 하여 타겟 환경에 맞게 다양한 구성을 이룰 수 있음을 보여주는 실험이다.

XImageSink는 xlib에서 xdifectFB을 사용하여 LCD 프레임 버퍼에 출력하는 싱크 플러인이며 DfbSink는 directFB을 사용하여 LCD 프레임 버퍼에 출력하는 싱크 플러그인이다. 두 종류가 가장 많이 사용하는 LCD 출력을 담당하는 모듈이다. FbSink는 이런 모듈을 사용하지 않고 바로 LCD 프레임 버퍼에 출력하는 싱크 플러그인으로 프레임버퍼에 직접 그리게 되므로 유연성은 떨어지지만 성능 문제로 인해 모바일 기기에서 많이 채택되는 출력 모듈이다.

동영상은 Xvid 비디오 스트림을 갖는 AVI 파일을 사용하였으며 QCIF 크기로 프레임을 달리하여 각 싱크 플러그인으로 수행했을 때의 성능을 측정하였다.

표 6. 싱크 플러그인 타입에 따른 성능 측정

실험 영상	싱크 플러그인 타입 (fps)		
	xImageSink	DfbSink	FbSink
QCIF, 15 fps	14	14	14
QCIF, 23 fps	21	22	23
QCIF, 30 fps	19	21	23

표에서 볼 수 있듯이 가장 성능이 좋은 것은 프레임버퍼에 직접 그리는 FbSink가 가장 우수함을 알 수 있으며 실험에 사용한 타겟 보드에서의 비디오의 최대 성능은 23 fps 임을 알 수 있다. 보통 영화가 24 fps이므로 모바일 기기에서 영화 감상하는 데는 적합한 성능임을 알 수 있다.

■ **OpenMAX IL 적용에 따른 성능 측정**

본 논문에 제안한 멀티미디어의 가장 큰 특징중의 하나는 GStreamer구조에 OpenMAX IL를 통합했다는 것이다. 두 구조가 비슷하여 통합이 어렵지 않았다하

더라도 성능을 중요시되는 모바일 기기에서는 성능 저하가 없는지 검증하는 것이 필수적인 작업이다. 여기에서는 OpenMAX IL 컴포넌트로 구현된 MPEG4 AAC-LC의 디코더 코덱과 인코더 코덱을 실험 대상으로 하였다.

다음은 MPEG4 AAC-LC 디코더에 사용한 다양한 오디오 스트림들이다. 속성을 각각 다르게 한 상태에서 프레임 크기를 달리한 스트림 그룹을 둘로 나누어서 속성과 프레임 크기에 따른 OpenMAX IL 의 성능을 측정한 것을 표 7에 나타내었다.

- Audio1: 128 kbps, 48 kHz, 2 channel, MS/ADTS, 402 frames
- Audio3: 128 kbps, 48 kHz, 2 channel, MS/ADTS, 1313 frames
- Audio3: 128 kbps, 48 kHz, 2 channel, MS/ADTS, 1977 frames

- Audio4: 128 kbps, 48 kHz, 2 channel, MS/ADIF, 510 frames
- Audio5: 128 kbps, 48 kHz, 2 channel, MS/ADIF, 831 frames
- Audio6: 128 kbps, 48 kHz, 2 channel, MS/ADIF, 1788 frames

표 7. AAC 디코더의 OpenMAX IL 적용에 따른 성능 측정

테스트 스트림	소요 시간 (sec)			
	전체 소요 시간	코덱 소요 시간	코덱 + OpenMAX IL] 소요 시간	OpenMAX IL 소요 시간
Audio1	8.576	0.724	1.888	1.164
Audio2	28.010	2.454	3.889	1.435
Audio3	42.176	3.475	5.040	1.565
Audio4	10.858	0.906	2.127	1.221
Audio5	17.728	1.533	2.836	1.303

Audio6	38.144	3.127	4.703	1.576

표에서 나타나듯이 오디오 종류에는 상관없이 OpenMAX IL 은 거의 일정한 성능을 나타낸다. 프레임 크기에 비례하여 소요 시간이 증가하는 양상을 보이지만 프레임 크기가 두 배로 늘어날 때 약 0.1~0.2 sec 정도의 증가율만 보이므로 그 차이가 매우 미미하다.

다음은 MPEG4 AAC-LC 인코더에 대해 실험한 결과이다. 테스트 스트림은 다음과 같이 종류와 프레임별로 세 분류로 나누어서 비교하였으며 그 측정 결과를 표 8에 나타내었다.

- Audio1: 64 kbps, 44.1 kHz, 1 channel, 1725 frames
- Audio2: 64 kbps, 44.1 kHz, 2 channel, 1165 frames

- Audio3: 64 kbps, 44.1 kHz, 2 channel, 111 frames
- Audio4: 64 kbps, 48 kHz, 2 channel, 174 frames

- Audio5: 64 kbps, 44.1 kHz, 2 channel, 288 frames
- Audio7: 64 kbps, 44.1 kHz, 2 channel, 1208 frames
- Audio8: 64 kbps, 44.1 kHz, 2 channel, 2285 frames

표 8. AAC 인코더의 OpenMAX IL 적용에 따른 성능 측정

테스트 스트림	소요 시간(sec)			
	전체 소요 시간	코덱 소요 시간	코덱 + OpenMAX IL 소요 시간	OpenMAX IL 소요 시간
Audio1	40.000	3.469	4.666	1.197

Audio2	27.000	4.218	5.440	1.222
Audio3	2.530	0.516	1.587	1.071
Audio4	3.688	0.819	1.862	1.043
Audio5	6.634	1.261	2.369	1.108
Audio6	28.000	4.316	5.473	1.157
Audio7	53.000	9.260	10.710	1.450

표에서 볼 수 있듯이 다른 코덱을 사용했지만 거의 동일한 성능을 보이고 있어 안정적인 상태임을 알 수 있다. AAC 인코더에서도 테스트 스트림 종류별에 따른 OpenMAX IL 의 성능 변동은 보이지 않으며 프레임 크기에 따른 소요시간의 증가도 프레임 길이가 두 배이상으로 증가할 때 0.1~0.2 sec 정도로 미미한 수준임을 알 수 있다.

제 7 장 결 론

리눅스 기반의 모바일 소프트웨어 플랫폼에서 GStreamer기반의 멀티미디어 프레임워크를 설계 구현하였다. GStreamer는 기본적인 기능을 담당하는 GStreamer core와 그 외 멀티미디어 어플리케이션에 필요한 기능들을 플러그인으로 제공함으로써 다양한 모바일 기기에 맞는 구성을 코드 수정없이 쉽게 이룰 수 있다. 본 논문의 멀티미디어 프레임워크에서는 GStreamer의 빈 구조를 도입하여 어플리케이션이 멀티미디어 각 기능에 따른 파이프라인을 손쉽게 구성할 수 있도록 하였으며 실험에서도 다양한 플러그인의 변경이 가능함을 보였다. 또한 GStreamer 자체의 최적화를 통해 초기화 시간를 최대 72.45%, 코드 크기를 66.27% 정도 감소시켰다.

본 논문의 멀티미디어 프레임워크의 다른 특징 중의 하나는 OpenMAX IL의 도입이다. 하드웨어 환경에 민감함 코덱들을 OpenMAX IL 컴포넌트로 구성하여 멀티미디어 프레임워크에서는 하드웨어의 사양과 상관없이 일관성 있는 코덱 API들을 사용할 수 있다. OpenMAX IL 컴포넌트를 GStreamer 플러그인으로 통합함으로써 두 구조의 장점을 모두 취하였으며 실험 결과 통합에 따른 소요 시간의 증가는 코덱 종류나 데이터 종류에 관계없이 1 sec 정도의 적은 수준으로 일정하게 나타났으며 프레임 길이 증가에 대해서는 0.1~0.2 sec정도의 무시할 만한 소요 시간이 측정되었다.

참 고 문 헌

[1] Bill Weinberg, "The State of Mobile Linux," *SDForum*, October, 2006.

[2] John Cook, "Introduction to the ACCESS Linux Platform," *SDForum*, October, 2006.

[3] Benoit Schillings, "Mobile and Embedded Linux; How Linux and Open Source Disrupt the Mobile and Embedded Industries," *SDForum*, October, 2006.

[4] GStreamer Open Source Multimedia Framework, [online] http://gstreamer.freedesktop.org.

[5] Wim Taymans, Steve Baker, Andy Wingo, Ronald S. Bultje, Stefan Kost, GStreamer Application Development Manual (0.10.6.1), [online] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/manual/manual.pdf.

[6] OpenMAX - The Standard for Media Library Portability, [online] http://www.kronos.org/openmax/.

[7] Brian Murray, "OpenMAX StreamingMedia Protabiltiy," *SIGGRAPH2006*, July, 2006.

[8] OpenMAX Integration Layer Application Programming Interface Specification Version 1.0, [online] http://www.khronos.org/openmax/spec/.

[9] OpenMAX Development Layer Application Programming Interface Specification Version 1.0, [online] http://www.khronos.org/openmax/spec/.

[10] OpenMAX - The Standard for Media Library Portability, [online] http://www.arm.com/products/esd/openmax_home.html.

[11] Neon Technology Data Sheet, [online] http://www.arm.com/products/CPUs/NEON.html.

[12] D.Melpignao, P.Sen, "Using OpenMAX Integration Layer with GStreamer," [online] http://www.khronos.org/openmax/whitepapers/OpenMAX_IL_with_GSstreamer.

pdf.

[13] Javier Orensanz, "RealView Hardware Platforms Product Selector," *ARM Whitepaper*, July, 2006.

[14] How fast does the MPCore + EB platform run? , [online]http://www.arm.com/support/faqdev/14537.html.

[15] ARM11 MPCore, [online] http://www.arm.com/products/CPUs/ARM11MPCoreMultiprocessor.html.

[16] Linux Operating System Download, [online] http://www.arm.com/linux/linux_download.html.

[17] Gsteamer Source Download, [online] http://gstreamer.freedetop.org/src/.

ABSTRACT

According as the requirements regarding recent mobile device technologies are grown rapidly, high-level features and high performances are required. As a result, users are looking forward to seeing the same operations which can be processed on the personal computer on mobile devices in a few years. Mobile platform is the framework regarding hardware and softeware which are embedded on mobile device and process various applications. As aspect of hardware, an application processor chipset for making mulitmedia application run and a modem chipset for processing connection with wireless network are key technologies. As aspect of software, key technology is the mobile software platform which includes kernel, middleware and application framework. These three key technologies should be interoperated and implemented completely for designing good architecture of mobile device.

In this paper, it is proposed to design GStreamer based multimedia framework on mobile software platform based on linux. GStreamer provides GStreamer core for basic functionality and various plugins for multimedia applications. The proposed multimedia framework adopts the architecture of GStreamer and makes it possible that applications can construct pipeline easily according to multimedia feature.

In experiments, it shows that proposed multimedia framework can change various plugins easily for any request. And the results show that initialization time is reduced by maximum 72.45% and code size is reduced by 66.27% through optimization of GStreamer itself.

The other characteristic of the proposed multimedia framework is an adoption of OpenMAX API. Hardware dependant codecs are constructed with OpenMAX IL component and multimedia framework can use codecs consistently through consistent OpenMAX IL APIs regardless of some change of hardware specification. Proposed multimedia framework can have both advantages of two solutions as intergration of OpenMAX IL component with GStreamer plugins. Experimental results show that it has enough small overhead as the result by integration and it works effectively.