특집

데이터방송의 상호 운영성을 고려한 GEM 기반 어플리케이션 관리 방안

차경호, 석주명, 이한규, 홍진우(한국전자통신연구원)

요 약

본 원고는 지역적으로 상이한 데이터방송 표준을 어플리케이션의 상호 운용성(interoperability) 보장이라는 관점에서 일련의 공통된 프레임워크(framework)를 제시하고 있는 GEM(Globally Executable MHP)을 기반으로 한 어플리케이션의 관리 방안을 소개한다. 특히, 데이터방송 컨텐츠의 상호 운용성을 고려하여 어플리케이션의 운영과 관련된 GEM의 어플리케이션 라이프사이클(application lifecycle) 부문에 초점을 맞춰 어플리케이션의 수명주기 즉, 어플리케이션의 생성 및 로딩(loading)에서부터 시작하여 종료(destroy)에 이르기까지의 어플리케이션의 전체 수명주기를 공통적으로 관리할 수 있기 위한 어플리케이션 라이프사이클의 프레임워크 및 시그널링(signalling) 프로세스를 제안한다. 본 원고에서 제안하고 있는 어플리케이션 관리 방안은 각 방송플랫폼간의 조화를 고려하고 있는 상황에서 해당 플랫폼에서의 어플리케이션 운용과 관련된 모듈을 구성하는데 공통적으로 적용될 수 있을 것으로 기대된다. 또한, 각 방송플랫폼 간의 어플리케이션의 상호 운용성을 고려함에 있어 어플리케이션 관리 측면에서의 접근 방법으로 제안될 수 있을 것으로 기대된다.

I. 서 론

디지털방송은 다채널화, 고품질화, 다기능화로 요약할 수 있다. 우리는 이미 국내의 디지털 위성방송 및 케이블 방송을 통하여 다채널방송을 경험하고 있으며, 지상파 디지털 TV 방송에서 실시되고 있는 고선명 TV 방송을 통하여 고품질 TV 방송을 접하고 있다. 이와 같이 다채널화 및 고품질화와 함께 다기능성도 제공되고 있다. 다기능성이란 멀티미디어 정보들을 방송매체를 통하여 제공하는 서비스 유형으로 방송의 다양화, 개인화, 양방향화, 네트워크화를 의미한다. 디지털방송에서 다기능 서비스를 제공하는 것을 데이터방송 또는 대화형방송이라고 하는데, 데이터방송은 '무엇을 제공하는가'라는 내용적 측면에서 사용되는 용어라 할 수 있는 반면, 대화형방송은 '어떻게 서비스 되는가'라는 기능적 측면에서의 용어이나, 데이터 방송은 당연히 대화형으로 제공되어야 하고, 대화형방송은 멀

티미디어 데이터를 방송해야 하기 때문에 혼재되어 사용된다[7].

데이터방송은 텍스트, 정지화, 그래픽, 문서, 소프트웨어 등의 멀티미디어 데이터를 방송매체를 이용하여 전송하고, 전용 셋탑박스 혹은 해당 처리기능을 보유한 PC를 통하여 시청자가 그 정보를 이용하게 하는 서비스이다. 이는 방송과 통신의 융합이라는 시대적 흐름에 가장 잘 부합되는 서비스로서, 사용자와의 상호작용과 방송정보의 개인화를 가능하게 하는 것으로 지금까지 체험해 오던 방송 서비스의 속성을 획기적으로 바꾸게 할 새로운 서비스이다[7].

현재 국내의 방송플랫폼에 따른 데이터방송 표준의 채택 상황을 살펴보면, 위성방송의 경우는 MHP(Multimedia Home Platform), 지상파 방송은 ACAP(Advanced Common Application Platform), 그리고 케이블 방송은 OCAP(Open Cable Application Platform)을 각각 수용하고 있다[6]. 이와 같이 상이한 데이터방송 표준의 채택은 각 방송플랫폼 간의 상호 조화(harmonization)를 고려하고 있는 상황에서 다양한 문제점들을 야기시키고 있다. 특히, 지상파 및 케이블 방송에서 각 데이터방송 규격 간의 상호 운용성(interoperability)에 대한 문제를 놓고 많은 논란이 대두되고 있는 실정인데, 실제로 이슈(issue)화되고 있는 논란들에는 지상파 방송의 재전송에 따른 재변조(8VSB-QAM) 방식의 논란이라든지, 혹은 보편적 무료서비스의 유료방송으로의 적용문제, 각 방송플랫폼간 어플리케이션의 상호 운용성 문제 등이 있다. 이러한 조화의 문제들은 기술적 혹은 법/제도적인 측면의 협의 및 접근을 통해 그 해결방안이 강구될 것으로 전망이 되고 있는데, 본 원고에서는 이와 같이 상이한 데이터방송 표준의 채택으로 인해 야기되고 있는 조화의 문제와 관련하여 그 근본 해법으로 고려될 수 있는 GEM(Globally Executable MHP)에 초점을 맞추고자 한다.

GEM은 유럽지역의 디지털 방송관련 단체들(JTC, EBU, CENELEC, ETSI)을 중심으로 제정된 표준으로서, ACAP, OCAP, ARIB(Association of Radio Industries and Business) 등과 같은 지역적으로 상이한 데이터방송 표준을 어플리케이션의 상호 운용성 보장이라는 관점에서 조화시키기 위한 공통의 프레임워크이다[2]. 초기에 GEM은 DVB(Digital Video Broadcasting)에서 MHP의 특정 버전에 참조하고 있는 다양한 표준들의 참조 문제점들을 해결하기 위해 개발되었고, 이후 OCAP과의 상호교류를 통해 DVB에 한정되어 있는 부분들을 제거함으로써, 현재는 특정 시장에 제한되어 있는 데이터방송 표준들이 GEM을 기반으로 해당 표준을 확장할 수 있도록 하고 있다.

GEM은 ACAP, OCAP, ARIB 등과 같은 데이터방송 표준에서 공통으로 기초되고 있는데[3][4], 본 원고에서는 이와 같은 GEM을 공통적으로 기초하고 있는 각종 미들웨어 표준들에서 데이터방송 컨텐츠(contents)가 원활하게 운용되기 위한 방안에 대해 초점을 맞추고자 한다. 특히, 데이터방송 컨텐츠의 상호 운용성을 고려하여 어플리케이션의 운영과 관련된 GEM의 어플리케이션 라이프사이클(application lifecycle) 부문을 중점적으로 취급하고자 하는데, 어플리케이션 라이프사이클은 데이터방송 어플리케이션의 관리와 운영을 담당하는 미들웨어의 구성모듈(module)로서, 어플리케이션의 수명주기 즉, 어플리케이션의 생성 및 로딩(loading)에서부터 시작하여 종료(destroy)에 이르기까지의 어플리케이션의 전체 수명주기를 관리하는 것을 목적으

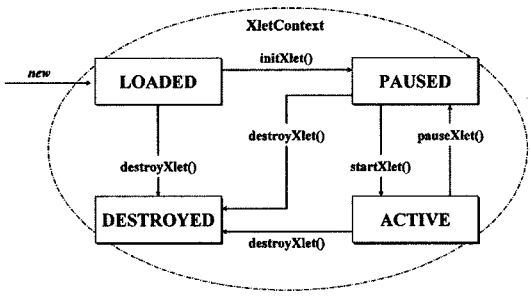
로 하고 있다. 따라서 본 원고에서는 이와 같은 어플리케이션 라이프사이클의 목적을 충족시키기 위해서는 어떠한 조건들이 존재하고, 그러한 조건들이 만족되기 위해서는 어떠한 메커니즘들로 어플리케이션 라이프사이클이 구성되어야 하는가에 대해서 논의하고자 한다.

본 원고의 구성은 크게 세 부분으로 나누어져 있다. 2절에서는 어플리케이션 라이프사이클 부문의 주요 인자(element)들 즉, Xlet, XletContext, 그리고 XletManager에 대한 요구사항에 대해 기술하고 이들간의 상호 관계를 정립한다. 3절에서는 그러한 요구사항을 바탕으로 실제 어플리케이션 라이프사이클이 구성될 수 있는 구조와 중심 프로세스에 대해 기술한다. 마지막으로 결론에서는 본 원고의 요약과 향후 연구방향에 대해 기술한다.

II. 어플리케이션 라이프사이클의 주요 인자

1. Xlet

일반적으로 어플리케이션 라이프사이클의 관리대상에는 DVB-MHP1.0에서 정의하는 바에 따라 DVB-J 어플리케이션과 DVB-HTML 어플리케이션으로 나누어 진다[1]. 이들은 모두 GEM 어플리케이션의 범주에 속하는데, GEM의 어플리케이션 모델(model)은 ACAP과 OCAP에서 그대로 차용하고 있다. 따라서, 이에 대해 ACAP에서는 각각 ACAP-J와 ACAP-X로 나누어지고, OCAP에서는 각각 OCAP-J와 OCAP-HTML로 나누어 진다[3][4]. 이와 같은 GEM 어플리케이션 모델 가운데 DVB-HTML에 해당 되는 어플리케이션에는 XHTML이나,



〈그림 1〉 Xlet의 상태 전이 모델

ECMAScript 등이 있고, DVB-J에 해당되는 어플리케이션에는 Xlet이 있다[1][2]. 본 원고에서는 어플리케이션 라이프사이클의 관리 대상이 되는 어플리케이션들 가운데 ACAP, OCAP, 그리고 MHP에서 공통으로 정의되어 있는 Xlet 어플리케이션에 초점을 맞추고자 한다.

일반적으로 Xlet의 개념은 Xlet 어플리케이션 라이프사이클을 이용하는 자바 어플리케이션을 Xlet 이라고 한다. 또한, API(application programming interface) 측면에서의 정의는 JavaTV에 정의되어 있는 Xlet 인터페이스(interface)를 구현하고 있는 어플리케이션을 바로 Xlet 어플리케이션이라고 한다[5]. Xlet은 일련의 라이프사이클을 따르는데, <그림 1>은 Xlet이 운용되는 라이프사이클 모델을 도식적으로 표현한 것이다[5]. Xlet 라이프사이클 모델은 Xlet의 구동이 시청자가 예상하는 형태의 동작에 가깝도록 일정한 규칙을 따르고 있다. 즉, Xlet의 시작(start)과 중지(pause) 그리고 종료에 대한 프로세스를 시청자 알 수 있도록 명시적으로 정의하고 있다. Xlet 라이프사이클 모델을 프로세스 측면에서 설명하면, 최초 Xlet이 생성되어서 로딩이 이루지고 나면 LOADED 상태가 되고, LOADED 상태에서 initXlet( ) 메서드가 호출되면 PAUSED 상태가 된다. 그리고 이 상태에

```
import javax.tv.xlet;

public class XletMain implements Xlet {

    public XletContext context;
    public HScene hscene;
    public App app;

    public void initXlet(XletContext ctx)
        throws XletStateChangeException {
        context = ctx;
        hscene = HSceneFactory.getInstance().
            getDefaultHScene();
        hscene.setLayout(new BorderLayout());
        app = new App();
        hscene.add(app);
        ...
    }

    public void startXlet()
        throws XletStateChangeException {
        hscene.setVisible(true);
        hscene.addKeyListener(app);
        hscene.requestFocus();
        app.setVisible(true);
        ...
    }

    public void pauseXlet() {
        hscene.removeKeyListener(app);
        app.setVisible(false);
        ...
    }

    public void destroyXlet(boolean uncondition)
        throws XletStateChangeException {
        app.setVisible(false);
        hscene.setVisible(false);
        hscene.removeKeyListener(app);
        hscene.removeAll();
        app.removeAll();
        if(app != null) {
            app.destroy();
            app = null;
        }
        hscene.setVisible(false);
        ...
        System.gc();
    }

    public XletMain() {
    }
}
```

〈그림 2〉 Xlet 코딩 예제

서 startXlet( ) 메서드가 호출되면 GUI(graphic user interface)로 구성된 Xlet인 경우 최종 TV 화면에 보여지게 되는 ACTIVE 상태가 된다. 또한, ACTIVE 상태에서 pauseXlet( ) 메서드가 호출이 되면 PAUSED 상태가 되는데, 각 상태에서 즉, LOADED, PAUSED, 그리고 ACTIVE 상태에서 destroyXlet( ) 메서드가 호출이 되면 Xlet은 DESTROYED 상태가 되어 최종 메모리에서 제거되는 그러한 상태 변경이 이루어지게 된다. 이러한 상태 전이 모델은 어플리케이션 라이프사이클 부문에 대해 Xlet을 관리하는 방법과 규칙을 제시하는 기본적인 모델이 된다.

Xlet이 이와 같은 상태 전이 모델에 맞게 어플리케이션 라이프사이클 부문에 의해서 관리되도록 하기 위해서는 다양한 조건들이 충족되어야 한다. 기본적으로 Xlet은 JDK1.1.8에 준하는 Personal Profile에 정의되어 있는 클래스(class)들을 사용해서 작성이 되어야 한다. 즉, GEM은 MHP1.0을 기초로 하고 있고, MHP1.0은 Personal JAVA를 기본으로 하고 있다. 따라서, Xlet은 Personal JAVA에 정의되어 있는 API를 이용해서 작성되어야 한다는 것이다. 또한, Xlet은 해당 인터페이스 내에 정의되어 있는 메서드에 대해 어플리케이션 라이프사이클 부문에서의 코딩 요구사항이 존재할 수 있다. 다시 말해서, Xlet 인터페이스는 4개의 라이프사이클 메서드 즉, initXlet( ), startXlet( ), pauseXlet( ), 그리고 destroyXlet( ) 메서드로 구성되어 있는데, 이들 메서드들은 해당 메서드의 용도에 맞게 코딩이 이루어져야 한다는 것이다. 참고로, 이와 같은 사항들은 실제 프로그램 제작자 혹은 콘텐츠 제작자에 의해서 Xlet이 코딩(coding)될 때 요구되는 코드 요구사항이 될 수 있는데, 이에 대해 보다 구체적으로 정리하면 다음과 같다.

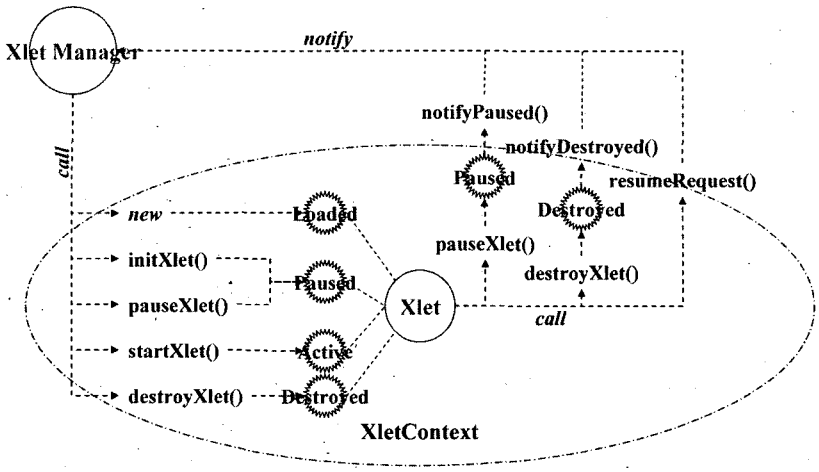
먼저, 생성자(constructor) 부문에 대한 요구사항이다. 기본적으로 Xlet은 javax.tv.xlet 패키지에 정의되어 있는 Xlet 인터페이스를 구현해야 하고, 생성자에는 아규먼트(argument)가 없으며 수정자(modifier)는 'public'으로 설정되어야 한다. 그리고 initXlet( ) 메서드 부분에서는 GUI

로 구성된 Xlet인 경우 각종 컴포넌트들을 초기화 하는 코드, 즉 Xlet에서 이용하는 각종 리소스에 대한 초기화를 수행하는 모듈이 이 메서드에 삽입되어야 한다. 참고로, 이 메서드의 파라미터로 XletContext 객체가 입력되는데, XletContext는 보통 Xlet과 XletManager 간의 통신을 수행하는 인터페이스 객체로써, Xlet에게 각종 런타임(runtime) 정보라든지, 루트 컨테이너(root container) 정보를 제공하는 역할을 한다. 다음은 startXlet( ) 메서드인데, 이 부분에서는 GUI로 구성된 Xlet인 경우, Xlet의 가시성(visibility)에 대해 TRUE가 되는 내용 즉, Xlet 어플리케이션이 화면에 보여지게 되는 내용이 포함되어야 한다. 또한, pauseXlet( ) 메서드의 경우, startXlet( )과 반대로 Xlet의 가시성에 대해 FALSE가 되는 내용이 포함되어 해당 UI 어플리케이션을 보이지 않게 하는 내용이 포함되어야 한다. 마지막으로 destroyXlet( ) 메서드인데, 이 메서드는 Xlet을 메모리에서 완전히 제거하는 내용이 포함될 수 있다. 예를 들면, 어플리케이션 내부에서 사용된 각종 객체들에 대해 NULL처리를 하거나, I/O 혹은 네트웍 연결 등을 종료시키는 등 어플리케이션 내부에서 사용했던 각종 리소스들을 제거하는 내용이 포함될 수가 있고, 최종 VM(virtual machine)의 가비지 컬렉션(garbage collection)이 수행되는 내용이 포함될 수가 있다. 이와 같은 Xlet의 코딩 요구사항들을 바탕으로 실제 코딩이 이루지는 경우 <그림 1>과 같이 구성될 수 있다.

2. XletContext

본 절에서는 XletContext에 대해 기술한다. XletContext라는 것은 Xlet을 위한 환경을 제공하는 인터페이스로써, Xlet과 해당 Xlet을 관리하는 XletManager 간의 통신을 수행하는데 주로 이용이 된다[5]. 일반적으로 Xlet의 상태 변경은 XletManager에 의해서 수행이 된다. 그러나, 간혹 Xlet 자체에서도 해당 Xlet의 상태 변경이 이루어지는 경우가 있는데, 이 때에는 반드시 Xlet이 자기자신의 상태변경을 XletManager에게 알려야 한다는 요구사항이 존재한다. 왜냐하면, Xlet 자체에서의 상태 변경에 대해 XletManager가 인지할 수 없기 때문에 해당 Xlet을 적절하게 관리할 수 없게 된다. 따라서 Xlet은 자기자신의 상태 변경을 XletManager에게 알려야 되는데, 이 때 이용되는 것이 바로 XletContext 인터페이스가 된다. 정리해 보면, Xlet은 XletContext 인터페이스 내에 정의되어 있는 메서드를 이용하여 자신의 상태를 PAUSED 혹은 DESTORY 상태로 변경할 수 있고, 또한 XletManager에게 Xlet 자신의 상태를 ACTIVE 상태가 되도록 요청할 수도 있다. 이러한 XletContext 인터페이스 내부에 정의되어 있는 메서드에는 크게 3개의 메서드 즉, notifyDestroyed(), notifyPaused(), 그리고 resumeRequest() 메서드가 정의되어 있는데, 각 메서드에 대한 용도에 대해 설명하면 다음과 같다.

먼저, notifyDestroyed( ) 메서드는 Xlet 자체에서 스스로를 종료시킨 후 XletManager에게 DESTROYED 상태를 알리는데 이용되는 메서드로써, 해당 메서드가 호출되기 전에 Xlet에서는 destroyXlet( ) 메서드가 호출된다. 그리고 notifyPaused( ) 메서드는 notifyDestroyed( )와 유사하게 Xlet 자체에서 스스로 중지를 수행한 후 XletManager에게 PAUSED 상태를 알리는 메서드로써, Xlet 자체에서는 pauseXlet( ) 메서드가 호출된다. 마지막으로, resumeRequest() 메



〈그림 3〉 Xlet, XletContext, 그리고 XletManager의 관계

서드는 앞의 두 메서드 즉, notifyDestroyed( )나 *notifyPaused*( )와는 달리 Xlet 스스로가 자기 자신을 활성화 시키고 난 뒤에 XletManager에게 알리기 위한 메서드가 아니라, XletManager에게 Xlet 자체에 대한 활성화를 요청하는데 이용되는 메서드이다.

3. XletManager

본 절에서는 실제 Xlet을 관리하는 주체인 어플리케이션 라이프사이클의 관리자 즉, XletManager에 대해 기술한다. XletManager는 ApplicationManager라는 용어로 더 일반적으로 사용되는데, XletManager는 Xlet을 로딩하고, Xlet의 상태를 추적하며, 그리고 Xlet의 상태 변경을 위해 Xlet 내부의 라이프사이클 메서드를 호출함으로써 Xlet을 운영/관리하는 역할을 수행한다[5]. 이와 같이 XletManager는 Xlet의 통제 역할을 수행하게 되는데, 이에 대해 좀 더 구체적인 요구사항을 정리하면 다음과 같다.

먼저, XletManager는 Xlet의 상태를 변경할 수 있어야 한다. 왜냐하면, XletManager의 가장 기본적인 역할이 바로 Xlet의 상태 변경을 수행하는 것이기 때문이다. 즉, XletManager는 Xlet을 생성/초기화 하고, 시작시키며, 중지/종료시킬 수 있어야 한다는 요구이다. 또한, XletManager는 Xlet이 어떠한 상태에 있더라도 해당 Xlet을 자유롭게 종료시킬 수 있어야 한다. 이는 XletManager가 항상 Xlet의 상태를 추적하고 있어야 한다는 것을 의미하는데, 즉 XletManager는 Xlet의 상태를 직접 통제할 수 있어야 하고, 반대로 Xlet은 XletManager가 해당 Xlet을 통제하는데 용이하도록 지원해야 한다는 의미가 된다. 추가적으로, XletManager는 Xlet의 상태를 관리하기 위한 메커니즘 즉, Xlet을 로딩하고, 런칭(launching) 및 종료시킬 수 있는 메커니즘을 포함하고 있어야 한다. 또한, XletManager는 메모리 관리 로직을 포함할 수 있는데, 예를 들어 메모리 용량을 고려하여 Xlet의 상태를 변경할 수 있거나, 혹은 Xlet 내부에서 사용하는 각종 리소스를 관리할 수 있어야 한다는 요구이다.

4. Xlet, XletContext, 그리고 XletManager의 관계

지금까지 Xlet, XletContext, 그리고 XletManager에 대해 기술하였다. 이들을 보통 어플리케이션 라이프사이클의 주요 인자라고 하는데, 이들 라이프사이클 인자들 간에는 상호 밀접한 관계를 가지고 있다. <그림 3>은 각 인자들 간의 관계를 도식적으로 표현한 그림으로써, 지금까지 설명한 내용을 종합한 것이 된다.

이 세 가지의 주요 아이템들 간의 관계는 크게 두 가지 측면으로 나누어 생각해 볼 수가 있다. 하나는 XletManager에서 Xlet쪽의 측면이 있고, 다른 하나는 Xlet에서 XletManager 쪽의 측면이 있다. 그리고 이들 두 측면의 인터페이스는 바로 XletContext에 의해 이루어 진다. <그림 3>에서와 같이 XletManager는 Xlet의 라이프사이클 메서드들 즉, initXlet( ), startXlet( ), pauseXlet( ), 그리고 destroyXlet( ) 메서드를 호출하여 해당 Xlet의 상태를 변경할 수 있고, 반대로 Xlet은 자기자신을 중지 혹은 종료시켜 자신의 상태를 XletContext를 통해 XletManager에게 통보하는 그러한 관계가 성립될 수가 있다.

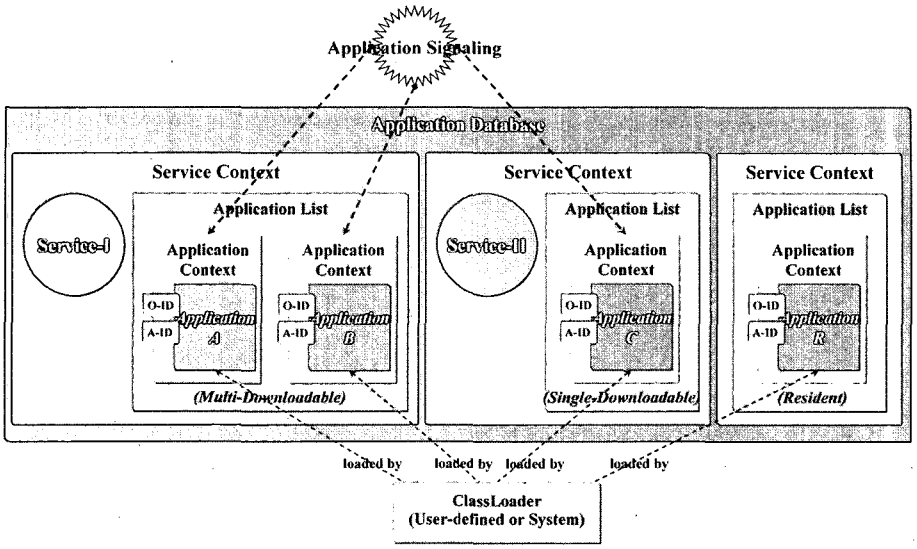
III. 어플리케이션 라이프사이클

1. 어플리케이션 라이프사이클의 프레임워크 (framework)

지금까지 라이프사이클의 주요 인자들에 대한 요구사항 및 상호 관계에 대해서 기술하였다. 본 절에서는 어플리케이션 라이프사이클의 전체 구조라든지, 내부 조건, 혹은 프로세스에 대해서 기술한다. <그림 4>은 본 원고에서 제시하고 있는 어플리케이션 라이프사이클의 전체 프레임워크이다. 어플리케이션 라이프사이클의 프레임워크는 GEM의 "Application Listing and Launching" API를 구현하기 위한 구조를 제시해 주고 있다. 실제로 해당 API를 구현하기 위해서는 내부에 정의되어 있는 각종 클래스들 간의 관계를 정립해야 하는데, 해당 프레임워크는 주요 클래스들 간의 상호 관계를 명확히 제시해 주고 있다.

어플리케이션 라이프사이클의 구성은 크게 XletManager 역할을 수행하는 Application Database와 서비스에 대한 환경을 제공하는 Service Context, 그리고 Service, 또한 해당 서비스에 포함되어 있을 Application과 해당 어플리케이션의 환경을 제공하는 Application Context로 구성될 수 있다. 각 어플리케이션의 식별은 organization_id와 application_id로 이루어지고, 각 어플리케이션의 로딩은 ClassLoader에 의해서 이루어질 수 있도록 구성되어 있다. 여기서 어플리케이션에 대한 식별 방법으로써 organization_id와 application_id가 이용되는데, organization_id는 하나의 서비스에 대해 유일하고, application_id는 하나의 organization_id에 대해 유일하다. 즉, organization_id는 다른 서비스에 동일한 organization_id가 존재할 수 있고, 마찬가지로 다른 organization_id에 대해 동일한 application_id가 존재할 수 있다는 것이다.

서론에서 설명하였듯이, 어플리케이션 라이프사이클은 어플리케이션에 대한 전체 수명주기를 관리하는 것을 주목적으로 한다. 따라서 이러한 구조상에서 이 부문에 대한 주요 역할은 어플리케이션이 시그널링(signalling) 되는 상황에서 서비스가 선택이 되면 ServiceContext가 구성이 되고, 서비스 내에 어플리케이션이 포함되어 있을



〈그림 4〉 어플리케이션 라이프사이클의 블록 구조도

경우 해당 어플리케이션을 다운로딩(downloading)하여 수행시키고 제어하는 즉, 어플리케이션 통제 역할을 수행하는 것이 어플리케이션 라이프사이클의 주요 역할이 된다. 이러한 어플리케이션의 통제 역할은 서비스의 선택으로부터 시작되는데, 서비스가 선택이 된 경우 해당 서비스는 ServiceContext 상에 등록된다. 여기서, ServiceContext는 서비스를 등록시키는 환경으로써 서비스에 대한 영역(boundary)으로 정의된다. 또한, ServiceContext는 한번에 하나의 서비스만을 등록하게 되는데, 예를 들어 이전 서비스가 실행 중에 있는 상황에서 새로운 서비스가 선택이 될 경우, 이전 서비스에 대한 ServiceContext가 종료되고 새로운 서비스에 대한 ServiceContext가 구성이 된다. 즉, 하나의 ServiceContext는 하나의 서비스만을 유지가 한다는 의미이다. 결과적으로 라이프사이클 블록 다이어그램에서 어플리케이션 라이프 사이클의 시작은 바로 서비스의 선택으로부터 시작이 된다는 것이다.

2. 어플리케이션의 시작과 종료

본 절에서는 어플리케이션 라이프사이클 부문에서 가장 중요하게 다루어져야 되는 어플리케이션의 시작과 종료에 대해 기술한다. 먼저 어플리케이션의 시작과 종료에 관한 내용을 설명하기 전에 어플리케이션 통제 코드(control code) 값의 의미에 대해 설명하겠다. 어플리케이션의 통제 코드라는 것은 단말에서 구동되는 어플리케이션을 서버 측에서 통제하기 위해서 AIT(application information table) 테이블 내에 정의해 놓은 코드를 어플리케이션 통제 코드라고 한다. 일반적으로, 어플리케이션 라이프사이클 부문에서는 어플리케이션 시그널링이라는 말을 많이 사용하는데, 이러한 시그널링의 수단이 되는 것이 바로 어플리케이션 통제 코드이다.

<표 1>은 AIT 테이블 내에 정의되어 있는 application_control_code라는 필드 값의 의미를 표로 정리한 것이다. 보는 바와 같이 통제 코드

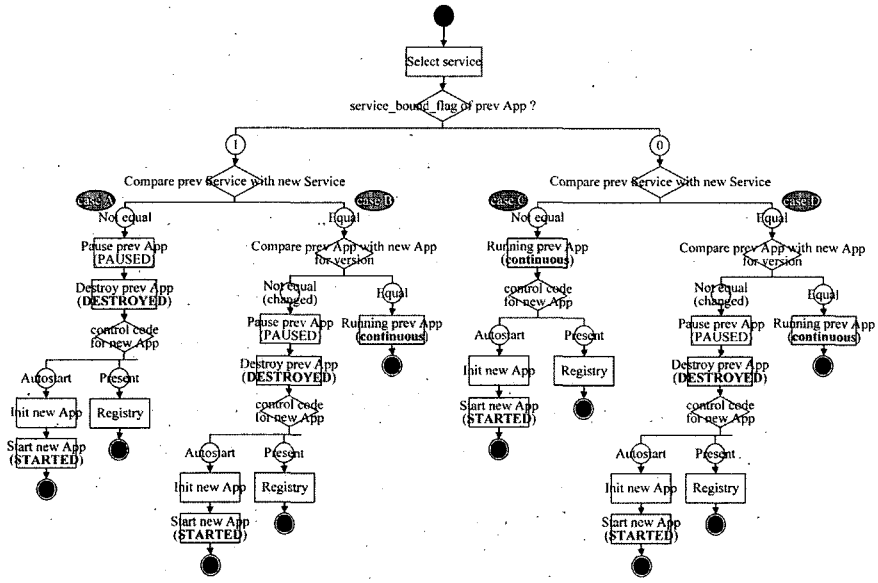
〈표 1〉 어플리케이션 통제코드

Code	Identifier	Semantics
0x00		reserved
0x01	AUTOSTART	The file system element(s)(e.g. an Object Carousel module) containing the class implementing the Xlet interface is loaded, The class implementing the Xlet is loaded into the VM and an Xlet object is instantiated, and the application is started subject to usual restrictions, etc.
0x02	PRESENT	Indicates that the application is present in the service, but is not autostarted.
0x03	DESTROY	When the control code change from AUTOSTART or PRESENT to DESTROY, the destroy method of the Xlet is called(with the unconditional parameter set to false) by the application manager and the application is allowed to destroy itself gracefully
0x04	KILL	When the control code change from AUTOSTARY or PRESENT to KILL, the destroy method of the Xlet is called(with the unconditional parameter set to true) by the application manager.

의 유효 값으로는 '1', '2', '3', 그리고 '4'가 있고, 각각 AUTOSTART, PRESENT, DESTROY, 그리고 KILL 이라는 의미를 가지고 있다. 여기서 AUTOSTART의 의미는 어플리케이션을 시작시키라는 의미이고, PRSENT는 해당 서비스에 등록시키라는 것을 의미하며, 그리고 DESTROY와 KILL은 해당 어플리케이션을 종료시키라는 의미가 된다. 여기서 DESTROY와 KILL은 해당 어플리케이션을 종료시킨다는 점에서 동일한 의미를 갖고 있으나, 종료 과정에서 프로세스 상의 차이점이 존재한다. 즉, 일반적으로 어플리케이션이 종료되는 상황에서 다양한 에러가 존재할 수 있는데, 이러한 경우 DESTROY는 해당 문제가 해결될 때까지 대기하다가 문제가 해결이 되면 그때 해당 어플리케이션을 제거하라는 의미가 되고, KILL은 DESTROY와 달리 문제가 발생하더라도 해당 문제와 관계없이 어플리케이션을 바로 종료시키라는 의미가 된다.

결론적으로 어플리케이션의 시작은 바로 어플리케이션 통제 코드의 AUTOSTART라는 명령에 의해 이루어지고, 어플리케이션의 종료는 DETROY 혹은 KILL이라는 서버 명령에 의해 이루어 진다. 그러나 여기서 문제가 되는 것이 바로 어플리케이션의 종료이다. 왜냐하면, 단순히 어플리케이션의 종료는 서버 측의 종료 명령에 의해서만이 이루어지는 것이 아니기 때문이다. 일반적으로 어플리케이션의 종료에는 여러 가지 다양한 상황들이 존재할 수가 있고, 그에 따른 조건들이 존재할 수가 있다. 다음의 내용은 바로 그러한 조건별 상황들에 대해 정리한 내용이다.

첫 번째 상황은 새로운 서비스가 선택이 되는 경우 이전 어플리케이션이 종료되는 상황이다. 즉, 새로운 서비스가 선택이 되면, 새로운 서비스 내에 포함되어 있는 어플리케이션이 구동되



〈그림 5〉 어플리케이션의 시작과 종료

면서 이전 서비스 영역 내에서 구동되고 있는 어플리케이션이 종료되는 경우이다. 여기서 어플리케이션의 종료 조건은 바로 service_bound_flag가 되는데, 여기서 service_bound_flag는 AIT 테이블 내에 포함되어 있는 application descriptor에 정의되어 있는 필드로써, 특정 어플리케이션이 해당 서비스 영역 내에서 구동되는가를 확인하는 필드를 의미한다. 그리고 이 필드의 값으로는 '0' 또는 '1'이 존재하는데, '0'인 경우에는 어플리케이션이 해당 서비스 영역 내에서 구동되고 있지 않다라는 의미 즉, 해당 서비스와 관계없이 어플리케이션이 구동이 된다라는 것을 의미한다. 반면에, '1'인 경우에는 어플리케이션이 해당 서비스 영역 내에서 구동되고 있다라는 의미 즉, 서비스가 종료되면 해당 어플리케이션도 같이 종료된다라는 것을 의미한다. 따라서, 첫 번째 상황에서는 service_bound_flag가 '1'인 경우에 한해서 해당 어플리케이션이 종료되는 경우가 된다. 두 번째 상황은 임의의 어플리케이션이 다른 어플리케이션을 종료시키는 경우인데, 여기서 종료 조건은 단말기의 보안(security) 정책이 된다. 즉, 특정 어플리케이션이 다른 어플리케이션을 종료시키기 위해서는 그러한 권한을 가지고 있어야 한다는 의미가 되는데, 이러한 권한 정책에 따라 임의의 어플리케이션이 다른 어플리케이션을 종료시킬 수 있는 경우가 된다. 세 번째의 상황은 앞서 설명한 어플리케이션 통제 코드에 의해 어플리케이션이 종료되는 상황이다. 즉, 서버 측에서의 어플리케이션 시그널링을 통해 해당 어플리케이션이 종료되는 경우이다. 마지막의 경우는 단말기의 성능 및 자원의 상황에 따른 어플리케이션의 종료인데, 이 경우는 단말기가 어플리케이션을 운영할 수 있는 한계 범위에 도달했을 때, 해당 어플리케이션의 우선순위에 따라서 어플리케이션을 종료시킬 수 있다라는 의미이다. 여기서 우선순위는 각 어플리케이션들 간의 중요도를 의미하는 것으로써, 우선순위가 가장 낮은 어플리케이

션이 가장 먼저 종료되는 조건이 된다.

지금까지 설명한 내용을 바탕으로 어플리케이션의 시작과 종료에 대해 조건별로 수행될 수 있는 상황을 프로세스로 표현하면 <그림 5>와 같다.

3. AIT(Application Information Table) 시그널링

AIT는 어플리케이션 대한 정보를 포함하고 있는 테이블로써[1][2], 앞서 설명하였던 어플리케이션 시그널링을 위한 용도로 쓰이는 테이블이다. AIT의 구성은 일반적인 시그널링 정보를 포함하고 있는 common descriptor loop와 어플리케이션과 관련된 정보를 담고 있는 application descriptor loop로 구성되어 있는데, 본 절에서는 어플리케이션 시그널링을 위해 기본적으로 요구되는 AIT 테이블 내의 정보에 대해서 설명한다.

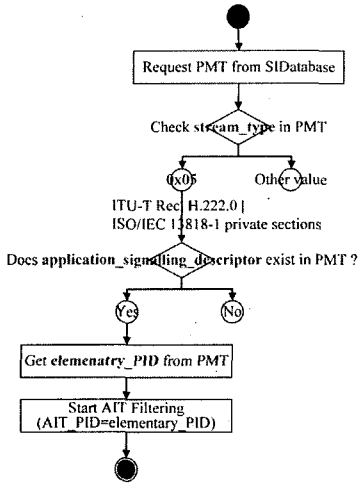
GEM에서는 어플리케이션 시그널링을 위해 요구되는 기본 정보로써, 다음의 사항을 정의하고 있다. 어플리케이션의 기본 식별 정보로서 어플리케이션 소스 및 각종 리소스 파일을 식별하는 정보, application_id와 organization_id를 식별하는 정보, 그리고 어플리케이션 이름을 식별하는 정보를 포함하는 것을 정의하고 있다. 또한, 어플리케이션의 시그널링 파라미터 정보와 어플리케이션의 생성을 위한 메인(main) 클래스 파일을 가리키는 정보를 추가적인 정보로 정의하고 있다[2]. GEM에서는 이에 대해 어플리케이션의 서술(application description)로 정의하고 있는데, MHP에서 정의하고 있는 AIT의 common descriptor loop와 application descriptor loop가 이에 해당된다[1].

common descriptor loop는 일반적인 시그널링에 요구되는 정보로써, application signalling

	No.of Bits	Identifier
application_information_section() {		
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved_future_use	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
application_type	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
reserved_future_use	4	bslbf
common_descriptors_length	12	uimsbf
for(i=0;i<N;i++){		
descriptor()		
}		
reserved_future_use	4	bslbf
application_loop_length	12	uimsbf
for(i=0;i<N;i++){		
application_identifier()		
application_control_code	8	uimsbf
reserved_future_use	4	bslbf
application_descriptors_loop_length	12	uimsbf
for(j=0;j<N;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

〈그림 6〉 AIT(Application Information Table)

descriptor, transport protocol descriptor, application descriptor, application name descriptor 등이 있다. Application signalling descriptor는 PMT에 포함되어 있는 descriptor로써 스트림 내에 AIT가 포함되어 있는가를 확인하기 위한 용도의 descriptor이고, transport protocol descriptor는 data broadcasting 부문의 DSM-CC(digital storage media-command and control) 필터링(filtering)에 요구되는 정보인 component_tag 정보가 포함되어 있는 descriptor이며, application descriptor는 어플리케이션의 종료 조건에 쓰이는 service_bound_flag, 그리고 application name descriptor는 어플리케이션에 대한 기본적인 이름이 포함되어 있는 descriptor 이다. 추가적으로 DVB-J application descriptor와 DVB-J application location descriptor가 있는데, 이들은 GEM-J 어플리케이션과 관련해서 파라미터 정보라든지, 패스정보 등의 해당 어플리케이션과 관련된 추가적인 정보를 제공하는 descriptor이다.



	No.of Bits	Identifier
progam_map_section() {		
table_id	8	uimsbf
...		
for(i=0;i<N1;i++){		
stream_type	8	uimsbf
reserved	3	bslbf
elementary_PID	13	uimsnf
reserved	4	bslbf
ES_info_length	12	uimsbf
for(j=0;j<N2;j++){		
descriptor()		
}		
}		
CRC_32	32	rpchof
}		

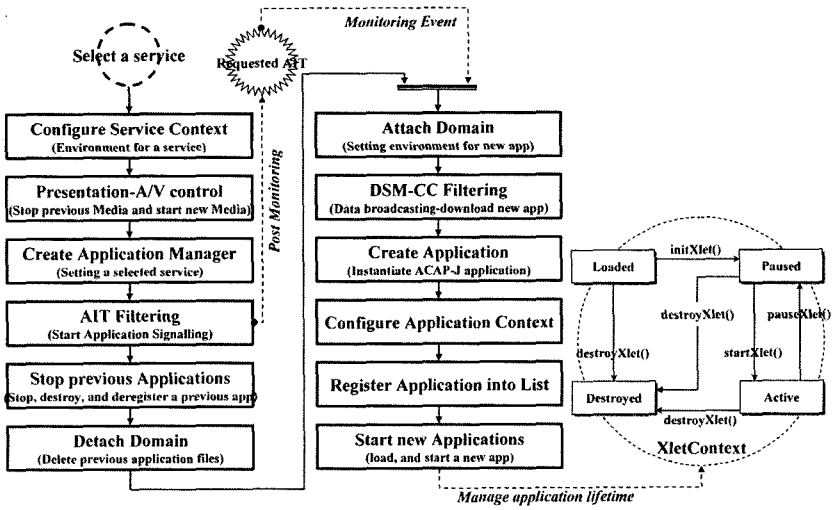
〈그림 7〉 AIT 필터링

<그림 7>는 AIT 시그널링 혹은 AIT 필터링이라고 해서 실제 AIT를 어플리케이션 라이프사이클 부문에서 얻는 과정을 표현한 것이다. 아래 프로세스 다이어그램에서 표현한 바와 같이 AIT 필터링은 SI(service information) 데이터베이스로부터 PMT를 요구하는 것으로부터 시작이 된다. 일단, PMT가 얻어지면, PMT에서 stream_type을 확인하게 되는데, stream_type이 '0x05' 인가를 확인한다. 만약 stream_type이 '0x05'인 경우, 다시 PMT 내에서 application signalling descriptor가 포함되어 있는가를 확인한다. 그리고 만약에 포함되어 있는 경우 바로 해당 elementary stream의 PID를 AIT의 PID로 지정해서 SF(section filter) 부문에 지정된 PID에 해당되는 AIT를 요청함으로써 AIT 필터링이 수행된다.

4. 어플리케이션 시그널링

본 절은 지금까지 기술한 내용을 바탕으로 어플리케이션의 시그널링을 위한 개략적인 프로세스에 대해 기술한다. <그림 8>은 어플리케이션 시그널링을 중심으로 한 어플리케이션 라이프사이클 부문의 전반적인 처리 흐름을 표현한 것이다.

<그림 8>과 같이 어플리케이션 라이프사이클 부문의 시작은 서비스의 선택으로부터 시작이 되는데, 서비스가 선택이 되면 ServiceContext가 구성이 되고, 그리고 미디어(media) 부문에서는 A/V 튜닝(tuning)이 일어나게 된다. 그리고 선택된 서비스는 ApplicationManager에게 지정 혹은 등록이 되는데, 이러한 서비스에 포함되어 있는 어플리케이션을 다운받기 위해서는 먼저 AIT 필터링이 수행된다. 단, 여기서 AIT 필터링은 AIT의 버전을 지속적으로 체크하도록 모니터링(monitoring)으로 필터링을 요청한다. 그리고 서비스의 바운딩에 따라 이전 어플리케이션을 종료시키고 그와 동시에 이전 어플리케이션과 관련된 정보를 삭제한다. 또한, AIT가 요청이 되면 해당 서비스와 관련된 새로운 어플리케이션에 대한 환경을 설정하게 되는데, 이 때 data broadcasting 부문에 새로운 어플리케이션을 다



〈그림 8〉 어플리케이션 시그널링

운로딩 하도록 요청한다. 그리고 어플리케이션의 다운로딩이 완료되면, 어플리케이션의 통제 코드에 따라 해당 어플리케이션을 시작시키고, 지속적으로 그 어플리케이션의 상태를 관리하도록 어플리케이션 라이프사이클 부문의 프로세스가 구성될 수가 있다.

Ⅳ. 결 론

본 원고에서는 GEM의 어플리케이션 라이프사이클 부문에 초점을 맞춰 데이터방송 컨텐츠가 상호 운용되기 위한 어플리케이션 관리 방안에 대해 기술하였다. 특히, A/V를 제외한 데이터 방송 컨텐츠의 실체인 Xlet과 XletContext, 그리고 XletManager에 대한 요구사항을 도출하였고, 각 인자들 간의 상호 관계를 정립하였다. 또한, 이를 바탕으로 데이터방송 컨텐츠의 운영을 담당하는 미들웨어 모듈인 어플리케이션 라이프사이클 부문의 구조와 내부 시그널링 프로세스에 대해 제안하였다.

본 원고에서 제안하고 있는 어플리케이션 관리 방안은 각 방송플랫폼간의 조화를 고려하고 있는 상황에서 해당 플랫폼에서의 어플리케이션 운용과 관련된 모듈을 구성하는데 공통적으로 적용될 수 있을 것으로 기대된다. 또한, 각 방송플랫폼 간의 어플리케이션의 상호 운용성을 고려함에 있어 어플리케이션 관리 측면에서의 접근 방법으로 제안될 수 있을 것으로 기대된다.

서론에서 필자는 국내에서 상이한 방송플랫폼으로 인해 다양한 조화의 문제들이 이슈화되고 있다고 하였다. 특히, 각 표준들 간의 상호 운용성 문제가 크게 대두되고 있다고 하였는데, 본 원고는 이러한 상호 운용성 문제에 대해 어플리케이션 관리 측면에서의 해결 방안을 고려하고 있다. 그러나, 본 원고에서 제시하고 있는 방안은 어플리케이션이 공통의 표준인 GEM에 정의되어 있는 API만을 사용해야 한다는 것을 전제로 하고 있다. 반대로, 이것은 어플리케이션이 각 플랫폼에 한정되어 있는 API를 사용하는 경우 어플리케이션의 상호 운용성을 보장할 수 없

다는 한계를 가지고 있다. 따라서, 향후에는 이러한 어플리케이션 관리 측면을 바탕으로 어플리케이션의 구성과 관련된 API 측면 즉, 특정 플랫폼에 국한되어 있는 API를 어플리케이션에서 사용하는 경우 각 플랫폼 간의 상호 운용성을 어떻게 보장할 것인가에 대해 고려해야 할 필요가 있을 것으로 판단된다.

참고문헌

[1] Digital Video Broadcasting(DVB); Multimedia Home Platform(MHP) Specification 1.0.2, TS 101 812 V1.2.1, ETSI, Jun.2002.

[2] Digital Video Broadcasting(DVB); Globally Executable MHP(GEM), TS 102 819 V1.2.1, ETSI, May.2004.

[3] Advanced Common Application Platform(ACAP), A/101, ATSC, Aug.2005.

[4] OpenCable™ Application Platform Standard Specification; OCAP1.0 Profile, ANSI/SCTE 90-1, SCTE, Nov.2003.

[5] JavaTV™ API Technical Overview Version 1.0, Sun Microsystems, Nov.2000.

[6] 한국전자통신연구원, 방송 · 통신 융합에 대비한 기술개발전략 연구, 한국전자통신연구원, Nov.2003.

[7] 양창모, 조위덕, 지능형 Interactive 데이터방송 기술(iDTV+MPEG-4/7), 전자부품연구원, Apr.2002.

용어해설

윈도우 미디어 포토
Windows Media Photo, WMP [화상통신]

마이크로소프트사에서 개발한 새로운 이미지 포맷.
차세대 운영체제인 윈도우 비스타에 적용될 예정으로 화질을 유지하는 상태에서 JPEG이 최대 12:1 정도의 압축율을 제공하는데 비해 WMP는 25:1의 압축율을 제공한다. WMP는 비손실, 손실 압축을 모두 지원하는 대칭형 직교 변환 알고리듬을 사용한다. 특별한 하드웨어나 복잡한 수학 연산을 필요로 하지 않으며 이미지 엔코딩과 디코딩을 교차적으로 할 수 있어 버퍼와 메모리의 요구량도 작다. 또한, 메타 데이터를 포함한 기존 TIFF 컨테이너를 사용하였으므로 기존 시스템과도 호환성이 있다.

사용자 제작 콘텐츠
User Created Contents, UCC,
使用者制作- [관리운용]

인터넷 사업자나 콘텐츠 공급자가 아닌 일반 사용자들이 직접 만들어 유통되는 콘텐츠.
사용자가 질문하고, 또한 사용자가 각종 지식과 경험을 댓글해 주는 포털의 지식검색 서비스나 위키(wiki)사전 등이 UCC의 효시라 할 수 있다. UCC는 텍스트에 이어 최근 이미지 · 동영상 · 음악 등 멀티미디어로 분야를 확대되어 가는 추세이다. 또한, 웹2.0에서는 사용자들이 보다 다양한 정보들 창조하고, 공유할 수 있어이 또한 향후 유통괴는 콘텐츠의 중요한 부분을 차지하게 될 전망이다.

저자소개

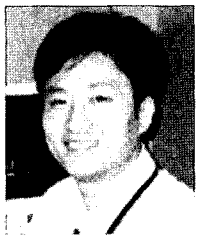
차 경 호

2000년 울산대학교 산업공학과(학사)
2002년 울산대학교 산업공학과(공학석사)
2002년 - 2005년 (주)아이셋 DTV사업부 팀장
2006년 - 현 재 한국전자통신연구원 연구원
주관심분야 디지털/데이터방송 기술

석 주 명

1997년 경희대학교 전자공학과(학사)
1999년 경희대학교 전자공학과(공학석사)
2002년 경희대학교 전자공학과(박사과정 수료)
1999년 - 현 재 한국전자통신연구원 선임연구원
주관심분야 멀티미디어 통신, IP 스트리밍 기술, 융합 방송서비스 개발

저자소개



이 한 규

1994년 경북대학교 전자공학과(학사)
1996년 경북대학교 전자공학과(공학석사)
1996년 - 현 재 한국전자통신연구원 맞춤형방송연구팀장 선임연구원
주관심분야 멀티미디어 서비스, 영상신호 분석 및 처리

홍 진 우

1982년 광운대학교 응용전자공학과(학사)
1984년 광운대학교 전자공학과(공학석사)
1993년 광운대학교 전자계산기공학과(공학박사)
1998년 - 1999년 독일 프라운호퍼연구소 파견연구원
1984년 - 현 재 한국전자통신연구원 방송미디어연구그룹장 책임연구원
2000년 - 현 재 한국음향학회 교육이사 및 뉴미디어음향 학술분과위원장, 한국방송공학회 편집위원, 한국해양정보통신학회 학술분과위원장
주관심분야 오디오 신호처리 및 부호화, 디지털 콘텐츠 보호 및 관리, 디지털 방송 기술, MPEG 21 멀티미디어 프레임워크 기술