논문-04-09-1-08

연동형 데이터 방송 애플리케이션의 구조

정 문 열*, 백 두 원**

The Structure of Synchronized Data Broadcasting Applications

Moon-Ryul Jung* and Doowon Paik**

요 약

디지털 방송에서 애플리케이션은 셋톱박스(TV 수신기)에서 수행되는 응용프로그램을 말하며, 연동형 애플리케이션은 비디오내의 특정 시각에 정해진 일을 수행하는 애플리케이션이다. 본 논문에서는 연동형 애플리케이션을 구현하는데 필요한 개념, 표준 및 기술을 분석하고, 이들을 통합하여 연동형 애플리케이션을 구현하는 방법을 제안한다. 본 논문은 특히 유럽식 데이터 방송 표준인 DVB-MHP를 염두에 두고 기술한다. DVB-MHP 규약에서는 비디오와 애플리케이션을 동기화시키는 방법으로 스케줄드 스트림 이벤트 방법을 권장하고 있다. 이 방법은 방송중간에 스케줄드 스트림 이벤트를 전송하고 애플리케이션이 스트림 이벤트를 수신, 스트림 이벤트에 명시된 시각에 스트림 이벤트를 처리하는 방법이다. 현재 상용으로 사용되고 있는 전송스트림 생성기, 즉, 다중화기는 스트림 이벤트를 사용할 수 있도록 전송스트림을 생성해 주지 않기 때문에 본 논문에서는 본 연구팀이 개발한 실험실용 스트림 생성기를 이용하여 필요한 전송스트림을 생성하였다. 그리고 실제로 하나의 연동형 애플리케이션을 구현하여 테스트했다. 그 결과 연동형 애플리케이션이 스트림 이벤트를 성공적으로 처리함을 확인하였다. 구현된 애플리케이션에서 스트림 이벤트들은 의도한 시점으로부터 240ms 이내에서 비디오와 동기화되었는데, 이 시간은 비디오와 그래픽 이미지간의 동기화 오차에 대한 허용치이다.

Abstract

In digital broadcasting, applications are computer programs executed by the set-top box(TV receiver), and synchronized applications are those that perform tasks at the specified moments in the underlying video. This paper describes important concepts, standards, and skills needed to implement synchronized applications and explains how to integrate them to implement these applications. This paper assumes the European data broadcasting standard, DVB-MHP. In DVB-MHP, scheduled stream events are recommended as a means of synchronizing applications with video streams. In this method, the application receives each stream event, and executes the action associated with the stream event at the time specified in the stream event. Commercially available stream generators, i.e., multiplexers, do not generate transport streams that support scheduled stream events. So we used a stream generator implemented in our lab. We implemented a synchronized application where the actions triggered by stream events are to display graphic images. We found that our synchronized application processes scheduled stream events successfully. In our experimentation, the stream events were synchronized with the video and the deviation from the intended time was within 240 ms, which is a tolerance for synchronization skew between graphic images and video.

Keywords : data broadcasting, synchronized applications, stream events, synchronization skew

* 서강대학교 영상대학원 미디어공학과
Dept. of Media Technology, Sogang University

** 숭실대학교 미디어학부
School of Media, Soongsil University

Ⅰ. 서 론

디지털 방송은 다채널과 고화질 서비스이외에도 데이터 방송(data broadcasting)을 가능하게 한다. 데이터 방송은 데이터를 비디오와 함께 방송하여 시청자에게 전달하는 것을 의미하는데, 여기서 데이터라 함은 실행 가능한(executable) 애플리케이션 프로그램과 애플리케이션이 사용하는 텍스트나 이미지 파일과 같은 데이터를 총칭한다. 데이터 방송 표준으로는 북미의 ATSC-DASE[1]와 OCAP[2], 유럽의 DVB-MHP[3] 등이 있는데, 본 논문은 우리나라 위성방송이 채택한 DVB-MHP를 다룬다. 애플리케이션은 Xlet 이라고도 불리는 MHP-J 와 MHP-HTML이 있는데, 본 논문에서는 전자만을 고려한다. 그것은 본 논문의 주제가 애플리케이션과 비디오간의 동기화인데, Java 프로그램인 MHP-J 애플리케이션만 비디오와의 동기화를 처리할 수 있기 때문이다. 애플리케이션과 비디오와의 동기화란 구체적으로 애플리케이션이 어떤 일을 처리할 때 그것을 비디오의 특정 시각에 처리하는 것을 말한다. 이런 애플리케이션을 연동형 애플리케이션이라 한다. 데이터 방송 애플리케이션에 대한 포괄적이고 심도 있는 설명은 문헌 [4], [10]을 참조하라.

본 논문은 연동형 애플리케이션을 구현하는데 필요한 개념, 표준 및 기술을 분석하고, 이들을 통합하여 연동형 애플리케이션을 구현하는 방법을 제안한다. 그리고 실제적으로 구현한 연동형 애플리케이션의 실험결과를 제시한다. TV에서 수행되는 애플리케이션은 많은 사람에게 낯설지만, 특히 비디오와 동기화된 연동형 애플리케이션의 경우는 더욱 그러하다. 연동형 애플리케이션의 구현은 방송규약 뿐 아니라, Java의 이벤트 처리 모델, 그리고 애플리케이션에 대한 이해가 필요하므로 본 논문은 이들을 연결하여 연동형 애플리케이션의 전체적인 구조를 파악하는데 중점을 두고자 한다. 이것은 TV를 위한 애플리케이션 프로그래머들이 그들이 하는 일을 정확하게 이해하면서 일을 하도록 도움을 줄 것이다.

DVB-MHP에서는 비디오와 애플리케이션을 동기화시키는 방법으로 (스케줄드) 스트림 이벤트를 이용하는 방법을 권하고 있다. 이 방법은 방송중간에 스트림 이벤트를 전송스트림에 포함시켜 전송하고 수신기의 애플리케이션이 스트림 이벤트에 명시된 시각에 이 스트림 이벤트에 연관된 일을 처리하는 방법이다[3]. 스트림 이벤트에서는 스트림 이벤트 처리시각을 표시하는 시간 기준으로 NPT(Normal Play Time)를 사용하도록 되어 있다. NPT는 단위 방송 프로그램내부의 시간으로 특정 NPT 값은 해당 프로그램이 언제 방송되는가와 관계없이 언제나 해당 프로그램내의 동일한 위치를 가리키는 시간이다.

비디오와 동기화된 애플리케이션을 실험하기 위해서는 애플리케이션 자체의 구현뿐만 아니라, 이 애플리케이션이 사용하는 여러 가지 스트림을 생성하여 전송스트림(TS)에 삽입하는 일이 필요하다. 현재 상용으로 나와 있는 TS 다중화기는 연동형 애플리케이션에 필요한 스트림 이벤트 서술자와 NPT참조서술자를 삽입하는 기능이 완비되어 있지 않으므로, 연동형 애플리케이션 실험을 위해 본 논문에서는 본 연구팀이 실험실에서 제작한 연동형 애플리케이션을 위한 스트림 생성기[11][13]을 사용하였다.

본 논문은 2장에서는 디지털/데이터방송 기술과 Java의 이벤트 구동 방식을 간략히 설명하고 3장에서는 연동형 애플리케이션의 구현 방법을 상세히 기술하며 4장에서는 구현된 애플리케이션의 검증을 위한 실험방법 및 결과를 기술한다.

Ⅱ. 디지털/데이터 방송 개요, Java 이벤트 처리 방식, 애플리케이션의 구조

본 장에서는 본 논문을 이해하는데 필요한 디지털/데이터 방송의 기본 개념, 스트림 이벤트, 이벤트 처리시각의 기준시간인 NPT, Java 이벤트 구동 방식, 그리고 연동형 애플리케이션의 구조를 설명한다.

1. 서비스와 이벤트

DVB에서는 방송의 기본단위를 서비스(Service)라고 한다. 하나의 서비스는 보통 하나의 채널에서 방송하는 방송물 전체를 가리킨다. 하나의 서비스는 보통 여러 개의 성분 스트림(Elementary Stream, 비디오/오디오/데이터 스트림)을 포함한다. 그리고 애플리케이션과 이것이 사용하는 데이터 스트림도 하나의 서비스의 구성요소이다. 디지털 방송에서는 서비스들을 수신기로 전송하기 위해서 MPEG-2 System[5]에서 정의된 MPEG-2 TS(Transport Stream)를 사용한다. 우리가 흔히 말하는 방송 프로그램 (9시 뉴스, 한일전의 전반전)을 이벤트(Event)라고 한다. 하나의 이벤트는 시작시각과 끝나는 시각이 있다. 이벤트를 서비스를 시간대별로 구분한 것이므로 한 이벤트는 다수의 성분 스

트림을 포함할 수 있다. "이벤트"는 "방송 프로그램" 이라는 의미 외에 보다 일반적으로 "사건" 이라는 의미로 쓰인다. 본 논문에서도 두 가지 의미로 사용되는데, "이벤트"가 프로그램을 가리킨다는 것을 확실히 하고자 할 때는 "이벤트 (프로그램)" 이라고 표기한다.

2. 객체 카루셀

DVB-MHP 표준에서는 서비스의 구성요소중의 하나인 애플리케이션과 애플리케이션이 사용할 데이터를 전송하기 위해 DSMCC Object Carousel을 기본으로 해서 정의된 DVB Object Carousel을 사용한다[3][6][10]. 일반 컴퓨터 프로그래밍에서 프로그래머가 파일에 대한 이해 없이 응용 프로그램을 구현하는 것을 생각할 수 없듯이 TV 프로그래머가 파일 시스템에 해당하는 객체 카루셀(Object Carousel)에 대한 이해 없이 TV 프로그래밍을 잘 하기를 기대할 수 없다. 객체 카루셀은 객체의 전송을 위한 프로토콜인데 MHP 수신기 입장에서 보면 쓰기가 불가능하고 읽기만 가능한 파일 시스템이라고 볼 수 있다. 그러나 파일 시스템이 그 안에 모든 데이터를 가지고 있는데 비해, 객체카루셀은 스트림에 대한 참조(reference) 정보만 가지고 있고, 스트림 자체는 전송 스트림(TS)을 통해 전송된다. 애플리케이션이 TS를 통해 전송되는 스트림을 읽으려면 일단 객체카루셀을 통해 이에 대한 참조정보를 획득한 후 읽을 수 있도록 되어 있다. 객체 카루셀은 데이터 카루셀(Data Carousel)로 인코딩되고 다시 MPEG-2 섹션으로 인코딩된 후에 MPEG-2 TS로 패킷화 되어 전송된다.

Object Carousel 규약에서 말하는 객체의 종류로는 디렉터리 객체(directory object), 파일 객체(file object), 스트림 객체(stream object) 및 스트림 이벤트 객체(stream event object)가 있다. 디렉터리 객체는 그 안에 파일객체, 스트림 객체, 스트림 이벤트 객체 및 다른 디렉터리 객체를 포함할 수 있다. 파일 객체는 애플리케이션 프로그램(Xlet) 파일과 이것이 사용하는 데이터 파일을 포함한다. 스트림 객체는 TS에 삽입되어 방송되는 비디오, 오디오 또는 데이터 스트림에 대한 정보를 가지고 있도록 되어 있다. 스트림 이벤트 객체는 스트림 이벤트에 대한 정보를 가지고 있다. 즉, 스트림 이벤트 객체와 스트림 객체는 스트림 이벤트 또는 스트림 자체를 가지고 있는 것이 아니라, 이들을 TS에서 찾는데 필요한 참조정보를 가지고 있다.

객체 카루셀에 속하는 각 객체는 표준으로 정해진 형식을 가지는 메시지에 의해 전송된다. 즉 객체 카루셀을 구성하는 객체들은 디렉터리 메시지, 파일메시지, 스트림 메시지, 스트림 이벤트 메시지의 형태로 전송된다. 객체 카루셀에서 "메시지"는 해당 객체를 표현하는 구문(syntax)상의 구조를 강조하고 "객체"는 해당 메시지에 의해서 표현되는 내용을 가리킨다. 객체 카루셀을 논할 때 "객체"와 "메시지"는 상호 교환적으로 사용되곤 한다. 특히, 객체 카루셀의 객체와 Java 언어에서 말하는 객체(class로부터 생성한 instance)가 혼동을 일으킬 가능성이 있는 문맥에서는 카루셀 객체를 메시지라고 부르겠다.

디지털 방송에서는 스트림에 대한 정보는 PSI/SI 테이블에도 포함되어 있는데, 이것은 객체카루셀의 스트림 메시지가 가지고 있는 스트림에 대한 정보와 그 종류가 틀리다. 객체카루셀의 스트림 메시지가 가지고 있는 스트림 정보 중 스트림 이벤트 서술자와 NPT 참조서술자 스트림에 대한 정보 등만 필수적이고 대부분은 필수적이지 않다[3]. 객체 카루셀을 구성하는 객체들은 방송국에서 정한 빈도에 따라 반복적으로 전송된다[3][6]. 본 논문의 실험에서 사용한 카루셀 생성기의 구현에 대해서는 문헌 [11,13]을 참조하라.

3. 스트림 이벤트

스트림 이벤트는 스트림 이벤트 서술자(stream event descriptor) 의 형태로 전송된다. 스트림 이벤트 서술자는 이벤트를 정의하는 eventID, eventData, eventNPT 등을 포함하고 있다. Xlet에서 스트림 이벤트 서술자를 수신하고자 할 때는 먼저 해당 스트림 이벤트 타입(이름)을 등록해야 한다. 그래야 Xlet이 어떤 스트림 이벤트가 전송되는지 알고 이를 모니터링할 수 있기 때문이다. 스트림 이벤트를 등록하는 방법은 Xlet 구현하는 장에서 자세히 설명한다. 스트림 이벤트 서술자의 eventNPT 필드는 스트림 이벤트의 처리시각을 나타내는데, 이 값이 정의되어 있지 않으면 스케줄드 스트림 이벤트가 아니라, "do it now" 스트림 이벤트로 간주된다.

스트림 이벤트 서술자를 생성하는 방법은 문헌[11][13]을 참조하라. 본 논문에서 연동형 애플리케이션의 실험을 위해서 사용된 스트림 이벤트 서술자의 eventNPT는 다음과 같이 생성하였다. 개발자가 비디오를 디스플레이하면서 특정 이벤트(프로그램)의 시작점의 NPT를 0으로 설정한다. 그리고 그 시점의 프레임 번호를 획득한다. 스트림 이벤트를

부여하고 싶은 비디오내의 특정 시점으로 이동하여 그 시점의 프레임 번호를 획득한 후, 그 프레임번호에 해당하는 NPT를 계산한다.

4. NPT(Normal Play Time)

연동형 애플리케이션은 특정 시점에 스트림 이벤트를 처리하도록 되어 있으므로 이 특정 시점을 애플리케이션에게 알려주는 방법이 필요하다. 이를 위해 스트림 이벤트는 자신이 처리될 시각을 포함하도록 되어 있다. 이 스트림 이벤트 처리시각은 NPT(normal play time)라는 시간을 사용하여 표시하도록 되어 있다. NPT(Normal Play Time)는 "9시 뉴스"처럼 특정시각에 시작하여 특정시각에 끝나는 이벤트 (프로그램) 내에서 진행되는 시간이다. NPT는 이벤트(프로그램)의 시작 시각에 시작하여 90KHz로 프로그램 내에서 연속적으로 증가한다. NPT는 해당 이벤트(프로그램)의 일부가 편집되어 없어지거나, 중간에 광고 등의 다른 이벤트(프로그램)가 삽입되는 경우에도 그 시간 값이 변하지 않는다. 즉, NPT는 해당 이벤트가 언제 방송되는 것과 관계없이, 나중에 이벤트(프로그램)가 편집되는 것과 관계없이, 한번 설정되면 계속 유효하므로 방송제작사의 스튜디오에서 스트림 이벤트와 특정 이벤트(프로그램)를 동기화시키는데 사용하는 시간으로 적절하다.

Xlet에서 현재순간의 NPT를 얻을 수 있도록 DVB-MHP에서는 getNPT()라는 API를 정의하고 있다. 이 getNPT()가 제대로 작동하려면 방송국의 다중화기/재다중화기는 NPT 참조서술자 (NPT Reference Descriptor)를 TS에 일정 빈도로 삽입하여 전송해야 한다. NPT 참조서술자 스트림은 각 이벤트별로 하나씩 존재하는데 물론 이벤트(프로그램)와의 동기화가 필요 없으면 이벤트에 NPT를 부여하지 않는다. NPT참조서술자 스트림을 생성하는 것은 해당 이벤트에게 그 내부의 시간인 NPT를 부여하는 것을 의미한다.

5. Java의 이벤트 처리 모델

Xlet에서 스트림 이벤트는 Java의 이벤트 처리 메커니즘에 의해서 처리된다. 이벤트란 무슨 일이 일어났다는 것을 알리는 일종의 신호이다. 이벤트는 마우스의 클릭이나 움직임과 같은 외부의 사용자 액션에 의해 발생된다. 그리고 이벤트는 통신망을 통해 어떤 데이터가 도착하는 경우와 같이 시스템 내부의 액션에 의해서도 발생된다. Xlet의 경우에는 방송스트림으로부터 발생되는 다양한 종류의 이벤트가 있다. Xlet이 이들 이벤트를 처리하기 위해서는 Java의 이벤트 처리모델인 위임 이벤트 모델(Delegation Event Model)[8][9]을 따라 프로그래밍 되어야 한다.

위임 이벤트 모델은 이벤트, 소스객체, 리스터 객체로 구성되어 있다. 시스템에서 처리할 수 있는 모든 이벤트 타입들은 java.util.EventObject의 서브 클래스들이다. 하나의 소스 객체는 정해진 이벤트들에 대해서 이것이 발생하는지를 감지하고, 이벤트가 발생하면 이를 입력인자로 하여 관련 이벤트 핸들러 메소드를 호출한다. 이벤트 핸들러 메소드들을 호출하려면 이들을 구현하고 있는 이벤트 리스너 객체를 알고 있어야 한다. 이벤트 리스너 객체는 정해진 이벤트 핸들러들을 구현하고 있는 클래스의 인스턴스이다. 이벤트 핸들러들은 관련 리스너 인터페이스에 정의되어 있고, 이를 구현하는 클래스에 의해 구현된다[9]. 소스 객체가 관련 리스너 객체를 알고 있도록 하기 위해 프로그래머는 관련 리스너 객체들을 소스 객체에게 미리 등록시킨다. 이를 위해 소스 객체는 리스너 객체를 등록시키는데 사용되는 메소드를 제공하고 있다.

위에서 "클래스가 인터페이스를 구현" 한다는 말을 했는데, Java에서 말하는 인터페이스는 사용자 인터페이스의 인터페이스와 관계없고, 특정 목적을 위한 관련된 메소드들의 집합을 말한다. 이 메소드들은 메소도의 이름, 입력변수의 타입, 결과변수의 타입만 선언되어 있으며 메소드들의 실제 내용은 이 인터페이스를 "implement(구현)" 하는 클래스에 의해 채워진다. 클래스는 기본적인 기능 외에 다른 기능을 추가하고 싶을 때 원하는 메소드들이 포함되어 있는 인터페이스를 구현한다. Java에서는 이벤트 타입별로 이 타입에 속하는 이벤트들을 처리하는 핸들러 메소드들을 정의하고 있는 인터페이스가 있다.

Java에서 가장 흔한 소스객체는 버튼이나 스크롤바 같은 가시적인 컴포넌트 클래스(visual component class)이다. 컴포넌트 소스 클래스는 Focus나 Key 이벤트를 감지하여 관련 이벤트 리스너 객체의 이벤트 핸들러를 호출한다. 따라서 Component 클래스는 일반적으로 Component 클래스가 제공하는 메소드인 Listener 객체를 리스너로 등록하는데, FocusListener, Key 객체 등록을 위해 addFocusListener (FocusListener fl), addKeyListener (KeyListener kl)를 사용한다. 예를 들어, button.addFocusListener(fl)라는 명령은 FocusListener 객체인 fl을 소스 객체인 button 객체에 등록

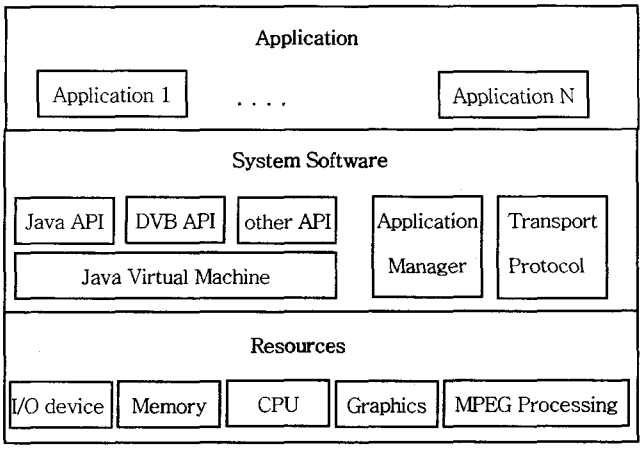


그림 1. 셋톱박스 내부의 시스템 및 프로그램 구조: Application은 미들웨어 (운영체제 위에서 돌아가는 시스템 소프트웨어) 가 구현하는 API를 이용하여 구현된다. Application 은 Application Manager가 로드 하여 실행을 시작하고 그 생명주기도 관장한다.
Fig. 1. Structure of the system and programs inside of a set-top box

시킨다. 이것은 button 객체에 Focus 이벤트가 발생했을 때 버튼 객체가 fl 리스너 객체에 의해 구현된 적절한 메소드를 호출할 수 있도록 한다.

구체적으로 KeyListener 객체는 세 개의 키 이벤트 핸들러 메소드, keyPressed(KeyEvent e), KeyReleased(KeyEvent e), KeyTyped(KeyEvent e)를 구현한다. 이 메소드들은 KeyListener 인터페이스에 의해 정의된 메소드들이다. 키 이벤트 리스너 객체의 클래스는 KeyListener 인터페이스에 속한 각 메소드가 해당 키 이벤트를 애플리케이션의 의도에 따라 적당하게 처리하도록 메소드들을 구현한다. 소스 객체인 button 컴포넌트는 실제로 발생하는 키 이벤트의 종류에 따라 세 개중의 한 핸들러 메소드를 호출한다.

6. 애플리케이션 (Xlet) 의 구조

DVB-J 애플리케이션은 DVB-MHP 수신기에 실행되는 자바 애플리케이션으로 인터넷의 자바 애플릿 (Applet)과 유사하여 Xlet이라 불린다[3]. Xlet은 비디오와 연동되어 실행되거나 비디오와 무관하게 독립적으로 실행될 수 있다. 비디오와 동기화된 애플리케이션에서는 방송국에서 어떤 일을 처리할 시각을 수신기에게 알려주고 애플리케이션이 이를 정해진 시각에 처리한다.

Xlet은 객체 카루셀에 포함되어 TS를 통해 전송된다. 객체 카루셀은 특정 채널에서 방송하는 특정 서비스의 한 요소로서 존재하므로, Xlet도 특정 서비스 안에 포함된다. 따라서 사용자가 Xlet을 선택하려면 그것이 포함되어 있는 서비스를 선택해야 한다. 이것은 셋톱박스에 내장되어 있는 EPG를 통해서 한다. Xlet이 포함된 서비스가 선택되면 Xlet중에서 "AutoStart" 라는 특성을 가진 것은 애플리케이션 매니저에 의해서 자동으로 로드 된다(그림 1, 2참조). AutoStart로 명시되지 않은 Xlet은 사용자가 직접 그 실행을 요구해야 된다. 이 요청은 EPG를 통해서 하게 되고, EPG는 애플리케이션 매니저에게 그 요청을 전달한다. 사용자가 새로운 서비스를 선택하게 되면, 현재 서비스 안에 포함되어 있던 Xlet는 소멸된다. 그런데 EPG Xlet처럼 다른 서비스로 전환이 된 이후에도 계속 실행될 필요가 있는 것은 해당 Xlet 애플리케이션의 id를 모든 서비스의

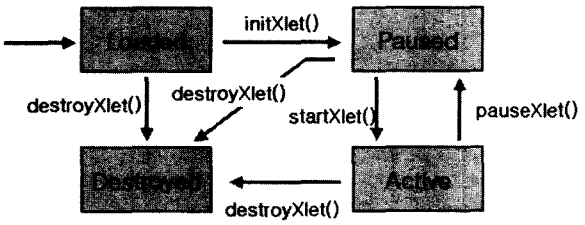


그림 2. Xlet 상태 전환표: 화살표에 표시된 메소드를 애플리케이션 매니저가 호출하면 Xlet의 상태가 바뀐다.
Fig. 2. State diagram of Xlet

AIT(Application Information Table)안에 넣어 놓으면 된다. 이런 경우, 해당 Xlet은 모든 서비스의 구성요소가 된다.

Xlet의 구동과정을 이해하려면 그림 1에 기술되어 있는 셋톱박스 내부의 시스템 소프트웨어 구조를 이해해야 된다. Xlet은 그림 1의 애플리케이션 매니저(application manager)를 통해 그 생명주기(life cycle)가 제어된다. Xlet 프로그래머는 Xlet의 생명주기를 제어하는데 필요한 메소드를 Xlet안에 정의해 놓아야 한다(그림 2). 이 메소드들은 웹 브라우저에서 실행되는 자바 애플리케이션인 애플릿(applet)의 생명주기를 제어하는 메소드인 init(), start(), pause(), destroy() 와 그 기능이 유사하다. 이 메소드들은 애플리케이션 매니저에 의해서 호출된다. initXlet(), startXlet()이 실행되면 Xlet이 실행상태에 들어가게 되고, destroyXlet()은 Xlet이 속한 DVB 서비스가 종료되는 경우와 같이 Xlet이 더 이상 실행될 필요가 없을 때 호출된다. Xlet의 생명주기는 그림 2에서 보듯이 loaded, paused, active, destroyed의 상태(state)를 가지고 있다. 그리고 각 상태간의 전환은 그림 2에서 보듯이 Xlet의 생명주기 메소드를 실행함으로써 이루어진다.

AutoStart 또는 EPG를 통해 애플리케이션 매니저가 셋톱박스 메모리내로 로드한 Xlet을 실행하기 위해 애플리케이션 매니저는 먼저 Xlet 객체의 initXlet()을 호출한다. 초기화 작업을 하는 initXlet()이 완료되면 그림 2에서 보듯이 Xlet은 paused 상태로 들어가고, 애플리케이션 매니저로 제어권이 넘어간다. 곧 이어 애플리케이션 매니저는 Xlet을 active 상태로 만들기 위해 startXlet()을 호출한다. 그림 2에서 보듯이 startXlet()은 Xlet이 active 상태에 있다가 pauseXlet() 메소드에 의해 paused 상태가 된 후 다시 실행될 수도 있다.

Xlet 프로그램이 Java로 구현되어 있고, 프로그램의 실행이 Java의 이벤트 구동(event driven)방식을 이용하기 때문에 Xlet의 진행은 해당 이벤트를 처리하는 이벤트 핸들러에 의해 이루어진다. 실제의 이벤트 처리는 startXlet()이 완료된 후 Xlet이 active 상태에 있을 때 이루어지지만, 이벤트를 처리할 이벤트 리스너를 생성하고 등록하는 작업 등 이벤트 처리에 필요한 준비작업은 그 전에 해 주어야 한다. 따라 준비작업은 Xlet이 active 상태가 되기 전 initXlet() 또는 startXlet() 메소드에 의해서 하게 되어 있다. 이 두 메소드 중 어떤 메소드에서 어떤 준비작업을 할 것인가는 이 준비작업이 Xlet 생명주기에서 한번만 해 주면 되는 일인가, 아니면 Xlet이 paused 되었다가 재 시작될 때마다 새로 해주어야 되는 일인가에 달려 있다. 단 한번만 해주어도 되는 일은 initXlet()에서 해주어야 하고, Xlet이 재 시작될 때 마다 해주어야 되는 일은 startXlet()에서 해주어야 한다.

Ⅲ. 연동형 애플리케이션 개발

1. 스트림 이벤트(Stream Event) 수신

스트림 이벤트는 스트림 이벤트 서술자(stream event descriptor) 내에 삽입되어 전송된다. 스트림 이벤트는 Java 프로그램인 Xlet의 입장에서 보면 하나의 이벤트이다. 따라서 Xlet에서 이 이벤트를 처리하려면 특정 소스 객체가 이 이벤트를 등록하는 단계가 필요하다. 특정 이벤트를 소스 객체에 등록하려면 그 이벤트 타입 이름과 그 이벤트 리스너 객체를 알아야 한다. 2.5절에서 예로 든 button.addKeyListener(KeyListener kl)의 경우, 프로그래머가 button 소소 객체에 등록할 이벤트 타입(Key 이벤트)과 리스너 객체인 kl을 알고 있어야 했다. 스트림 이벤트를 처리하는 Xlet의 경우도 같은 작업을 해주어야 한다.

DVB-MHP 규약에서 스트림 이벤트의 타입은 스트림 이벤트의 이름으로 표시된다. 따라서 만약 애플리케이션 프로그래머가 전송 스트림을 통해 전송될 스트림 이벤트들의 이름을 다 알고 있다면 적절한 소스객체를 통해 스트림 이벤트들을 등록할 수 있다. 그러나 이 방법은 애플리케이션 프로그래머에게 지나친 부담을 줄 뿐만 아니라, 애플리케이션이 특정 TS에 종속되는 결과를 가져온다. 따라서 DVB-MHP에서는 프로그래머가 특정 스트림 이벤트의 이름(eventName)을 미리 알고 있는 것이 아니라, 객체 카루셀의 스트림 이벤트 메시지를 통해서 그 정보를 획득할 수 있도록 되어 있다.

Xlet은 스트림 이벤트를 처리하기 위한 사전단계로서 스트림 이벤트 이름(eventName), ID (eventID), 스트림이벤트 서술자 스트림의 ID 등을 포함하는 스트림 이벤트 메시지를 객체 카루셀로부터 획득하고, 획득한 모든 스트림 이벤트 이름들을 특정 소스 객체를 이용하여 등록한다[10]. 이렇게 하면 프로그래머가 스트림 이벤트 타입(이름)에 대해서 미리 알 필요가 없다. 그 다음 TV 미들웨어가 TS로부터 스트림이벤트 서술자 스트림을 읽어 해당 소스 객체

로 하여금 스트림 이벤트가 발생했음을 인지하게 한다. 소스객체는 이 스트림 이벤트를 입력인자로 하여 미리 정의되어 있는 이벤트 핸들러 메소드를 호출한다. 스트림이벤트 서술자가 가지고 있는 ID 는 Tap이라고 부르는데, Tap은 스트림의 타입(스트림 이벤트 서술자, NPT참조서술자, 또는 일반 비디오 스트림 등)과 그 스트림의 associationTag를 가지고 있다. associationTag는 스트림이 생성될 때 부여되는 id인데, 이 id에 해당되는 PID는 전송스트림이 다중화될 때 결정되며 associationTag과 PID의 관계는 PMT에 기록된다. 그러나 애플리케이션 프로그래머는 이러한 과정을 알 필요는 없고, 스트림 이벤트 메시지에서 해당 스트림 이벤트 서술자 스트림을 가리키는 Tap를 가지고 있다는 것 정도만 알고 있으면 된다.

이렇게 스트림 이벤트를 보낼 때 먼저 스트림 이벤트 메시지를 보내는 것은 수신기로 하여금 앞으로 발생할 수 있는 스트림 이벤트를 예측하고 기다릴 수 있게 한다. 하나의 스트림 이벤트 메시지는 Xlet으로 하여금 그 스트림 이벤트 메시지에 포함된 스트림 이벤트 ID와 같은 ID를 가지면서 다른 시점에 발생하는 다수의 스트림 이벤트들을 처리할 수 있게 한다[7][10].

2. Xlet 에서의 스트림 이벤트 처리 구현

본 Xlet 애플리케이션은 스트림 이벤트 서술자의 구조로 MPEG-2 TS에 포함되어 전송되는 스트림 이벤트를 받아서 이를 스트림 이벤트 서술자내에 명시된 시각(eventNPT)에 처리하는 일을 수행한다. 그림 3의 약식 코드에서 보듯이, Xlet 애플리케이션은 먼저 스트림 이벤트 메시지의 경로명 "StreamEvents"을 이용하여 DSMCCObject 클래스의 객체 dsmobj를 생성한다(라인 1). 여기서 DSMCCObject 는 객체 카루셀의 객체(메시지)들을 읽고 처리하는데 사용되는 Java 클래스이고 이의 인스턴스 dsmobj를 Java에서 객체라고 하는데, 이 객체는 객체 카루셀에서 말하는 객체와 관계없다. 대부분 문맥에 의해 어떤 객체를 말하는지 알 수 있으나 혼돈을 피하기 위해 객체 카루셀이 객체는 메시지라고 칭한다.

스트림 이벤트 메시지를 로딩(라인 3) 한 후, 스트림 이벤트 메시지의 내용을 읽는데 필요한 DSMCCStreamEvent 객체 dsmSE을 dsmobj를 이용하여 생성한다(라인 4). dsmSE 객체는 스트림 이벤트 메시지 안에 포함되어 있는 이벤트 이름들의 리스트를 가져온다(라인 5). 그리고

```
public class StreamEventXlet extends HComponent
implements Xlet, StreamEventListener, Timer {
...
 public void startXlet() {
(1) DSMCCObject dsmobj = new DSMCCObject("StreamEvents");
(2) DSMCCStream dsmStr = new DSMCCStream(dsmobj);
(3) dsmobj.synchronousLoad();
(4) DSMCCStreamEvent dsmSE
                = new DSMCCStreamEvent(dsmobj);
(5) String eventList[] = dsmSE.getEventList();
(6) for(int i = 0; i < eventList.length; i++)
      dsmSE.subscribe(eventList[i], this);
     /* this는  StreamEventListener 인터페이스를 구현하고 있는
       StreamEventXlet 클래스의 인스턴스임 */
(7) Let the timer source object register "this" object as a listener
to the timer event.
}
 (8 ) public synchronized void
                receiveStreamEvent(StreamEvent ste)
    { static short I =0; /* set to 0 only once initially */
      byte eventData[] = ste.getEventData();
      eventNPT = ste.getEventNPT(); ...
      if (npt = null) processEvent(eventData);
      else { i++; streamEvents[i].eventNPT = eventNPT;
            streamEvents[i].eventData= eventData;
            streamEvents.number = i;
      }
    }
 (9) public synchronized void onTick()
    { for (i=1; I <= streamEvents[i].number; i++) {
       currentNPT= dsmStr.getNPT();
       if ( currentNPT = streamEvents[i].eventNPT )
                        processEvent(streamEvents[i]);
    }
}
```

그림 3. 스트림 이벤트 처리 Xlet의 약식코드
Fig. 3. Pseudo code of the Xlet

dsmSE 객체는 각 이벤트 이름에 대해 이의 리스너 객체를 dsmSE객체에 등록한다(라인 6). 이를 Java의 위임 이벤트 처리 모델의 관점에서 보면 DSMCCStreamEvent 클래스의 dsmSE 객체는 스트림 이벤트의 소스객체 역할을 한다. 소스 객체는 이벤트의 리스너 객체를 자신에게 등록하고, 해당 이벤트가 발생하면 리스너 객체가 구현하고 있는 이벤트 핸들러 메소드를 호출한다. 이 경우 리스너 객체의 클래스는 org.dvb.dsmcc.StreamEventListener 인터페이스를 구현하고 있는 클래스인데, 그림 3의 약식코드에서 보듯이 이

것은 바로 StreamEventXlet 클래스이다. 그리고 그 클래스의 인스턴스인 "this" 가 스트림 이벤트의 리스너 객체이다. StreamEventXlet 클래스는 StreamEventListener 인터페이스의 메소드인 receiveStreamEvent()를 정의함으로써 해당 인터페이스를 구현한다(라인 8).

스트림이벤트 서술자를 포함하는 TS 패킷이 수신되면 소스객체인 dsmSE 객체는 이를 인지하여 이벤트 객체인 StreamEvent 객체를 생성한다. 그리고 이를 입력인자로 하여 이벤트 핸들러인 receiveStreamEvent() 메소드를 호출한다. 이 이벤트 핸들러는 StreamEvent 클래스의 객체의 메소드인 getEventNPT()를 이용하여 eventNPT를 구하며 getEventData()를 이용하여 eventData를 얻는다(라인 8의 메소드 내부 참조). 그리고 스트림 이벤트가 "do it now" 스트림 이벤트인지 (getEventNPT()가 null 인 경우) 스케줄드 스트림 이벤트인지를 구분한 후, 전자이면 바로 eventData를 이용하여 처리하고 후자이면 그 정보를 저장만 한다(라인 8의 메소드내부 참조). 왜냐하면 스케줄드 스트림 이벤트는 처리할 시각 (eventNPT) 보다 미리 전송되며, 현재 시각 (currentNPT)이 처리시각과 같을 때 비로소 처리를 해야 되기 때문이다.

그림 3의 약식코드의 라인 9는 onTick()이라는 타이머 이벤트 핸들러를 구현하고 있다. 이 이벤트 핸들러는 라인 8 의 메소드 내에서 저장한 스트림 이벤트를 처리할 것인지를 체크하고 필요하면 처리한다. onTick 메소드는 시간이 정해진 단위시간만큼 흐를 때 마다 호출된다. 그림 3의 약식코드는 타이머 이벤트와 관련하여 세 가지를 기술하고 있다. 첫째는 StreamEventXlet 클래스가 onTick 메소드를 정의하는 인터페이스 Timer를 구현한다는 것이다 (그림 3의 첫줄). 둘째는 타이머 소스 객체로 하여금 StreamEventXlet 클래스의 인스턴스인 "this" 객체를 타이머 이벤트의 리스너 객체로 등록하는 것이다 (라인 7). 셋째는 타이머 이벤트 핸들러인 onTick 메소드를 구현하는 것이다(라인 9). 타이머 이벤트 핸들러는 실행될 때 마다 현재 순간의 NPT (currentNPT)를 DSMCCStream 객체 dsmStr(라인 2에서 생성) 의 getNPT() 메소드를 이용하여 구한 다음, 이것이 저장된 스트림 이벤트의 처리시각 (eventNPT) 와 같은지를 체크하고 같으면 해당 이벤트를 처리한다.

라인 9에서 currentNPT를 DSMCCStream 객체의 메소드인 getNPT() 사용해서 구한 것으로 기술을 했다. 그러나 실제 구현에서는 이 메소드를 사용하지 않았다. 그것은 아직 상용 스트림 생성기와 셋톱 미들웨어가 NPT를 사용할 수 있도록 되어 있지 않기 때문이다. getNPT() 메소드는 구현되었다고 리포트 되고 있으나, 아직 NPT 참조 서술자 스트림을 생성하는 상용도구 등의 미비로 이 메소드가 충분히 테스트되어 있지 않다. 따라서 본 논문에서는 본 연구팀이 구현한 근사적인 NPT 재구성 방법[12]을 사용하였다.

Ⅳ. 실험 및 결론

본 논문에서 본 논문에서 기술한 방법을 따라 실제로 연동형 애플리케이션을 구현하여 실험하였다. 구현된 연동형 애플리케이션을 검증하기위해 스트림 이벤트가 포함된 전송스트림을 생성한 후 이를 셋톱박스에 전송하여 정해진 시각에 스트림 이벤트가 처리됨을 확인하였다. 실험의 구체적인 방법은 다음과 같다. 애플리케이션 제작자가 스트림 이벤트를 동기화 시킬 비디오를 플레이하면서 특정 시점을 선택하여 스트림 이벤트의 시점을 정한다. 이 시점에 특정 그래픽 이미지가 보이도록 "scheduled" 스트림 이벤트가 포함된 TS를 생성한다. 구현한 연동형 애플리케이션에서 스트림 이벤트의 eventData는 이미지파일 이름으로서 실제로 스트림 이벤트를 처리할 때는 그 이름을 가진 이미지 파일을 객체 카루셀로부터 읽어 해당 이미지를 디스플레이 하도록 되어 있다. 스트림 이벤트가 포함된 TS는 본 연구팀에서 개발한 스트림 생성기를 사용하였다[11][13]. 동기화가 성공적으로 되는지를 검증하기 위해, 셋톱박스로 전송되어 플레이되는 방송의 화면을 캡쳐하였다. 그리고 캡쳐된 동영상을 초당 25프레임단위로 나누어 프레임별로 점검하면서 그래픽 이미지가 실제로 디스플레이 되는 시점이 스트림 이벤트를 전송스트림에 삽입할 때 제작자가 명시한 시점과 일치하는지를 확인했다. 실험결과 명시된 모든 지점에서 240ms 이내에서 비디오와 그래픽 이미지가 동기화됨을 확인하였다. 동기화 오차범위 240ms는 실험적인 결과로서 그래픽 이미지와 비디오와 동기화에서 그래픽 이미지가 원래 의도된 시점과 240ms 어긋나는 시점에 디스플레이 되어도 사람이 그 차이를 인식하지 못한다는 것이다. 이 실험결과는 한 연구논문[14]의 결과에 불과하나, 하나의 지표로 사용할 수는 있다고 본다.

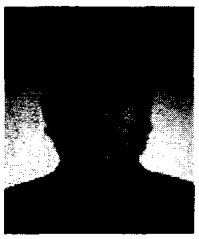
본 논문은 디지털 방송의 새로운 기능인 연동형 애플리케이션을 DB-MHP 환경에서 구현하는 방법을 기술하고, 실제로 구현하여 실험을 했다. 특히 스케줄드 스트림 이벤트를 이용하여 연동형 애플리케이션을 구현했다. 현재 상용

전송스트림 생성기에서는 스케줄드 스트림 이벤트를 지원하는 전송스트림을 생성해주는 기능이 없다. 따라서 본 논문에서 실험적으로 연동형 애플리케이션을 구현하고, 그 구현방법을 기술한 것은 연동형 애플리케이션에 관심이 있는 사람들에게 의미 있는 일이다.

참 고 문 헌

[1] ATSC. http://www.atsc.org ,2003.
[2] OCAP. http://www.opencable.com ,2003.
[3] ETSI TS 101 802 : Digital Video Broadcasting-Multimedia Home Platform 1.0.1, 2001.
[4] Richard Chernock, "Data Broadcasting", McGraw-Hill, 2001.
[5] ISO/IEC 13818-1 Generic Coding of Moving Picture and Associated Audio : Systems
[6] ISO/IEC 13818-6 Generic Coding of Moving Picture and Associated Audio: Digital Storage Media Command and Control
[7] TR 101 202 DVB: Implementation guidelines for Data Broadcasting, 1999.
[8] Sun Microsystems, Java AWT: Delegation Event Model, http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html , 2002.
[9] Y.D. Liang, "Introduction to Java Programming", Que E&T 1999
[10] Steven Morris. The Interactive TV Web. http://www.mhp-interactive.org/tutorial 2002.
[11] 김세훈, 정문열, 연동형 데이터 방송 애플리케이션을 위한 스트림 생성기 개발, 2003 년 한국방송공학회 학술대회, pp.267-270, 2003.
[12] 정문열, 김용한, 백두원, 동기화된 데이터방송을 위한 근사적인 NPT 재구성 기법, Technical Report, 서강대학교 영상대학원 미디어공학과, 2003. (방송공학회 논문지 제출)
[13] 정문열, 김세훈, 백두원, 연동형 데이터 방송 애플리케이션을 위한 스트림 생성기, Technical Report, 서강대학교 영상대학원 미디어공학과, 2003. (방송공학회 논문지 제출)
[14] Ralf Steinmetz, "Human Perception of Jitter and Media Synchronization", IEEE Journal on Selected Areas in Communications, vol. 14, no. 1, Jan. 1996, pp. 61-72.

저 자 소 개



정 문 열

- 서울대학교 계산통계학과 졸업 (BS)
- 카이스트 전산학과 졸업 (MS)
- University of Pennsylvania 전산학과 졸업 (Ph. D.)
- 일본 큐슈공과대학 정보공학부 조교수 역임
- 숭실대학교 컴퓨터학부 부교수 역임
- 현 서강대학교 영상대학원 미디어공학과 부교수
- 주관심분야 : 컴퓨터 그래픽스 인터랙티브 방송

백 두 원

- 서울대학교 수학과 졸업 (BS)
- University of Minnesota 전산학과 졸업 (MS, Ph. D.)
- AT&T Bell Lab., Member of Technical Staff
- Cadence Design System, Member of Consulting Staff
- 현 숭실대학교 미디어학부 부교수
- 주관심분야 : 디지털방송 컴퓨터 그래픽스 DRM 알고리듬