□신기술해설□

멀티플랫폼 실시간 분산 멀티미디어 시스템 MuX-II

김 두 현[†] 황 승 구[††] 박 치 항[†††]

◈ 목 차 ◈

요 약

네트워크에 의한 실시간 멀티미디어 서비스의 필요성이 인터넷의 확산과 아울러 증대되어 가고 있다. 본 고에서는 멀티플랫폼 상에서 실시간 분산 멀티미디어 스트림 서비스를 제공하는 MuX-II 시스템(Multimedia Crossing System II)의 개념과 구조 그리고 실제 프로그램 예제를 설명한다. MuX-II의 근간을 이루는 MuX 멀티미디어 처리 모델에는 스트림 층, 프리젠테이션 층, 하이퍼프리젠테이션 층이 있으나 본 고에서는 이 중에서 스트림 층의 처리 요소와 처리 방식에 중점을 두어 필터, 원점(Source) 객체, 종점(Destination) 객체, 합성기(Mixer), 분배기(Copier), 접합기(Weaver), 분리기(Unweaver), 동기 객체 등에 대하여 설명한다. 아울러 이러한 객체의 멀티쓰레드 처리를 위한 동적 객체의 상속 체계와 스트림 연결 방식, 그리고 동기화 방식에 대하여도 소개한다.

1. 서 론

네트워크에 의한 실시간 멀티미디어 서비스의 필요성이 인터넷의 확산과 아울러 증대되어 가고 있다. 본 고에서는 멀티플랫폼 상에서 실시간 분산 멀티미디어 스트림 서비스를 제공하는 MuX-II 시스템[1](Multimedia Crossing System II)의 개념과 구조 그리고 실제 프로그램 예제를 설명한다.

MuX-II 시스템이 목표로 하는 계산 환경은 <그림 1>과 같이 멀티미디어 정보 통신망 상에서 분산

† 정 회 원 한국전자통신연구원 책임연구원
†† 정 회 원 : 한국전자통신연구원 컴퓨터연구단 멀티미디어연구부장
††† 종신회원 : 한국전자통신연구원 컴퓨터연구단장

[1] MuX-II는 한국전자통신연구원에서 윈도우 NT용인 MuX V1R3와 솔라리스용인 MuXV2R1에 이어 현재 개발 중에 있다.

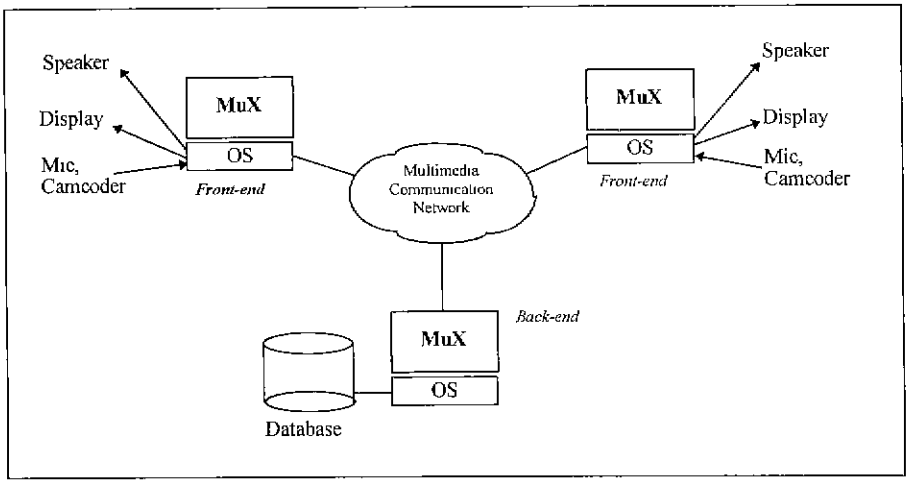
되어 수행되는 멀티미디어 관련 응용 프로그램 또는 멀티미디어 서비스를 위하여 각 사용자의 가정이나 직장 또는 품안에 멀티미디어 정보 통신용 단말기가 존재하고 멀티미디어 정보 통신 서비스를 위한 서버급 컴퓨터가 망 상에서 존재하는 멀티미디어 분산 환경이다.

MuX-II 시스템은 이러한 분산 환경에서 멀티미디어 회의, 원격 교육, 원격 프리젠테이션, 하이퍼미디어 정보 검색, 멀티미디어 메일 등의 멀티미디어 통신 서비스를 위하여 사용자가 소유한 각 단말기나 서비스 서버 상에서 멀티미디어 입출력을 관리하고 멀티미디어 정보를 실시간에 처리하는 Front-End 및 Back-End 멀티미디어 서버를 위한 기본 계산 방식을 제시하는 것이다.

서 스트림 층의 처리 요소와 처리 방식에 중점을 두어, 스트림 층의 기본 처리 요소인 필터, 원점 객체, 종점 객체, 합성기, 분배기, 접합기, 분리기, 동기 객체 등에 대하여 설명한다.

2.1 미디어와 프레임

본 고에서 취급하는 미디어는 주로 컴퓨터 영상회의에서 사용되는 실시간 미디어인 비디오와 오디오로 제한한다. 프레임은 시간축 상에서 임의 시간간격에 해당되는 미디어 데이터를 포함하고 있는 구조체로서, 미디어의 종류나 임의 시점의 QoS(Quality of Service)에 따라 그 크기가 가변적일 수도 있다. 본 논문에서 논하는 프레임은 입출력시 사용자가 느끼는 감각적 단위와 다를 수도



(그림 1) MuX 모델이 목표로 하는 분산 멀티미디어 환경

2. MuX 멀티미디어 처리 모델

MuX-II의 근간을 이루는 MuX[Kim92, Ren92, Bak92, Kim95a, Kim95b, Kim96]의 멀티미디어 처리 모델에는 스트림 층, 프리젠테이션 층, 하이퍼프리젠테이션 층이 있으나 본 고에서는 이 중에

있다. 예를 들어 비디오의 경우 한 프레임을 사용자가 시각적으로 느끼는 정지화면 한장을 한 프레임으로 삼을 수 있으나, 압축방법에 따라서는 그렇지 않을 수도 있다. 또한 오디오의 경우에는 기본적으로 프레임 개념이 없기 때문에 가상적인 프레임을 만들어야 할 경우도 있다. 따라서 프레

임은 연속 미디어의 컴퓨터 처리를 위하여 사용하는 내부 구조체이다.

모든 프레임은 입력단말장치에 의하여 부여된 시간표지 (Time Tag)를 갖고 있다고 가정한다. 시간표지에 기록된 시각은 SMPTE[2] 코드 등 입력장치의 실시간일 수도 있고 일련번호를 나타내는 정수일 수도 있으나 이들은 프레임률과 시작시간만 알고 있으면 서로 변환이 가능한 것이기 때문에 본 모델에서는 프레임률과 시작시간을 알고있다는 가정하에 일련번호를 나타내는 정수 시간표지를 사용한다.

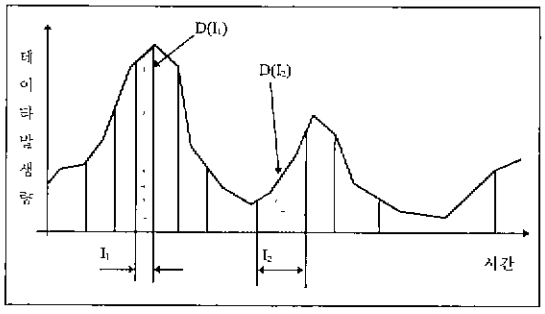
2.1.1 주기적 연속 미디어

주기적 연속 미디어(Periodic Continuous Media)의 특징은 일정량 또는 가변량의 데이터가 일정한 주기로 발생된다는 점이다. 이러한 미디어는 각종 디지타이저에 의하여 일정한 프레임율로 샘플링되는 경우 또는 임의 데이터 모델에 따라 합성 또는 생성되어지는 경우에 속한다. 스트림은 주기적 연속 미디어의 흐름을 실현함에 있어서 데이터 발생 당시 부터 내재되어 있는 주기를 최대한 유지시켜 주어야 한다.

2.1.2 부분주기적 연속 미디어

부분주기적 연속 미디어(Semi-Periodic Continuous Media)의 특징은 주기적 연속 미디어와 같이 일정량 또는 가변량의 데이터가 지속적으로 발생되기는 하나 그 주기가 약간 이상의 가변성을 갖는 경우이다. 이러한 미디어는 주로 차분적코딩(Differential Coding)을 하는 경우 코덱 장치가 데이터를 발생시키는 시간 간격이 일정하지 않은 경우에 속한다. 스트림은 부분주기적 연속 미디어의 흐름을 실현함에 있어서 데이터 발생 당시 부터 내재되어 있는 데이터 간의 시간 간격을 최대한 유지시켜 주어야 한다.

현재 MuX-II 서버는 윈도즈 NT 3.5에서 H.261 [H261] 압축형식의 비디오와 8 Bit Sample Size, 11.04Khz Sampling Rate의 PCM[G711] 형식의 오디오를 사용하고 있다. 따라서 비디오는 부분주기적 연속 미디어이고 오디오는 주기적 연속 미디어이다. H.261은 기본적으로 차분정보를 이용한 압축방식이므로 데이터 발생량이 시간에 따라 가변적이며, H.261 압축 디바이스가 압축한 데이터를 매번 메모리로 전달하는 크기가 고정되어 있으므로 압축 디바이스가 중앙처리장치에 인터럽트를 보내주는 주기도 가변적이 된다. 이러한 상황이 그림 2에 예시되어 있다. 즉 그림에서 나타냈듯이 시간 간격 동안의 데이터 발생량을라 할 때 인 반면 이다. 한편 PCM 방식의 오디오는 차분정보를 이용하지 않으므로 임과 아울러 이 된다.



(그림 2) 부분주기적 연속 미디어의 데이터 발생 예시

2.1.3 이벤트형 미디어

이벤트형 미디어의 특징은 주로 마우스 입력과 같이 비주기적이면서 불연속적이라는 점이다. 엄밀히 말해서 이벤트형 미디어는 멀티미디어 본연의 데이터는 아니나 멀티미디어와 함께 사용되는 데이터이므로 스트림이 처리하여야 할 대상에 속한다. 스트림은 이벤트형 미디어를 처리함에 있어서 이벤트 처리의 실시간성을 최대한 만족시켜 주어야 한다.

[2] Society of Motion Picture and Television Engineers

2.2 스트림

스트림은 단일 미디어의 흐름으로서 미디어 데이터 입출력의 가장 기본적인 구조이다. 스트림이 처리하는 미디어의 종류는 이상의 세가지 유형이 있을 수 있으며, 미디어 스트림은 이들 미디어에 대하여 화일, 디바이스, 네트워크, IPC, 분배기(Copier), 접합기(Weaver), 분리기(Unweaver), 합성기(Mixer) 사이의 입출력 관계를 맺고 그 입출력 기능을 수행하여 미디어의 흐름이 형성되도록 한다. 스트림은 상기의 각종 미디어의 흐름을 실현함에 있어서 그 성능면에서 연속 미디어에 대해서는 연속성을 보장함과 아울러 이벤트형 미디어에 대해서는 실시간성을 최대한 보장한다.

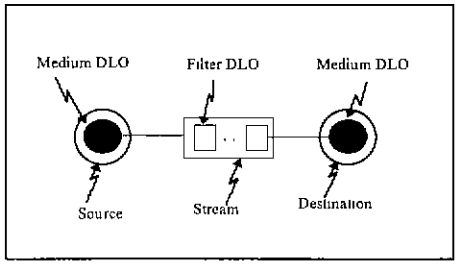
2.2.1 스트림 기능

스트림이 미디어의 흐름을 실현하기 위하여 세부적으로 다음과 같은 기능을 제공한다.

- 화일, 디바이스, 통신, IPC(Inter-Process Communication)와의 입출력 서비스
- Mixer, Weaver, Copier, Unweaver와의 입출력 서비스
- 스트림 필터링 서비스
- 다른 스트림과의 동기화를 위해 스트림으로 들어오는 데이터의 시표(Time Tag) 표시

2.2.2 스트림 처리 요소

스트림은 상기의 서비스들을 제공하기 위하여 Stream, Source, Destination, Filter, Medium 등의 요소가 필요하며 이들의 관계가 그림 3에 나타나 있다.



(그림 3) 스트림 개념도

2.2.2.1 Medium

Medium 객체는 각종 멀티미디어 디바이스, 화일 포맷, 통신 프로토콜과의 입출력을 일관성 있게 하기 위하여 사용되는 객체로서 DLO(Dynamic Linking Object)[Jun93]로 구현되어 진다. 미디엄 DLO는 각종 포맷의 멀티미디어 화일 및 멀티미디어 디바이스와의 인터페이스를 위한 DLO와, 통신을 위한 DLO 등이 있는데, 대표적인 예로 JPEG 비디오 화일 입출력 DLO, H.261 비디오 입출력 DLO, UDP 통신용 DLO, TCP 통신용 DLO, Multicast 통신용 DLO 등을 들 수 있다.

2.2.2.2. Source 및 Destination 객체

Source 객체는 입력용 Medium을 소유하고 있는 객체로서 스트림에 흐르는 단위 데이터인 프레임의 시발점이 된다. 따라서 Source 객체에서는 Source 객체가 소유한 DLO를 통하여 각종 입력 장치로부터 프레임 단위의 정보가 읽혀 들여진다. 따라서 Source 객체는 Medium 에서 프레임을 읽어 Destination으로 전달하는 주기적 동작을 위한 쓰레드의 출발점이며, Source 객체는 프레임의 흐름을 제어하기 위한 Play, Stop, Pause, Jump, SetFrameRate, SetSpeed, SetDuration 등의 함수를 제공한다.

Destination 객체는 출력용 Medium 객체를 소유하고 있는 객체로서 Source에서 생성된 프레임의 종착점이 된다. 따라서 프레임 구조체를 위하여 할당된 메모리를 돌려준다.

2.2.2.3 Stream과 필터 객체

Stream 객체는 그림 3과 같이 Source 객체와 Destination 객체의 연결로서, Source 객체가 Medium 객체로부터 데이터를 읽어들여 생성한 프레임을 Destination 객체가 소유하고 있는 출력용 Medium 객체로 전달하는 기능을 수행한다. 특히 Stream 객체는 이러한 전달의 과정에 있어서 자신에게

등록된 Filter 객체에 매 프레임을 적용함으로써 Stream 객체를 통과하는 프레임에 적절한 변화를 줄 수 있도록 해준다.

필터는 DLO로 구현되는 데, Stream 객체가 생성된 후 응용 프로그래머가 원하는 필터의 DLO 명칭을 Stream 객체에 API를 통하여 지정할 수 있다.

2.3 Channel (채널)

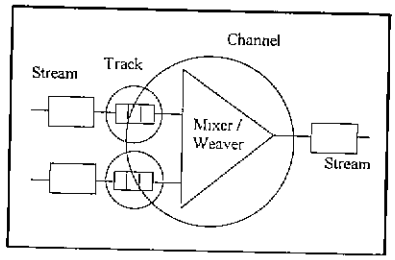
스트림이 미디어 입출력의 가장 기본적인 독립 구조라 한다면, 채널은 여러개의 독립적인 스트림을 하나의 스트림으로 합치는 작업을 한다.

2.3.1 채널의 기능

- 스트림 시작 시점 및 종료 시점 동기화 (End-point Synchronization): Channel 객체에 등록된 Track이나 Source에 지정된 시작 시점과 종료 시점에 근거하여 Channel 자체의 시간 축 상에서 해당 시점에 시작 및 종료를 시킨다.
- 동종 미디어 스트림의 통합: 예를 들어, 여러 비디오 신호를 혼합하여 하나의 그래픽 표면에 매핑하거나, 다중 오디오 스트림을 하나의 스트림으로 혼합하는 등의 기능을 수행한다.
- 이종 미디어 스트림의 접합: 예를 들어, 오디오 스트림과 비디오 스트림을 접합(Interleave)하여 단일 스트림으로 만들어 주는 기능을 수행한다.
- 스트림 간의 동기화(Interstream Synchronization): 상기 항목의 통합 및 접합 기능을 수행하면서 프레임에 부착된 시간표지를 이용하여 프레임 간의 동기화를 시킨다.

2.3.2 채널 처리 요소

채널은 상기의 서비스들을 제공하기 위하여 트랙 객체, 채널 객체, 합성기(Mixer), 접합기(Weaver) 등의 요소가 필요하며 이들의 관계가 그림 4에 나타나 있다.



(그림 4) 스트림 합성 및 접합 개념도

2.3.2.1 트랙 객체

트랙 객체는 일종의 Destination 객체로서 Stream 객체를 통하여 전달되어 오는 프레임을 큐에 입력하고, 채널의 요구에 따라 큐로 부터 프레임을 인출하여 주는 역할을 한다. 트랙 객체가 사용하는 큐는 프레임에 부착된 시간표지(Time Tag)를 이용하는 시간큐(Time-based Queue)로서 채널이 Dequeue시에 채널 객체의 시점을 매개변수로 넘겨주면 시간큐는 그 시간에 가장 근사한 시간 표지를 갖는 프레임을 Dequeue시키고 그 이전시간의 프레임들을 소멸시킨다. 이러한 시간큐의 기능에 의해서 채널 객체가 프레임의 합성이나 접합시에 동기화 기능까지 수행 할 수 있게 한다.

2.3.2.2. 채널 객체

채널 객체는 일종의 Source 객체로서 자신에게 접속된 트랙 객체들로 부터 프레임을 인출하여 이들을 합성기(Mixer) 또는 접합기(Weaver)를 이용하여 합성 또는 접합하여 출력측 Stream 객체에게 전달하는 기능을 한다. 합성기나 접합기는 DLO로 구현되어 있는 데 채널의 생성시 매개변수로 지정하게 되어 있다. 채널의 생성시 어느 DLO를 지정하느냐에 따라 트랙 객체로 부터 인

출한 프레임들의 데이터 자체를 믹싱하여 한 프레임으로 만드는 합성을 할 수도 있고, 프레임들을 단지 인터리브시키는 접합을 할 수도 있다. 믹싱을 시키는 경우는 주로 음악과 음성 등과 같이 동종 미디어의 스트림이 트랙을 입력으로 받는 경우이고, 인터리브시키는 경우는 주로 비디오와 오디오 등 이종 미디어일지라도 동기정보를 유지한채 네트워크로 전송하기 위한 경우에 주로 사용한다. 또한 채널 객체가 수행 중이라도 트랙 객체의 접속이 가능하다. 즉 예를 들어, 오디오 스트림 하나와 음성 스트림 하나를 믹싱하고 있는 도중에라도 세번째 오디오 스트림이 접속되는 것이 가능한데 세번째 스트림이 접속되면 접속되는 순간부터 세개의 스트림이 합성되게 된다.

2.3.2.3 합성기(Mixer)

합성기의 주요 기능은 각 트랙으로 부터 채널 시점에 가장 근사한 시간지표를 갖는 프레임을 인출하여 하나의 합성된 프레임을 생성하고 이를 출력 스트림으로 출력시키는 기능을 한다. 또한 합성기 DLO을 만들기 여부에 따라서는 몇 개의 트랙들이 합성되어 내부적인 스트림을 형성하고 이러한 내부 스트림 몇 개가 다시 합성되어 새로운 스트림을 형성하는 등 계층적인 형태도 가능하다.

2.3.2.4 접합기(Weaver)

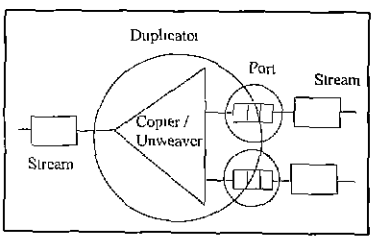
스트림 통합과 스트림간 동기화는 합성기에 의하여 주로 이루어 지지만 때로는 합성을 하지 않고 동기를 맞추어 단순히 접합만을 하여 인터리브 시켜야할 경우가 있다. 이를 위한 것이 접합기이다. 접합가는 합성기와 마찬가지로 각 트랙으로 부터 채널 시점에 가장 근사한 시간지표를 갖는 프레임을 인출하여 복합적인 구조의 하나의 접합 프레임을 생성하고 이를 출력 스트림으로 출력시키는 기능을 수행 한다. 이렇게 만들어진 복합 프레임은 Weaver의 짝인 Unweaver에 의하여 다시 단위의 프레임으로 분리된다. Unweaver는 Duplicator를 통하여 사용할 수 있다.

2.4 DUPLICATOR

Duplicator는 채널 객체와 대칭되는 개념으로써 그림 5와 같이 하나의 스트림을 여러개의 스트림으로 복제하거나 접합되어 있는 프레임을 분리하는 기능을 한다.

2.4.1 Duplicator의 기능

- 입력 스트림을 통하여 전달되어 오는 프레임을 하나 이상의 출력 스트림으로 복제하여 분배하는 기능
- 입력 스트림을 통하여 전달되어 오는 프레임이 채널의 접합기(Weaver)에 의하여 여러개의 프레임을 내포하고 있는 경우 이를 분리하여 각 해당 출력 스트림으로 분리하는 기능



(그림 5) 스트림 분리(Unweave) 및 복제(Copy) 개념도

2.4.2 Duplicator 처리 요소

2.4.2.1 Duplicator 객체

Duplicator 객체는 일종의 Destination 객체로 스트림 객체을 통하여 전달되어 오는 프레임을 복제기(Copier)나 분리기(Unweaver)를 통하여 여러개의 프레임으로 복제 또는 분리하여 이들을 자신에게 접속된 포트 객체에게 전달해 주는 기능을 한다. 복제기나 분리기는 DLO로 구현되어 있는데 Duplicator의 생성자를 호출할 때 반드시 DLO의

이름을 매개변수로 제공하도록 되어 있다. 어느 DLO를 지정하느냐에 따라 프레임을 복제할 수도 있고 인터리브된 프레임을 분리할 수도 있다.

2.4.2.2 포트(Port) 객체

포트 객체는 일종의 Source 객체로서 Duplicator 객체로 부터 프레임을 전달받아 이를 자신의 내부에 갖고 있는 큐에 넣고, 출력 스트림이 큐로부터 프레임을 인출해 갈 수 있도록 하는 기능을 한다. 포트 객체는 Duplicator객체가 수행 중이라도 Duplicator 객체에 접속이 가능하다. 단, 수행 중 접속된 포트에는 복제 또는 분리된 프레임이 접속된 순간부터 포트 객체로 출력되기 시작한다.

2.4.2.3 복제기(Copier)

복제기는 하나의 입력 스트림을 다중의 목적지에 보내는 것이다. 예를 들어, 비디오 디바이스로부터 생성되는 YUV 입력 스트림을 디스플레이하기 위해 RGB 비디오로 변환된 스트림으로 출력되어질 수 있고, 원거리에 위치한 곳으로 보내기 위해 비디오 스트림을 MPEG과 같은 압축된 스트림으로 출력할 수 도 있다.

2.4.2.4 분리기(Unweaver)

스트림 분리기는 입력되는 스트림이 하나 이상의 미디어를 접합(Interleaving)하고 있을 경우 각 접합된 미디어들을 분리하여 각각을 해당 출력측 스트림으로 출력한다. 예를 들어, 영상회의시 사정에 의하여 PCM 방식의 오디오와 H.261의 비디오가 채널 객체의 접합기 기능에 의하여 접합되어 전송되어 오는 경우 이를 분리하여 오디오와 비디오를 각 해당 목적지인 오디오 및 비디오 디바이스로 전달해 주는 것이 가능하다.

2.5 멀티미디어 프리젠테이션 (Presentation)

멀티미디어 프리젠테이션은 각종 미디어의 스트림과 채널을 동일한 시간축 상에서 시간의 흐름을 따라서 출몰하도록 제어하는 것을 말한다. 이러한 기능을 위하여 MuX-II는 Presentation 객체를 제공한다. Presenation 객체는 Source 객체, Channel 객체, 또는 DuplicatorPort 객체들을 하나의 시간축 상에 정렬하여 각 객체의 시작과 종료를 일괄적으로 제어받도록 한다. Presentation 객체는 스트림이나 채널의 시간적 정렬을 위하여 Source 객체나 채널 객체를 등록하게 하고 각 등록된 요소들이 Presentatin 객체의 시간축 위에서 시작 및 종료 시점을 지정하도록 한다. 스트림의 경우 Source 객체가 능동적인 객체이기 때문에 Presentation 객체에는 해당 스트림의 Source 객체가 등록된다.

3. 멀티플렛폼 클라이언트 서버 구조

3.1 클라이언트 서버 모델

분산 환경에서 멀티미디어 자원을 네트워크에 투명하게 사용할 수 있도록 하기 위한 방안으로는 MuX-II를 단지 라이브러리로 제공하는 방법과 서버 프로세스로 제공하는 방법이 있을 수 있겠다. 이 둘 중 현재의 MuX-II 시스템은 그림 6에서 예시한 것과 같이 후자인 클라이언트-서버 컴퓨팅 방법[OMG91]에 따라 설계 구현되었다. 이러한 접근 방법을 채택한 이유는 다음과 같다.

- 라이브러리로 제공할 경우 멀티미디어 입출력 및 처리가 각 응용 프로그램의 프로세스마다 수행되기 때문에 멀티미디어 자원의 일괄적인 관리와 전반적인 스케쥴링이 불가능하다.
- 클라이언트-서버 접근 방법의 가장 큰 문제는 빈번한 컨텍스트 스위칭 등으로 인한 성능의 저하인데, MuX-II의 경우 클라이언트-서버 접근 방법을 취하게되면 클라이언트가 멀티미디어 데이터 자체를 다루기 위하여

빈번히 RPC(Remote Procedure Call)를 통하여 서버로 부터 데이터를 건내 받기를 요구하는 경우는 극히 드물고 오히려 creation, play, stop, open, close, delete 등의 지극히 간단한 제어적 차원의 RPC 만을 수행하고 멀티미디어 프리젠테이션이 수행되는 도중에는 서버와의 RPC를 거의 하지 않을 뿐아니라 오히려 클라이언트 프로세스는 멀티미디어 이외의 작업을 효율적으로 수행할 수 있기 때문에 전체적인 반응 시간을 높일 수 있게 된다.

MuX-II가 클라이언트 서버 모델에 의하여 구현되었으므로 MuX-II를 사용하고자 하는 응용 프로그램 즉, 클라이언트 프로세스는 먼저 통신망 상에 존재하는 MuX-II 서버와 접속해야 한다. 이를 위하여 MuX-II의 클라이언트 라이브러리는 다음과 같은 객체지향형 API를 제공한다.

*MuX::MuX(char *host_name)*

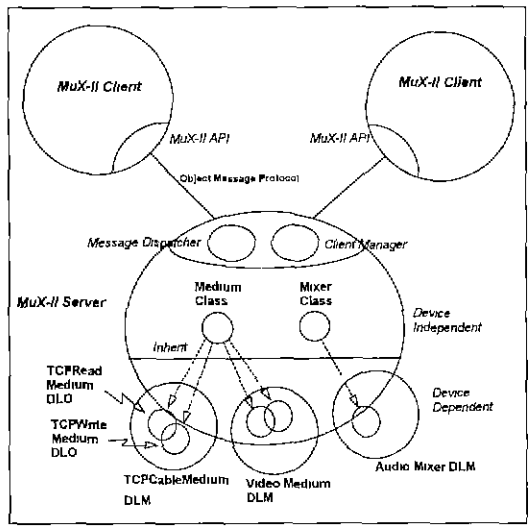
즉, MuX 객체를 생성하므로써 서버와 접속되는데 MuX 객체의 생성은 아래와 같이 실시한다.

*MuX *hama_mux = new MuX(hama.etri.re.kr);*

이와 동시에 서버측에는 클라이언트와의 통신을 담당하는 객체가 하나 생성되는 데 이 객체는 자체 쓰레드를 이용하여 클라이언트로 부터 전달되는 메세지들을 수신받아 그 메세지에 해당하는 서비스 함수를 호출하고 그 결과를 돌려 주는 기능을 수행한다. 이러한 객체를 본 문서에서는 proxy client라 부르며, 프로세스로서 존재하는 클라이언트를 MuX 클라이언트라 부른다.

한편 클라이언트 서버 모델에서는 클라이언트 측의 명령이 서버에서 수행되는 시점이 클라이언트 프로그래머에게 투명해야 하는데 이를 위하여 MuX API는 FlushMessage()와 SyncMessage() 함수를 제공한다. MuX 클라이언트의 API 함수 중

- 서버로 부터 리턴값을 필요로 하지 않는 함수의 경우: 이러한 함수는 모두 클라이언트측 메세지 큐에 보존되고, 클라이언트 프로세스가 FlushMessage()나 SyncMessage()를 호출할 때 일괄적으로 전송되어 순서적으로 수행된다. 이 중 SyncMessage()는 버퍼의 내용을 전송 하고 난 후 그 모든 내용의 수행이 완료되기를 기다리는 일종의 동기함수(blocking function)이고, 반면 FlushMessage()함수는 수행의 완료를 기다리지 않고 바로 리턴되는 형태의 함수이다.
- 서버로 부터 리턴 값을 필요로 하는 함수의 경우: 이러한 함수는 마치 SyncMessage() 함수를 미리 호출을 하고 그 함수를 호출하는 것과 같이 수행이 된다. 즉 그 함수의 리턴 값이 그 이전에 호출하는 함수들의 결과와 관계가 있으므로 반드시 메세지 큐에 쌓여 있는 함수 호출들을 모두 수행시키고 그 함수를 최종적으로 수행시킨 후 그 결과를 클라이언트로 리턴한다.



(그림 6) MuX-II 시스템의 구조

따라서 클라이언트 프로그래머는 자신의 프로그

램 중 MuX의 함수 호출들이 수행되어야 할 시점에서

hama_mux->FlushMessage() 또는

hama_mux->SyncMessage()

를 호출해야 한다.

3.2 가상 객체 인터페이스

MuX의 모든 클라이언트 응용 프로그램은 C++ 클래스 중심으로 구성된 MuX API를 이용하여 작성되어야 하는 데 MuX API는 가상 객체 인터페이스(Virtual Object Interface)를 사용하여 MuX-II 서버 내에서 멀티미디어를 처리하는 기본 모델을 거의 투명하게 반영하고 있다.

가상 객체 인터페이스는 서버와 클라이언트 간에 객체의 생성과 소멸, 그리고 소속 함수의 호출 등에 관한 특수한 프로토콜을 통하여, 서버 내에 실제적인 객체가 존재하도록 하고 클라이언트에게는 가상적인 객체를 만들어주어 클라이언트로 하여금 마치 실제 객체인양 사용할 수 있도록 하는 기법이다. 따라서 API로 설명되는 모든 객체는 가상 객체라 할 수 있으며 실제 객체는 연결되어진 MuX 서버 내에 존재한다.

예를 들어, 추후 설명할 Source 객체를 클라이언트가 사용하기 위해서 클라이언트측에는 Source 가상 객체(Virtual Source Object)를 만들고 이에 상응하는 실제 Source 객체(Real Source Object)는 서버에 만들어야 한다. 이를 위하여 MuX는 다음과 같은 형태의 API를 제공한다.

*Source::Source(MuX *, ...)*

그리고 상기의 API를 클라이언트에서 다음과 같이 수행 함으로서 MuX 클라이언트측에는 가상 Source 객체, 그리고 MuX 서버측에는 실제 Source 객체를 만들게 된다.

*Source *mic_source = new Source(hama_mux, ...);*

그런 연후에 MuX 클라이언트 프로그래머는 예를 들어,

mic_source->Open();

위와 같이 *mic_source*가 포인트하고 있는 MuX 클라이언트 내의 가상 Source 객체의 멤버 함수, 여기서는 Open() 함수를 호출하면 MuX API 메카니즘에 의하여 자동적으로 MuX 서버내의 실제 Source 객체의 실제 Open() 멤버 함수를 호출하게 된다.

3.3 Dynamic Linking Object : 멀티플랫폼을 위한 고려

전술한 바와 같이 MuX가 추구하는 컴퓨팅 환경은 분산 환경이다. 이러한 환경에서 MuX는 멀티플랫폼에서 소스 코드 차원의 호환성을 제공할 수 있어야 한다. 이를 위하여 MuX 시스템은 설계시에 Dynamic Linking Library개념을 사용하였다. 멀티플랫폼 환경에서는 플랫폼이 변경됨에 따라 입출력 디바이스 인터페이스나 멀티미디어 처리를 위한 특수 하드웨어 등이 바뀌게 된다. 이런 경우 MuX의 공개 초기에 몇가지의 지정된 입출력 장치 제품과의 입출력 만을 고려하여 MuX 내에 고정적으로 프로그램해 놓는다면 플랫폼이 바뀔 때마다 MuX 내부의 프로그램을 변경하거나 확장해야 하고 이렇게 하기 위해서는 MuX 소스 프로그램의 많은 부분을 이해해야만 할 것이다. 따라서 이러한 접근 방법을 취할 경우 새로운 하드웨어 및 디바이스 드라이버 개발자가 그들의 결과물을 MuX에 접속하여 사용해 보는 것이 매우 어려운 일이 될 것이다.

DLM(Dynamic Linking Module)은 DLL(Dynamic Linking Library) 기능에 객체 처리 기능을 추가한 것으로 MuX 서버에 동적으로 접속된다. DLL은 윈도우95 및 윈도우 NT 환경 뿐아니라 UNIX 환경에서도 Shared Object 또는 Shared Library 등

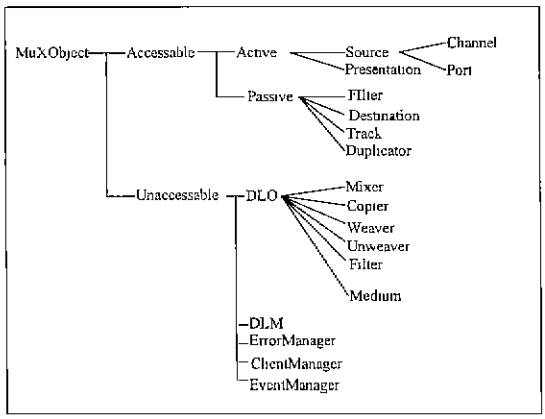
으로 제공된다. 플랫폼이 변경됨에 따라서 함께 변경되어야 하는 부분이나 또는 새로운 장비의 구입이나 개발에 따라 MuX 입출력을 확장해야 하는 부분, 즉 Device Dependent 부분을 MuX가 제시하는 DLM 작성 지침을 따라 작성하면 MuX가 수행 중에도 새로운 DLM을 로드하여 사용할 수 있고, 필요없는 DLM은 없애기도 할 수 있다.

그림 6에 나타낸 바와 같이 DLM의 프로그래밍은 DLM 내의 객체인 DLO(Dynamic Linking Object)를 프로그램 하는 것이 주된 작업이다. DLO의 기본 클래스는 MuX의 고정부분 즉, Device Independent 부분이 사용하는 클래스들로서 통신, 화일 등을 포함한 각종 입출력 장치와의 일관성있는 인터페이스를 위한 미디엄(Medium) 클래스, 스트림 복제기나 분리기와 유사한 기능을 위한 Splitter 클래스, 채널의 다양한 기능을 위한 Mixer 클래스, 스트림의 다양한 필터 기능을 위한 Filter 클래스 등이 제공된다. 현재 제공되는 복제기(Copier)와 분리기(Unweaver)도 모두 Splitter의 서브클래스 DLM으로 작성되었다. 또한 채널이 사용하는 합성기(Mixer)와 접합기(Weaver)도 DLM으로 만들 수 있도록 하여 현재 Mixer의 서브클래스로서 Wavemixer와 Weaver 등이 제공되고 있다.

4. 멀티쓰레드와 동기화

4.1 쓰레드 객체의 상속 체계

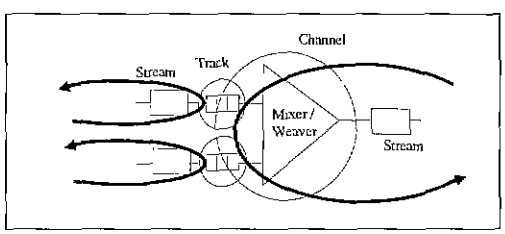
본 고에서 제시한 스트림, 채널, Duplicator 등 멀티미디어 요소 객체는 개별적으로 동작이 가능하지만 다양한 응용 프로그램을 만들어내기 위하여 이러한 요소 객체들이 연결될 수 있다. 이러한 연결은 우선 그림 7에 나타낸 바와 같이 상기의 요소 객체가 서로 간에 상속체계를 가지므로서 가능해 진다. 상속체계의 가장 상위 클래스는 MuXObject 클래스이다. MuXObject 클래스는 상기에서 설명된 모든 객체에 상속되며 객체관리에 필요한 기능을 수행한다. Accessable클래스는 MuX-II 서버 내의 모든 객체가 네트워크 환경에서 투명한 접근이 가능하도록 객체의 생성과 소멸시 이름관리 및 디렉토리 관리 부분에 해당 객체를 등록 또는 삭제시키는 관리기능을 한다. MuX-II API를 통하여 응용 프로그램이 사용할 수 있는 모든 클래스는 바로 이 Accessable 클래스를 상속 받는다. Accessable 클래스 중에서 Active 클래스는 동적인 역할을 하는 객체의 클래스로서 각자 쓰레드 하나씩을 갖고 미디어 처리에 필요한 단위 작업을 동적으로 수행한다. Active 클래스에는 상기에서 설명한 클래스 중 Source 클래스, 채널 클래스, Port 클래스, 프리젠테이션 클래스 등이 있다. 반면 Passive 클래스는 Active 클래스에 접속되어 접속된 Active 클래스를 수행하는 쓰레드에 의하여 수행된다. Unaccessable 클래스는 클라이언트 프로그램이 접근할 수 없는 서버 내부 클래스로서 주로 DLO의 서브클래스인 미디엄, 믹서, 복제기, 분리기, 접합기, 합성기용 DLO 서브클래스 등이 이에 해당된다. 또한 DLO를 관리하기 위한 DLM 클래스와 ErrorManager, EventManager, ClientManager 등도 모두 Unaccessable 클래스에 속한다.



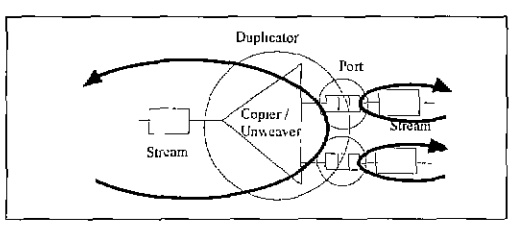
(그림 7) 클래스 상속체계

4.2 쓰레드의 접속

미디어의 데이터 플로우는 각 Active 객체의 쓰레드 구동에 의하여 일어난다. 특히 원점 객체는 스트림 중에서 유일한 Active 객체로서 스트림이 개념적 뿐만아니라 실제의 동작에서도 컨커런시의 기본 단위 객체라 할 수 있다. 이뿐아니라 원점 객체는 다른 Active 객체인 채널과 포트에 상속되어 이들 역시 컨커런트하게 동작될 수 있도록한다. 따라서 전술된 스트림, 채널, Duplicator와 관련된 그림 3, 그림 4, 그리고 그림 5의 개념도는 사실상 그림 8과 그림 9에 굵은 화살표로 나타낸 바와 같이 여러개의 컨커런트한 스트림이 접속되어 있는 것이다.



(그림 8) 채널 관련 컨커런트 스트림의 접속



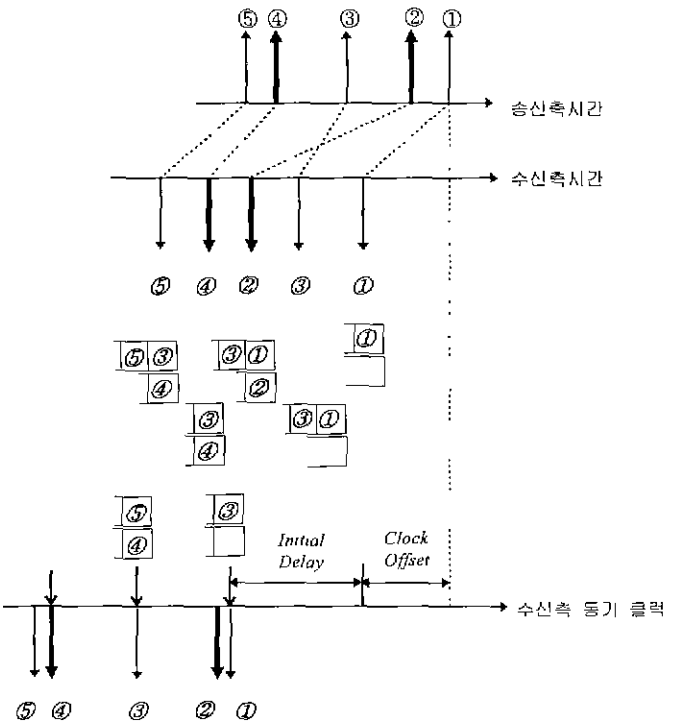
(그림 9) Duplicator 관련 컨커런트 스트림의 접속

이상과 같이 MuX-II의 멀티미디어 처리 모델은 쓰레드에 의하여 독립적으로 수행되는 단위 스트림을 접합체라할 수 있는 복제기, 합성기, 접합기, 분리기 등에 접속함으로서 다양한 멀티미디어 응용에 필요한 멀티미디어 데이터의 흐름을 용이하게 만들어 낼 수 있도록 되어있다[Gib91, Min94, IMA93].

이와같이 스트림의 연결에 기반한 모델을 사용하는 이유는 응용 프로그램이 시작되기 전에 이미 연결 형태가 정해지는 정적인 경우 뿐아니라 응용 프로그램 수행 중에 데이터 흐름 구조를 변경하여야 하는 경우에도 단순히 상기 객체들의 접속 관계를 변경함으로서 이러한 동적인 요구를 만족시킬 수 있다는 장점을 갖고 있기 때문이다. 이러한 동적 연결은 많은 부분에서 요구될 수 있는데 특히 다자간 영상회의시 참석자의 이참석이나 발언권의 부여 등에 따라 스트림 연결의 변경이 필요한 경우, 또는 통신망이나 데이터 흐름에 참여되어 있는 호스트 등에 문제가 발생하거나 QoS 유지에 어려움이 있어서 흐름의 경로를 수행 중에 변경하여야 하는 경우 등이 있을 수 있다.

4.3 동기화

동기화 방식은 수신측에서 각 스트림 마다 특수한 큐를 두고 프레임이 수신되면 이를 일단 큐에 저장한 후 큐에서 프레임을 인출할 때는 동기가 맞는 프레임만을 인출하는 것이 골자이다. 우선 Initial Delay 기간동안 수신되는 프레임을 큐에 저장한 후 그 시기의 끝부터 동기 클럭으로부터 주기적으로 타이머콜백을 받는다. 그 예가 그림 10에 나타나 있다. 현재의 예에서는 Initial Delay 기간동안 ①②③ 프레임이 수신되어 큐에 저장되어 있다. 이 시점에서 첫번째 타이머 콜백이 발생하면 그 콜백이 발생한 시각을 환산하여 송신 시각으로 변환하고 큐에서 이보다 이른 타임 스탬프를 갖고 있는 ①② 프레임을 인출하게 된다. 다음 콜백이 들어오기까지는 다시 송신된 프레임을 큐에 입력하기만 하기 때문에 ④ 번 프레임이 큐에 입력되어 이제 큐에는 ③④ 프레임이 저장되게 된다. 그 후 두번째 콜백이 들어오면 이의 시각을 환산하여 송신시각으로 변화하고 이 시각 이른 타임 스탬프를 갖는 프레임을 인출하게 되



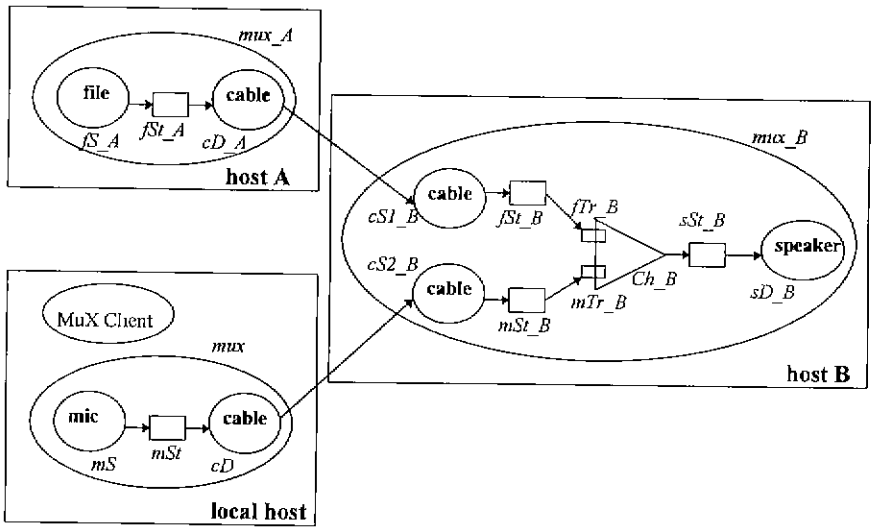
(그림 10) 큐를 이용한 동기화 예

어 ③번 프레임을 출력시키게 된다. 다시 다음 콜백까지는 송신된 프레임을 큐에 입력하여야 하므로 ⑤번 프레임을 큐에 넣게되고 세번째 콜백이 들어왔을때 ④⑤번 프레임이 동기가 되어 출력되게된다. 이렇게 함으로서 지터에 의하여 동기가 어긋난 ②③ 프레임이 제 순서를 찾게 된다. 또한 동기 클럭의 해상도만 높다면 각 프레임 간의 시간 간격도 원 상태로 복구할 수 있다.

이러한 큐에 의한 동기화 방식의 특징은 프레임이 일단 큐에 저장되었다가 나가므로 그동안 시간적인 여유가 있어서 지터의 완충 작용을 할 수 있을 뿐아니라 시간큐의 특징상 순서의 엉킴도 복구하면서 동기를 맞출 수 있다는 점이다. 뿐만아니라 한 스트림 내의 연속된 두 프레임간의 간격 자체도 일정하게 유지되므로 스트림 내 동기 및 스트림 간 동기라는 결과를 동시에 거둘 수 있는 이론상 최적의 방안이라 할 수 있다. 그러나 큐에 일단 얼마간의 프레임을 저장해 놓은 상태에서 동기를 맞추어야 하기 때문에 초기에 얼마간의 지연을 시켜야 하고 이 지연값은 그 이후에 동기 방식의 평균 지연 값으로 작용하게 됨으로서 지연이라는 측면에서 볼때는 단점이 있다고 할 수 있다. 따라서 지연의 증가를 방지하면서 동기왜곡치를 최대한 낮추는 방식이 고안되어야 할 것이다.

5. 프로그램 예제: 분산 환경에서의 오디오 믹싱

이제 우리가 흔히 사용하는 사운드 카드를 이용하여 수행가능한 예제를 다루어 본다. 리모트



(그림 11) 오디오 믹싱 개념도

호스트 A에 있는 오디오 화일과 로컬 호스트에 있는 마이크로 부터 오디오 입력을 받아 제 3의 리모트 호스트인 B의 스피커로 오디오 믹싱 결과를 출력하는 MuX-II 응용 프로그램을 작성한다. 이 때, 본 응용 프로그램 즉 MuX-II 클라이언트는 로컬 호스트에서 수행되며, 각 호스트 간의 네트워크 연결은 TCPCable DLL을 이용하여 이루어진다. 이 예제의 객체 구성 개념도는 그림 11과 같다. 클라이언트인 본 예제 프로그램은 로컬 호스트에서 수행되고, 오디오 믹싱은 스피커를 갖고 있는 호스트 B에서 이루어진다. 로컬 호스트가 아닌 경우에 각 객체의 변수 이름은 각 리모트 호스트의 별명을 함께 붙여서 서로 구분하였다.

각 MuX 객체의 생성 부분은 각 호스트별로 구분되어 있다. 각 호스트의 다른 자원의 객체를 생성하기 전에 MuX-II 서버와의 통신을 대행할 클라이언트 측 프록시인 *MuX 프록시 객체*를 먼저 생성해야 한다. 이 때, 로컬 호스트의 MuX 객체는 아무 매개변수 없이 mux = new MuX()를 수행하면 되고, 리모트 호스트의 MuX 객체는 해당 호스트의 이름을 매개변수로 하여 mux_A = new MuX(Host_A) (혹은 mux_B = new MuX(Host_B))처럼 수행하면 된다. 이제 변수 mux와 mux_A는 서로 별개의 MuX-II 서버를 지칭하는 상호 독립적인 MuX 프록시 객체이다. 이렇게 함으로써 본 예제 프로그램은 MuX-II 클라이언트로서 세개의 MuX-II 서버에 연결된다.

cD = new Destination(mux, "TCPCableWrite", 0, (char *) NULL) 및 cD_A = new Destination (mux_A, "TCPCableWrite", 0, (char *) NULL)는 네트워크에 데이터를 전송하기 위한 Destination 객체를 생성하는 부분으로서 TCP- CableWrite DLO를 사용하였다. TCPCableWrite DLO는 Destination 객체에서만 사용될 수 있으며, Destination 객체를 생성할 때에 세번째와 네번째 매개변수를 취하지 않아 0과 NULL값으로 지정한다.

반면에 네트워크에서 데이터를 읽어오기 위한 Source 객체를 생성하는 부분은 cS1_B = new Source(mux_B, "TCPCableRead", (unsigned long)

```
/* 로컬 호스트에 대한 객체 생성 */
mux = new MuX();
mS = new Source(mux, "AudioInput", 0, (char *) NULL);
cD = new Destination(mux, "TCPCableWrite", 0, (char *) NULL);
mSt = new Stream(mux, mS, cD);
mux->FlushMessage();

/* 리모트 호스트 A에 대한 객체 생성 */
mux_A = new MuX(Host_A);
fS_A = new Source(mux_A, "WaveFileInput", 0, PowerOfLove);
cD_A = new Destination(mux_A, "TCPCableWrite", 0, (char *) NULL);
fSt_A = new Stream(mux_A, fS_A, cD_A);
mux_A->FlushMessage();

/* 리모트 호스트 B에 대한 객체 생성 */
mux_B = new MuX(Host_B);
cS1_B = new Source(mux_B, "TCPCableRead",
                   (unsigned long) cD_A, Host-A);
char this_host[64];
gethostname(this_host, 64);
cS2_B = new Source(mux_B, "TCPCableRead",
                   (unsigned long) cD, this_host);
Ch_B = new Channel(mux_B, "WaveMixer");
fTr_B = new Track(mux_B);
mTr_B = new Track(mux_B);
fSt_B = new Stream(mux_B, cS1_B, fTr_B);
mSt_B = new Stream(mux_B, cS2_B, mTr_B);
Ch_B->AddTrack(fTr_B);
Ch_B->AddTrack(mTr_B);
sD_B = new Destination(mux_B, "AudioOutput", 0, (char *) NULL);
sSt_B = new Stream(mux_B, Ch_B, sD_B);
mux_B->FlushMessage(),

/* 각 호스트의 리소스를 오픈하고 플레이 */
fS_A->OpenAsync();
cD_A->OpenAsync();
mux_A->FlushMessage(),

cS1_B->OpenAsync();
cS2_B->OpenAsync();
sD_B->OpenAsync();
mux_B->FlushMessage();

mS->OpenAsync();
cD->OpenAsync();
mux->FlushMessage();

Ch_B->Play();
mux_B->FlushMessage();
```

cD_A, Host_A)과 cS2_B = new Source(mux_B, "TCPCableRead", (unsigned long) cD, this_host)처럼 TCPCableRead DLO를 이용한다. TCPCable- Read DLO는 Source 객체에서만 사용될 수 있으며, Source 객체를 생성할 때 세번째와 네번째 매개변수는 의미를 가진다. 즉, 세번째 매개변수는 이 Source 객체와 네트워크를 통해서 연결될 리모트 호스트의 Destination 객체를 의미하고, 네번째 매개변수는 해당 리모트 호스트의 이름을 의미한다.

본 예제 프로그램은 MuX-II 클라이언트가 세개의 MuX-II 서버에 연결되기 때문에 각 서버로 보내지는 메시지를 위한 메시지 큐가 각 해당 프록시 객체 내에 별도로 존재한다. 따라서 FlushMessage()도 mux->FlushMessage()와 mux_ A->FlushMessage() 등과 같이 대상 proxy client에 대하여 각각 구분하여 해야 한다. 그런 후 입출력 디바이스나 네트워크에 관련된 객체를 open하여 데이터의 흐름을 준비한 후, 실제 데이터의 흐름을 개시하기 위하여 가장 상위의 객체인 Ch_B->Play()를 수행한다.

6. 결 론

본 고에서는 최근에 인터넷의 대중화와 아울러 그 필요성이 날로 증대되어 가고 있는 실시간형 분산 멀티미디어 서비스를 위한 시스템으로서 MuX-II에 대하여 소개하였다. MuX-II의 큰 특징으로는

- 서버-클라이언트 모델로서 OMG의 CORBA [OMG91]와 연계하여 사용하여 CORBA 환경상에서 실시간 멀티미디어 서비스를 제공하는 데에 적합한 구조를 갖고 있다는 점
- 객체 연결 모델에 근거하여 응용 프로그램 개발을 위한 직감적인 API를 제공하고 있다는 점
- 디바이스에 무관한 부분과 관련이 있는 부분을 명확히 구분하여 멀티플랫폼 상에서 재구성이 용이하다는 점 등을 들 수 있다.

한편 MuX는 현재 국제 표준화 기구인 ITU에서 인터넷용 실시간 멀티미디어 서비스를 위한 단말기 표준으로 제시하고 있는 H.323[H323]을 쉽게 수용할 수 있다. 즉, 호처리 프로토콜인 H.225.0[H225]과 영상회의 제어를 위한 표준인 H.245[H245]는 모두 MuX-II의 API위에 존재하는 MuX-II의 영상회의용 툴킷으로 만들 수 있다. 또한 H.323에서 제시하는 통신 프로토콜인 RTP[H225]와 RTCP[H225]도 MuX-II 서버 내의 하나의 DLO로 작성할 수 있다. 이러한 사항은 현재 MuX-II의 개발 과정에서 고려되고 있는 사항으로 구현이 완료되는 시점에서는 MuX-II를 이용한 국제 표준 영상회의의 응용 프로그램을 누구나 개발할 수 있을 것으로 기대된다.

참고문헌

[1] [Kim92] Doohyun Kim, SooHyoung Oh, JinKyung Hwang, YoungHwan Lim and Earl Rennison, An Integration and Synchronization Model for Audio, Video and Timebased Graphics, The 2nd Pacific Rim Conference on Articifial Intelligence, Seoul, pp. 1120 - 1123.

[2] [Ren92] Earl Rennison, Rusti Baker, Doohyun Kim, Young-Hwan Lim, MuX: An X Co-Existent, Time-Based Multimedia I/O Server, The X Resource 1, Winter 1992, pp 213-233.

[3] [Baker93] Rusti Baker, Alan Downing, Kate Finn, Earl Rennison, Doohyun Kim, and YoungHwan Lim, Multimedia Processing Model for a Distributed Multimedia I/O Systems, NOSSDAV, 1993, pp. 233-239.

[4] [Kim95a] 김 두현 외, MuX V1 R2 Tutorial, MuX 사용자 그룹, 1995.

[5] [Kim95b] 김 두현 외, MuX V1 R2 API Reference Manual, MuX 사용자 그룹, 1995.

[6] [Kim96] Doohyun Kim, Young-Hwan Lim, "An Object-Oriented, Client-Server Architecture for a Generalized Multimedia Processing Model in a Distributed Multimedia System," KIPS Vol. 3, No. 1, 1996.1, pp.9-32.

[7] [Jun93] 전윤호, 송동호, 박치항, DLL에 의한 멀티미디어 데이타 처리 객체의 구현, 한국정보과학회 추계 학술대회, 1993, pp. 455-458.

[8] [OMG91] Object Management Group, The Common Architecture Request Broker: Architecture and Specification, OMG Document Number 91.12.1 Revision 1.1

[9] [IMA93] Interactive Multimedia Association, Multimedia System Services, Version 1.0, Contributed by Hewlett-Packard Company, IBM Inc., and SunSoft Inc., June 1, 1993

[10] [GIB91] Gibbs, S. "Composite Multimedia and Active Object," Proc. OOPSLA'91, pp. 97-112.

[11] [MIN94] Mines, R., J. Friesen, and C. Yang, "DAVE: A Plug and Play Model for Distributed Multimedia Application Development," Proc. ACM Multimedia94, 15-20 Oct. 1994, San Francisco, CA, pp.59-66.

[12] [H225] ITU-T Recommendation H.225.0 (199X): "Media Stream Packetization and Synchronization for Visual Telephone Systems on Non-Guaranteed Quality of Service LANs ".

[13] [H245] ITU-T Recommendation H.245 (1995): "Control of communications between Visual Telephone Systems and Terminal Equipment".

[14] [G711] ITU-T Recommendation G.711 (1988): "Pulse Code Modulation (PCM) of Voice Frequencies".

[15] [H261] ITU-T Recommendation H.261 (1993): "Video CODEC for audiovisual services at p X 64 kbit/s"

[16] [H323] ITU-T Recommendation H.322 (1995): "Visual Telephone Systems and Terminal Equipment for Local Area Networks which Provide a Non-Guaranteed Quality of Service".

김 두 현

1985년 서울대학교 컴퓨터공학과 졸업 (공학사)
1987년 한국과학기술원 전산학과 졸업 (이학석사)
1987년-현재 한국전자통신연구원 책임연구원
1991년-1993년 미 스탠포드연구소 객원연구원
1993년 정보처리기술사 취득

황 승 구

1979년 서울대학교 전기공학과 졸업 (공학사)
1981년 서울대학교 전기공학과 졸업 (공학석사)
1986년 University of Florida 전기공학과 (공학박사)
현재 한국전자통신연구원 컴퓨터연구단 멀티미디어연구 부장
관심분야 . 네트워크 컴퓨팅, 인텔리전트 컴퓨팅, 비쥬얼 컴퓨팅, 모빌 컴퓨팅

박 치 항

1974년 서울대학교 응용물리학과 졸업 (이학사)
1980년 한국과학기술원 전산학과 졸업 (이학석사)
1987년 파리 6대학 전산학과 졸업 (공학박사)
1974년-1978년 한국과학기술연구소 연구원
1978년-1985년 한국전자기술연구소 선임연구원
1985년-현재 한국전자통신연구원 컴퓨터연구단장
관심분야 : 멀티미디어, 분산시스템, 그룹웨어, 네트워크 컴퓨팅, 에이전트 아키텍처, 가상현실