멀티미디어 입출력 서버를 위한 분산 MIDI 인터페이스의 설계 및 구현

조 병 호[†] · 강 태 진[††] · 유 기 영[†††] · 이 경 희[††††] · 궁 상 환[††††]

요 약

멀티미디어 입출력 서버인 MuX는 다양한 입출력 디바이스와의 인터페이스를 위해 장치독립적인 DLM(Dynamic Linking Module)을 사용한다. 본 논문에서는 MuX 시스템의 입출력 디바이스 인터페이스 기능을 확장하기 위해서 MIDI(Musical Instrument Digital Interface)형식의 오디오 데이터를 처리할 수 있는 DLM을 추가하였다. MIDI 데이터의 특성과 MIDI 파일 형식을 분석하여 MIDI를 지원하는 디바이스와의 인터페이스를 위한 MIDI DLM(Dynamic Linking Module)과 MIDI 파일을 지원하는 DLM을 설계하고 구현하였다.

The Design and Implementation of the Distributed MIDI Interface for Multimedia I/O Server

Byoung Ho Cho[†] · Tae-Jin Kang[††] · Ki-Young Yoo[†††] · Kyung-Hee Lee[††††] · Sang-Hwan Kung[††††]

ABSTRACT

MuX, the multimedia I/O server, uses device-independent DLMs(Dynamic Linking Module) for the interface with various I/O devices. In this paper, in order to expand the device interfacing ability of MuX system, we added DLMs for manipulating MIDI-format audio data. Analyzing the properties of MIDI data and the MIDI file format, we designd and implemented MIDI DLM for the MIDI devices and MIDI file DLM for the MIDI files.

1. 서 론

MuX는 네트워크상의 분산 환경에서 다양한 멀티미디어 정보의 창출, 접근, 저장, 처리, 전송 및 수신을 원활히 할 수 있게 해주는 멀티미디어 입출력 서버이다. 이는 앞으로 구축될 초고속 정보통신망 상에서 다양한 멀티미디어 응용 프로그램 개발을 위한 미들웨어(middleware)로 사용하기 위해 한국 전자 통신연구원에서 개발하였다[3,6].

현재 MuX는 Solaris 버전과 Windows NT 버전이 발표되었고, MuX 사용자 그룹의 결성과 활동, 그리고 제반 연구과제의 수행으로 인해 MuX의 활용도가 점차 높아지고 있다. 그러나, MuX를 사용 가능한 컴퓨터 시스템도 제한되어 있으며, MuX의 여러 가지 기능을 좀 더 보완 확장할 여지가 있는 실정이다. 특히, MuX 시스템에서 사용 가능한 입출력 디바이스는 입력용으로

† 준 회 원 : 경북대학교 컴퓨터공학과 박사과정
†† 정 회 원 : (주)MJL 연구원
††† 정 회 원 : 경북대학교 컴퓨터공학과 교수
†††† 정 회 원 : 한국전자통신연구원 분산멀티미디어 연구실

논문접수 : 1998년 2월 9일, 심사완료 : 1998년 5월 8일

마이크와 카메라, 출력용으로 스피커와 모니터 정도로 상당히 제한되어 있으며 기본적인 입출력 디바이스만을 지원하고 있다. 따라서 MuX 시스템이 좀더 포괄적인 의미의 멀티미디어 데이터 입출력을 수행하기 위해서는 여러 다른 입출력 디바이스와의 인터페이스를 추가, 확장하는 것이 요구되어진다[7,9,12].

MuX는 다양한 입출력 디바이스의 추가, 확장을 쉽게 하기 위해서 DLM(Dynamic Linking Module)을 사용하여 디바이스 독립적인 인터페이스를 제공한다. 즉, 새로운 디바이스를 MuX 시스템에서 사용하려면 MuX가 제공하는 공통된 인터페이스에 맞게 해당 디바이스의 입출력을 위한 DLM을 만들어 추가하면 된다. DLM은 하나 이상의 DLO(Dynamic Linking Object)로 구성되고, 이 DLO들이 실질적인 디바이스 입출력을 행하는 디바이스 의존적인 부분이다.

본 논문에서는 MuX NT 버전의 오디오 측면의 보강을 위해 MIDI를 지원하는 디바이스와의 인터페이스를 위한 DLM과 MIDI 파일을 위한 DLM을 개발하였다. 사용자들은 MIDI를 지원하는 전자악기를 MuX상에서 사용할 수 있게 되고, 자신의 연주를 원거리에 있는 다른 사람에게 실시간에 보내거나 파일로 저장하였다가 다시 재생할 수 있어 오디오 측면의 풍부한 표현력을 제공받게 된다. 따라서, MIDI 디바이스 인터페이스의 개발로 새로운 응용 프로그램 개발을 기대할 수 있다.

서론에 이어서 2장에서 MIDI와 MIDI 파일 형식에 대해서 기술하고, 3장에서는 MuX에서의 디바이스 인터페이스 기술을 소개하며, 4장에서 본 논문에서 구현한 4개의 DLO에 관해서 설명하고, 5장에서 실험 결과를 보인 후에, 6장에서 결론을 맺는다.

2. MIDI

2.1 MIDI의 정의

MIDI란 Musical Instrument Digital Interface의 머릿글자를 따온 약어로 악기들간에 사용할 수 있는 통신의 수단이라 할 수 있다. Musical Instrument란 음악에서 사용될 수 있는 악기를 비롯한 모든 장비(equipment)를 의미하며 또한 Interface란 서로 분리된 두 개 이상의 장비를 연결시키는 것을 의미한다. MIDI는 서로 다른 음악용 기기(신디사이저, 시퀀서, 드럼머신, 사운드 모듈 등)들이 서로 정보를 주고받으며 자유롭게 서로를 컨트롤하고 합주를 할 수 있도록 하기 위해서 전세계의 모든 음악용 기기 회사들이 오랜 회의를 거쳐 만들어 따르고 있는 '국제적 표준으로 통일된 규격'을 말한다. 다시 말해서, MIDI는 전자악기 제조자들이 동의한 표준안으로 서로 다른 제조자들이 만든 악기들이 어려움 없이 서로간에 음악정보를 전송하도록 하는 규격의 집합이다[1,5].

2.2 MIDI 메세지 형식

MIDI 규격에서는 MIDI 정보를 메시지로 표현하고 있다. 이 메시지는 바이트를 기본 단위로 하며, MIDI 정보의 기능 분류를 위한 상태 바이트(status byte)와 무엇을 어떻게 연주하라는 복수의 데이터 바이트(data byte)로 구성된다. 데이터 바이트의 수와 의미는 앞의 상태 바이트에 의해 결정되어 진다. 상태 바이트와 데이터 바이트의 구별은 최상위 비트로 하는데 상태 바이트의 최상위 비트는 '1'이고 데이터 바이트의 최상위 비트는 '0'이다.

MIDI에는 논리적 채널의 개념이 있다. 이것은 복수

〈표 3〉 MIDI 메시지
〈Table 1〉 MIDI messages

채널 메시지	채널 보이스 메시지	연주되는 음과 악기에 관련된 메시지 (음의 ON/OFF, 음색변화, 악기 컨트롤 스위치 조정)
	채널 모드 메시지	모드 설정 메시지 옴니모드 ON/OFF, 폴리모드 ON/OFF
시스템 메시지	시스템 커먼 메시지	시퀀서 제어 메시지 (Song position 지정, Song 선택)
	시스템 리얼타임 메시지	시퀀서 제어 메시지 (시퀀서의 연주 시작, 중지, 연주 계속)
	시스템 익스클루시브 메시지	모든 MIDI 악기들의 공통된 기능외에 악기 제조 회사마다 고유한 기능들을 지정할 때 사용된다. 각 제조 회사마다 고유의 ID 지정

의 음원을 동시에 연주할 때 특정 소리를 내는 부분의 조성을 지정하는데 사용된다. 모두 16개의 논리적 채널이 있는데 이를 표시하기 위해 상태 바이트에 채널 번호가 있다. 그리고 이런 MIDI 정보들과는 무관하게 효율적인 MIDI 정보의 송수신을 위해서 상태 바이트나 데이터 바이트의 앞뒤에 시작 비트와 정지 비트를 덧붙여서 전체 10비트의 구조로 MIDI 메시지가 전송된다.

MIDI 메시지는 크게 두 종류로 나누어진다. 하나는 채널 메시지(channel message)이고, 또 하나는 시스템 메시지(system message)이다. 채널 메세지는 채널 보이스 메시지(channel voice message)와 채널 모드 메시지(channel mode message)로 구성된다. 또한, 시스템 메시지는 시스템 커먼 메시지(system common message), 시스템 리얼타임 메시지(system realtime message), 및 시스템 익스클러시브 메시지(system exclusive message)로 구성된다. 여러 개의 MIDI 채널 메시지를 연속해서 보낼 때 같은 상태 바이트가 중복되는 경우에는 송신의 효율을 위하여 처음의 상태 바이트는 정상적으로 송신하고 그 다음부터 중복되는 상태 바이트는 생략할 수가 있다. 이런 방식을 러닝 스태이터스(running stauts)라고 한다.

2.3 MIDI 파일 형식

표준 미디 파일 규격(SMF : Standard MIDI File)은 시퀀서(sequencer)가 녹음하거나 연주하는 미디 데이터를 저장하기 위해 특별히 고안된 것이다[2]. 이 규격은 상태 바이트와 적절한 데이터 바이트로 이루어진 표준 미디 메시지와 이 메시지를 연주하기 전에 어느 정도 클럭 펄스(clock pulse)를 기다리는지를 표시하는 시간 정보, 빠르기, 분해능(division), 박자와 음정 표시, 및 트랙과 패턴 이름 등의 다양한 정보들을 저장하는 형식을 나타내준다. 여기에서 사용하는 데이터 구조는 8비트(1 바이트) 데이터 구조이다. 이 규격에서는 8비트 데이터 구조에 무엇이 들어가야 하는지를 지정한다.

MIDI 파일의 규격에서는 헤더(header)와 트랙(track)의 두 가지 청크를 정의한다. 여기서 청크는 서로 연관이 있는 데이터들을 모아 놓은 것인데, 미디 파일에는 여러 개의 청크가 존재하고, 각 청크는 각기 다른 크기로 되어 있다. 미디 파일은 항상 하나의 헤더 청크로 시작하고, 그 뒤에 하나 이상의 트랙 청크로 구성되어진다.

헤더 청크(header chunk)에는 미디 파일 형식과 파일에 몇 개의 트랙이 저장되어 있는지를 나타내는 트랙 청크의 수, 그리고 시간정보의 기준이 되는 분해능이 저장되어있다. 트랙 청크에는 시간정보와 함께 저장된 미디 데이터와 미디 데이터는 아니지만 미디 파일을 설명하는데 필요한 정보들이 저장한다. 다시 말해 시간정보를 가진 연속적인 이벤트 데이터로 되어 있다. 이 벤트는 크게 미디 이벤트(MIDI event), 메타 이벤트(Meta event), 익스클루시브 이벤트(Exclusive event)로 나누어진다. 미디 이벤트는 미디 메시지를 의미하며, 상태 바이트와 데이터 바이트로 이루어진 미디 메시지가 그대로 파일에 저장된 것이라 할 수 있다. 메타 이벤트는 실제로 음악에 관련된 정보는 아니지만 시퀀서에 유용한 정보가 파일에 저장되어 있는 것이고 익스클루시브 이벤트는 미디 메시지중 시스템 익스클루시브 메시지를 파일에 저장하는데 사용된다.

2.4 MIDI의 시간 표현 방법

MIDI 파일에서는 각 이벤트의 연속적인 시간 관계를 delta time을 통해서 표현한다. 하나의 이벤트는 이전 이벤트와의 시간 차이와 실질적인 이벤트정보로 구성되는데, 이전 이벤트와의 시간 차이는 delta time으로 표현되고 실질적인 이벤트는 MIDI 메시지가 된다. 이때, delta time의 단위시간은 header chunk에 있는 division 2바이트와 메타 이벤트로 표현되는 3바이트의 tempo정보로 결정된다.

2.4.1 PPQN(pulse per quater note)

Delta time의 단위시간을 4분 음표 길이의 상대적인 값으로 표현하는 방법이다. 이는 header chunk의 division값 중에서 첫 번째 바이트가 양수일때에 해당하고 이때 division의 값은 4분음표 하나가 가지는 delta time 값이다. 따라서 나머지 모든 dalta time은 이 값에 상대적인 값이된다. 예를 들어서 division이 16진수로 0060(hex)이라면 4분음표 하나가 96 pulse를 가진다는 것이다. 만약 delta time이 0030(hex)이라면 4분음표의 반 즉, 8분음표 길이만큼의 시간이 되는 것이다. 여기서, 이 delta time의 절대시간은 tempo와 결부되어 결정된다.

Tempo는 메타 이벤트 3바이트로 표현되고 4분음표의 길이를 microsecond 단위로 표현한다. PPQN인

경우에 delta-time의 한 단위의 절대시간은 tempo / division의 값이 된다. 예를 들어, tempo가 07A120 (hex)라면 4분음표의 길이가 약 500,000 microsecond (0.5초)가 되고 이때 division이 96이라면 dalta time의 한 단위시간은 0.5초 / 96이 된다.

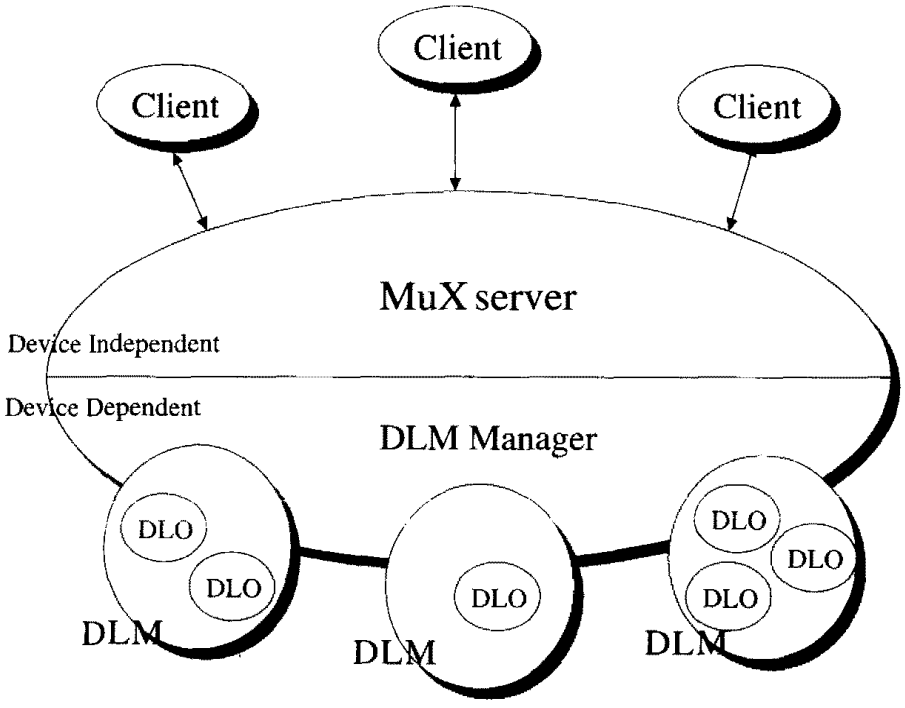
2.4.2 SMPTE

4분음표 길이의 상대적인 값을 표현하는 PPQN과는 달리 SMPTE 방법은 delta time의 단위 시간을 절대시간으로 준다. Division의 첫 번째 바이트가 음수일 경우가 여기에 해당하며 첫 번째 바이트는 1초당 프레임수를 음수로 나타내고 두 번째 바이트는 한 프레임당 서브 프레임의 수를 나타낸다. 이때 서브 프레임이 dalta time의 단위가 된다. 예를 들어 division이 E728(hex)라면 첫 번째 바이트(E7)가 -25 즉, 초당 25 프레임을 나타내고 두 번째 바이트(28)가 40 즉, 프레임당 40 서브 프레임을 가짐을 알 수 있다. 따라서, 초당 25*40 = 1000 서브 프레임을 가지므로 한 서브 프레임은 1 millisecond가 되고, 이것이 delta-time의 단위가 된다.

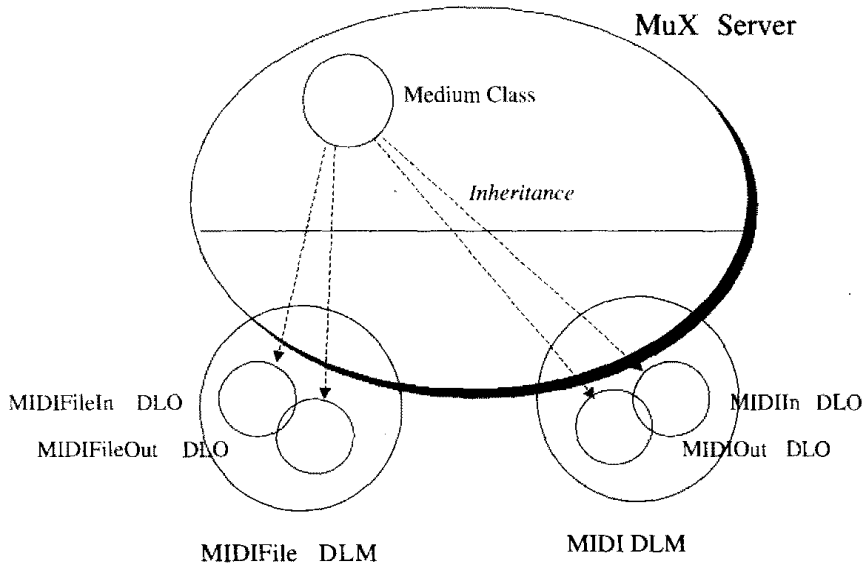
3. MuX의 디바이스 인터페이스

MuX는 다양한 멀티미디어 디바이스를 손쉽게 지원하기 위해서 DLM(Dynamic Linking Module)을 사용하여 디바이스 독립적인 인터페이스를 제공한다. 즉, 새로운 디바이스를 MuX 시스템에서 사용하려면 MuX가 제공하는 공통된 인터페이스에 맞게 해당 디바이스의 입출력을 위한 DLM을 만들어 추가하면 되도록 설계되어있다[10].

MuX 서버는 여러 가지 클래스 객체들로 구성되어 수행되는데 그 중에서 디바이스와 관련된 클래스는 DlmMgr(DLM 매니저), Dlm(DLM), Dlo(DLO), Medium등이다. MuX 서버는 하나의 DLM 매니저를 생성하여 서버내에 등록된 DLL들로부터 DLM들을 생성하여 DLM 테이블에 등록하고 관리한다. Dlo 클래스



(그림 9) MuX의 디바이스 인터페이스
(Fig. 1) Device interfaces of MuX



(그림 10) MIDI DLM의 구조
(Fig. 2) The architecture of MIDI DLM

는 멀티미디어 입/출력 기능을 가지는 기본 객체로서 DLL내의 모든 객체는 Dlo로부터 상속된다. DLO 객체의 기본적인 속성은 DloAttribute 구조체에 저장된다. Medium 클래스는 멀티미디어 데이터를 MuX와 디바이스, 네트워크 또는 파일 시스템간에 입출력하는 객체의 클래스로 입력 또는 출력을 하나 가진다.

DLM들은 DLL로 구현되는데 DLL내에는 DLM을 접근하기 위한 공통된 인터페이스 함수와 DLO 객체로 생성될 클래스 정의가 포함되어 있다. 인터페이스 함수는 DLM을 초기화하는 함수, DLM에 있는 DLO를 생성하는 함수, DLO 정보를 제공하는 함수등이 포함된다. DLM 매니저에 의해 생성된 DLM 객체는 DLL에 정의된 인터페이스 함수의 포인터들을 저장하였다가 DLO를 포함하는 MuX 객체가 생성될 때 함수 포인터를 사용하여 DLO 객체를 생성한다. DLL내에 정의되는 클래스는 Medium 객체로부터 상속받아 정의되고 Medium 객체가 Dlo 클래스로부터 상속받는 클래스이므로 DLO의 기능을 하는 것이다. 새로이 정의된 클래스의 각 함수들이 실질적인 디바이스 입출력을 수행하는 부분이 된다.

따라서, 실질적으로 새로운 디바이스를 추가하기 위해서는 Medium 객체로부터 상속받는 새로운 클래스와 DLM을 위한 인터페이스 함수를 포함하는 DLL을 만들어 MuX에 등록하면 된다.

4. 설계 및 구현

본 논문에서는 MuX에서 MIDI 디바이스와 파일을 사용할 수 있도록 하기 위한 DLM을 개발하고 추가하였다. MIDI DLM은 크게 두 부분으로 나눌 수 있는데, MIDI 파일로부터 입출력하는 DLM과 MIDI 디바이스로부터 입출력하는 DLM이다. 각 DLM은 입력과 출력에 해당하는 두 개의 DLO로 구성된다. 본 논문에서 개발되어 MuX에 추가되는 DLM과 DLO는 그림 2와 같은 관계를 가지며, 다음 표 2과 같이 요약된다.

〈표 4〉 MIDI를 위한 DLL과 DLO
〈Table 2〉 DLLs and DLOs for MuX

DLL 이름	DLO 이름	기 능
FileIn.DLL	MIDIFileIn	MIDI 포맷의 오디오파일을 입력받는다.

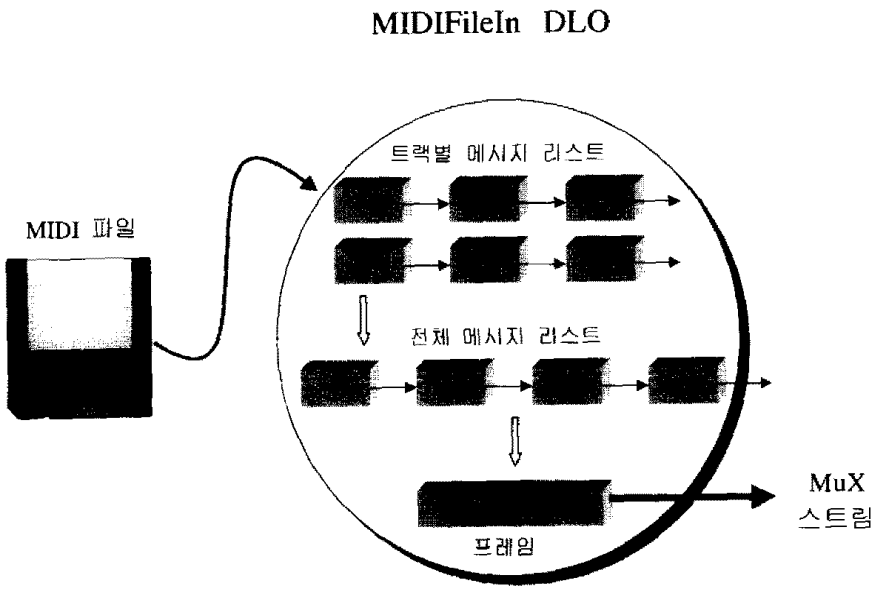
FileOut.DLL	MIDIFileOut	MIDI 포맷의 오디오파일을 출력한다.
MidiIn.DLL	MidiIn	MIDI 포맷의 오디오정보를 전자악기로부터 입력받는다
MidiOut.DLL	MidiOut	MIDI 포맷의 오디오정보를 음원보드에 출력한다.

4.1 MIDI 파일 입력 DLO

MIDI 파일은 크게 header chunk와 track chunk로 구성되는데 header chunk에는 파일 전체에 대한 정보 즉, 파일 형식, 트랙의 수, 시간 단위로 사용되는 delta-time에 관한 정보들이 있고, 그 다음에 실질적인 MIDI 메시지를 가지고 있는 track chunk들이 연속해서 저장된다. Track chunk에는 MIDI 메시지와 부가적인 meta event들이 저장되는데 이들 앞에는 시간 정보인 delta-time이 선행한다. Delta-time은 상대적인 값으로 바로 앞 메시지와의 시간 간격이다.

파일에 저장된 메시지의 종류는 일반적인 보이스 메시지, 메타 이벤트, 시스템 익스클루시브 메시지로 나누어지고 이들은 각각을 Message라는 상위 클래스에서 상속받은 클래스 객체에 저장한다. 이렇게 저장된 객체들은 각 트랙별로 MessageList라는 linked list 형식의 클래스로 만들어진다. 복수개의 트랙을 가지는 MIDI 파일은 트랙 수만큼의 track chunk를 가지는데 이 때의 delta-time은 각 트랙마다 독립적으로 0에서 시작하여 이전 메시지와의 시간 간격을 표현하므로 전체 곡의 메시지 스트림을 시간 순으로 맞게 만들려면 트랙별로 만들어진 MessageList를 읽어서 이들을 곡의 시작으로부터의 절대시간으로 바꾸고 이들 전체를 시간 순으로 재정렬해야한다. 이렇게 만들어진 메시지 스트림을 MuX 스트림을 통해서 다른 DLO에 넘겨주기 위해서는 MuX에서 정의된 frame 클래스를 사용해야 하는데, 하나의 메시지의 내용을 각 형식에 맞는 구조체(VOICE, META, EXCLUSIVE)로 옮기고 이 구조체를 문자배열에 옮긴 후 이를 고정된 크기로 모아서 하나의 프레임으로 만든다.

위의 기능들을 MIDI 파일의 입력을 담당하는 MidiFileIn 이라는 C++ 클래스로 구현하였다. 멤버 함수는 크게 두 종류로 나눌 수 있는데 한 종류는 MuX 서버의 Medium 클래스의 함수를 대치하는 가상 함수들이고, 다른 하나는 MIDI 파일을 처리하기 위해 개발된 함수들이다.



(그림 11) MIDI 파일 입력 DLO
(Fig. 3) DLO for MIDI file input

4.2 MIDI 파일 출력 DLO

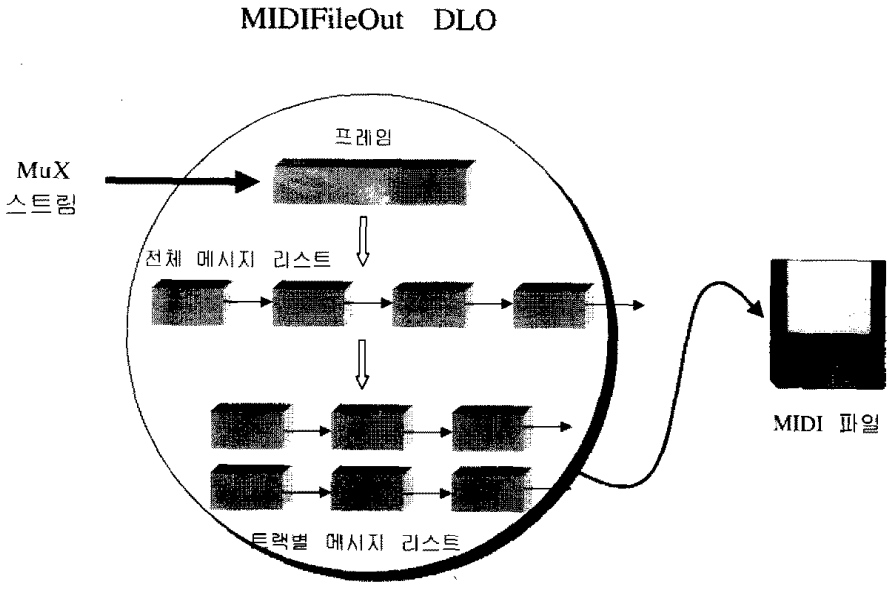
스트림으로부터 전달받은 프레임들을 MIDI 파일로 저장하기 위해서는 프레임의 데이터만으로는 얻을 수 없는 정보들이 있다. MIDI 파일의 시간 정보를 나타내는 분해능(division)과 프레임의 크기가 바로 그것이다. 여기에 작업의 편리를 위해서 트랙의 수를 프레임의 헤더에 포함시킨다. MIDI 파일 출력을 담당하는 MidiFileOut 클래스에서 이들을 이용하기 위해서는 먼저 MidiFileIn 클래스와 MidIn 클래스에서 프레임을 만들 때 헤더를 생성해야만 한다.

MIDI 파일을 만들기 위해서는 MuX 스트림으로부터 전달받은 프레임들을 다시 하나의 메시지 리스트로 만드는 작업이 필요하다. 왜냐하면 이렇게 다시 재구성해야만 곡의 시간정보가 유지되기 때문이다. 그런 다음 전체 메시지 스트림을 다시 트랙별 MessageList로 나누어야한다. 이것은 MIDI 파일 형식이 MIDI 데이터를 트랙별로 순차적으로 저장하기 때문이다. 그런 다음 각 메시지의 절대시간을 메시지들 사이의 상대시간으로 변환시키고, 이를 다시 Variable Length Quantity라는 비트 패턴으로 바꾼다. 이 패턴은 MIDI 파일에서 시간 정보와 MIDI 데이터를 구별하기 위해 고안된 것으로 Standard MIDI File Format에 정의되어 있다.

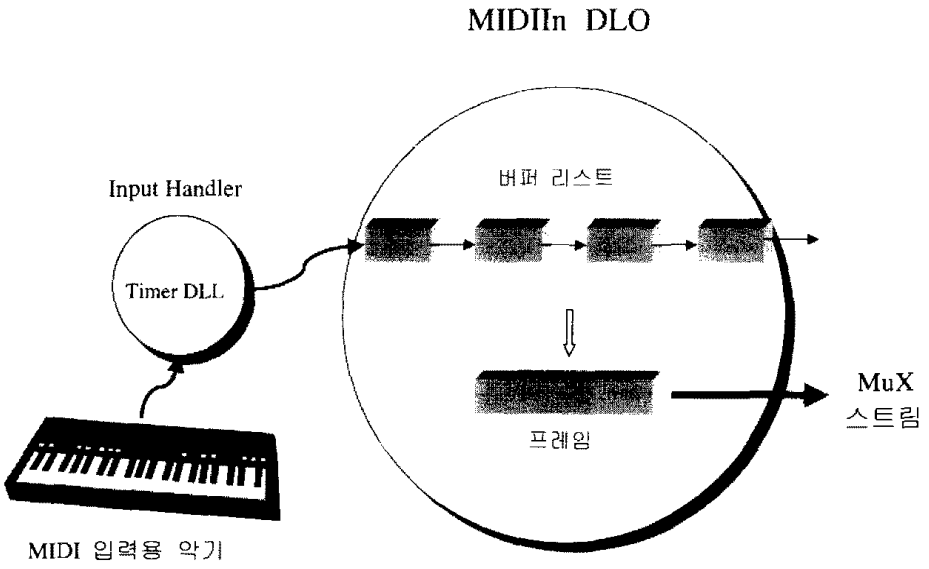
4.3 MIDI 입력 DLO

MIDI 입력 부분을 담당하는 MidiIn 클래스에서는 전자악기로부터 입력되어지는 MIDI 데이터 처리를 위해 두 가지의 Callback 함수가 필요하다. 하나는 MIDI 입력 디바이스에서 발생하는 메시지를 처리하기 위한 것이고, 다른 하나는 MIDI 출력 디바이스용이다. MIDI 출력 디바이스 Callback 함수가 필요한 이유는 입력된 데이터를 저장하기 전에 확인을 위해 MIDI 출력 디바이스에 출력하기 때문이다[8,13].

MIDI 입력 디바이스 Callback 함수에서는 입력된 MIDI 데이터를 앞서 MidiFileIn 클래스를 설명할 때 사용한 VOICE 구조체를 사용하여 MIDI 데이터를 문자배열에 저장하는 일을 한다. 이를 가변 크기의 프레임으로 만들어 MuX 스트림으로 보내는 일은 MidiIn 클래스에서 담당한다. 프레임의 크기가 가변인 이유는 프레임을 DLO 사이에 전달하는 MuX 서버의 메커니즘이 일정 시간마다 동작을 하고, 그래서 일정 주기마다 사용자의 입력을 저장하였다가 프레임으로 만들기 때문이다. 이러한 이유로 네트워크로 연결되어 있는 원격 호스트 사이에서는 약간의 지연시간이 생길 수 있다. 그리고, Callback 함수는 반드시 DLL에 존재해야 하므로 MIDI 입력 디바이스 관리용 DLL과 MIDI 출



(그림 12) MIDI 파일 출력 DLO
(Fig. 4) DLO for MIDI file output



(그림 13) MIDI 입력 DLO
(Fig. 5) DLO for MIDI input

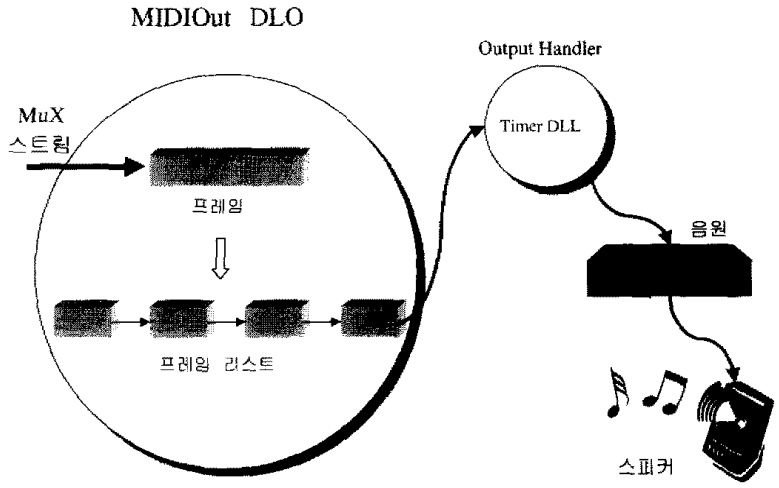
력 디바이스 관리용 DLL을 따로 구현한다.

입력된 MIDI 데이터들을 가지고 MIDI 파일을 만들기 위해서는 반드시 필요한 템포를 나타내는 메타 이벤트와 트랙의 끝을 나타내는 메타 이벤트를 시간상으로 MIDI 데이터의 처음과 끝에 넣을 수 있게 메타 이벤트를 만드는 일도 MidiIn 클래스에서 담당하게 된다. 그리고, 사용자가 연주하는 곡의 빠르기를 결정하는 템포값은 MuX 서버의 클라이언트 프로그램에서 입력을 받는데, 이는 이 클래스의 사용되지 않는 생성자의 인수를 사용한다.

4.4 MIDI 출력 DLO

MIDI 출력 부분을 담당하는 MidiOut 클래스에서는 정확한 시간에 MIDI 출력 디바이스로 MIDI 메시지를 보내기 위해서 Win32 API가 제공하는 timer를 사용한다[4]. 이 timer 함수는 지정된 시간 간격마다 WM_TIMER 메시지를 윈도우에 보내도록 하여 이 메시지를 받으면 수행되는 callback 함수를 지정하도록 되어있다. 또 callback 함수로 사용자가 원하는 데이터를 보낼 수 있도록 DWORD 크기의 파라미터를 제공한다.

윈도우의 타이머는 millisecond 단위로 시간 간격이 설정되는 반면 MIDI delta-time은 microsecond 단위로 표현되므로 어느 정도의 보정이 필요하다. MidiOut 클래스는 MIDI 데이터를 출력하기 위해 두 가지의 Callback 함수가 필요하다. 하나는 MIDI 출력 디바이스에서 발생하는 메시지를 처리하기 위한 것이고, 이를 위해 MIDI출력 디바이스 관리용 DLL로써 MidiIn DLO를 구현할 때 구현한 것을 재사용하였다. 또 다른 하나는 timer에서 지정되는 Callback 함수인데 이는 MIDI 데이터를 MIDI 출력 디바이스에 정확한 시간에 보내는 작업을 수행하게 되고, 이를 timer DLL로 구현하였다. timer DLL에는 TimeFunc()이라는 Callback 함수가 존재하는데 이 함수가 하는 일은 시간 카운터를 두어 함수가 불려질 때마다 증가시키고 이 값이 메시지의 시간과 동일하면, 그 때 그 메시지의 종류에 따라 처리한다. 그리고 메시지를 모두 처리한 후에는 MIDI 출력 디바이스를 닫고 timer를 중지시키는 일을 한다.



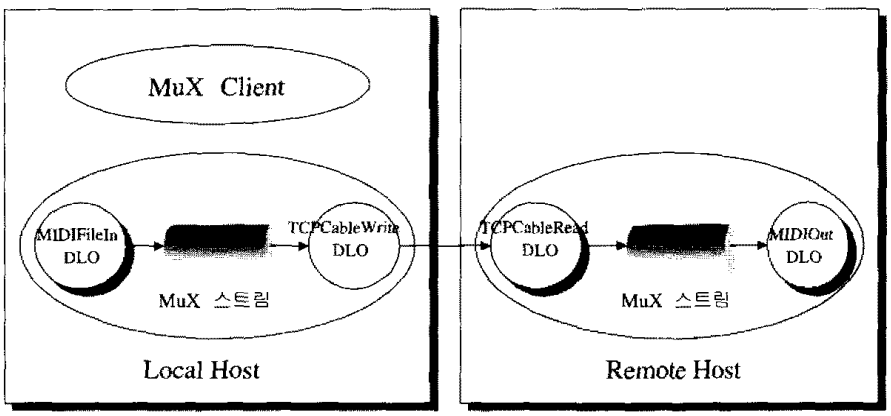
(그림 14) MIDI 출력 DLO
(Fig. 6) DLO for MIDI output

MidiOut 클래스에서는 MuX 스트림으로부터 전달받은 프레임들을 모아서 FrameList라는 linked list 형식의 클래스의 객체로 저장한다. 그리고 이를 timer DLL에 넘겨주는 작업을 수행한다. 이 때 만약 MIDI 데이터의 재생시간보다 프레임이 지연되어 이 DLO에 전송된다면 심각한 문제가 발생하게 되는데 이를 본 논문에서는 처음 MIDI 데이터 재생을 시작할 때 미리 몇 개의 프레임들을 받은 후 timer를 시작시키는 일종의 prefetching 방식으로 해결했다. 여기에서 고려할만한 인수들로써 prefetching시켜야 하는 프레임의 수와 하나의 프레임의 크기가 있다. 본 논문에서는 반복적 실험을 통해 이들 값을 구했다.

5. 실 험

5.1 네트워크 상에서의 MIDI 파일재생

로컬 호스트에 있는 MIDI 파일을 네트워크로 전송하여 원격 호스트의 MIDI음원과 스피커를 통해 소리를 내는 예제이다[11]. 다음의 그림 7은 이 실험의 개념도이다.



(그림 15) 네트워크상의 MIDI 파일 재생
(Fig. 7) MIDI file playback on the network

5.2 실험 환경 및 결과

본 논문에서 설계하고 구현한 분산 MIDI 인터페이스는 Windows NT 4.0환경에서 MuX Windows NT 버전을 사용하고, Microsoft Visual C++ 버전 2.0 언어로 구현하였다. 컴퓨터와 MIDI 디바이스간의 하드웨어적인 인터페이스들로 MIDI 디바이스 인터페이스용 카드인 MPU-401, 입력용 전자 악기로 Rorland M1 Synthesizer, 실제음을 소리내기 위해 MIDI가 지원하는 128개 음의 샘플들과 percussion 소리들을 저장하고 있는 음원 모듈 Rorland Sound Canvas를 사용했다.

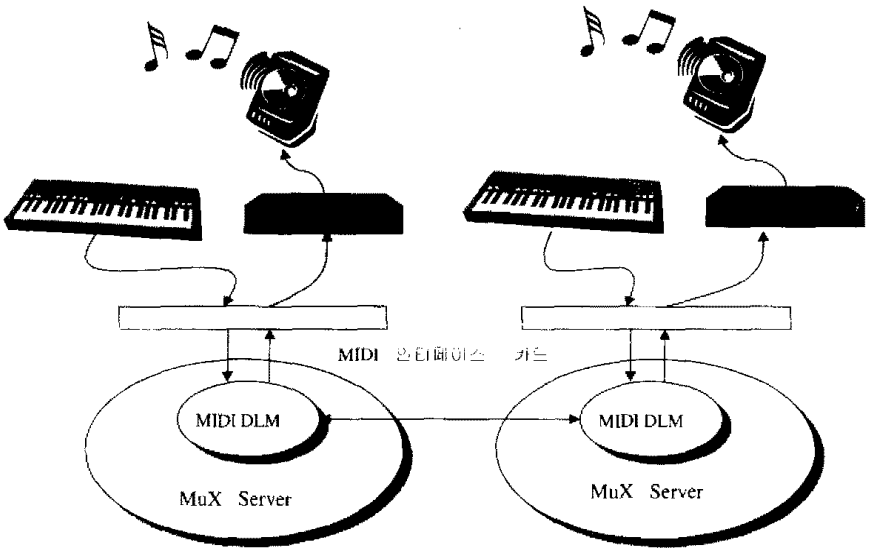
실험은 LAN 환경에서 단일 미디 스트림의 입출력을 대상으로 MIDI 파일 입출력, MIDI 디바이스 입출력을 수행했으며, 그 결과 사람이 느낄 수 있는 지연은 전혀 없이 자연스럽게 연주됨을 확인하였다.

6. 결 론

본 논문에서는 MIDI 파일로부터 MIDI 메시지를 입력받거나 MIDI 메시지를 MIDI 파일로 저장하는 MIDIFile DLM과 신디사이저와 같은 MIDI 디바이스로부터 MIDI 메시지를 입력하거나 MIDI 메시지를 음원장치로 출력하는 MIDI DLM을 구현하였다. MuX Windows NT 버전의 디바이스 인터페이스 메커니즘에 따라 구현된 MIDI 파일 지원을 위한 MIDIFile DLM에는 MIDIFileIn DLO와 MIDIFileOut DLO가 포함되고, MIDI DLM에는 MIDIIn와 MIDIOut DLO가 포함되어 있다. 그리고, MuX API 프로그램을 작성하여 구현한 DLM의 4개의 DLO를 테스트하여 정상대로 수행됨을 확인하였다.

본 논문의 결과로 구현된 MIDI DLM들를 여러 응용분야에 활용하는 방안과 기대되는 성과는 다음과 같다. 첫째, MuX의 입출력 기능을 강화하는 MIDI 디바이스 인터페이스를 구현하여 좀더 포괄적인 의미의 입출력 서버로서의 MuX의 활용을 기대할 수 있다. 둘째, MIDI 인터페이스를 이용하여 오디오와 관련된 새로운 응용 프로그램의 개발이 가능해진다. 셋째, MIDI를 이용한 응용 프로그램으로는 MIDI 악기를 이용하여 악기 연주를 가르치는 원격 음악 교육, 작곡 편곡 프로그램, 원거리 뮤지션들이 한자리에 모이지 않고 MuX의 MIDI 디바이스 인터페이스를 이용하여 연주하고 녹음하는 가상 스튜디오를 만들 수 있으며, 또한 웹상에서 운영되는 음악에 관한 여러 응용문제를 개발할 수 있을 것으로 기대된다.

이러한 여러 가지 응용을 위해서는 현재 단일 MIDI 스트림을 확장하여 여러 개의 MIDI 스트림을 통합하는 믹서의 구현이 요구되며, WAN 상에서 생길 수 있는 데이터 지연을 해결하는 연구가 필요하다.



(그림 8) MIDI 하드웨어 환경
(Fig. 8) MIDI hardware environment

참 고 문 헌

[1] Michael boom, Music through MIDI, Microsoft Press, 1988.

[2] International MIDI association, "The Standard MIDI File(SMF) specification", 1988.

[3] E. A. Fox, "Standard and Emergence of Digital Multimedia Systems," Communication of the ACM, Vol.34, No.4, 1991.

[4] Microsoft, Multimedia Programmer's Reference, Microsoft Press, 1991.

[5] 조동희, 이영보, 오!미디?, 성안당, 1993.

[6] Steve Cunningham, Multimedia system, AddisonWesley, 1994.

[7] 임영환, 김두현, MuX Multimedia I/O Server, ETRI Presentation Material, 1994.

[8] Steve Rimmer, Advanced multimedia programming, McGraw-Hill, 1995.

[9] 임영환, 김두현, MuX : 분산 멀티미디어 처리 모델, ETRI Report, Feb. 1995.

[10] 임영환, 김두현, MuX User's Manual, MuX User's Group, 1995.

[11] 임영환, 김두현, Application program interface for MuX, ETRI Report, Feb. 1995.

[12] Doo-Hyun 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.

[13] Mark Andrews, Visual C++ and Windows NT 4.0 Programming, M&T Books, 1997.

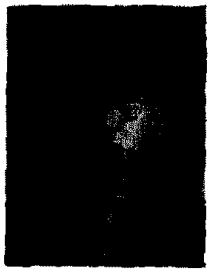
조 병 호

1995년 경북대학교 컴퓨터공학과 졸업(공학사)

1997년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사)

1997년~현재 경북대학교 대학원 컴퓨터공학과 박사과정

관심분야 : 분산 멀티미디어, 멀티미디어 DBMS



강 태 진

1996년 경북대학교 컴퓨터공학과 졸업(공학사)

1998년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사)

1998년~현재 (주)MJL 연구원

관심분야 : 분산 멀티미디어, 분산 시스템

유 기 영

1976년 경북대학교 수학교육과 졸업(이학사)

1978년 한국과학기술원 전산학과 졸업(공학석사)

1992년 Rensselaer Polytechnic Institute 졸업(이학박사)

1978년~현재 경북대학교 컴퓨터공학과 교수

관심분야 : 병렬 및 분산처리, 병렬 컴파일러, 어레이 프로세서 설계

이 경 희

1990년 경북대학교 컴퓨터공학과 졸업(공학사)

1992년 경북대학교 대학원 컴퓨터공학과 졸업(공학석사)

1992년~현재 한국전자통신연구원 연구원

관심분야 : 분산멀티미디어, 실시간 미디어 처리 등

궁 상 환

1977년 숭실대학교 전산학과 졸업

1977년~1983년 육군 제2군수지원사령부 전산 장교

1983년 고려대학교 경영대학원 전산정보 석사

1998년 충북대학교 전자계산학과 졸업(이학박사)

1982년~현재 한국전자통신연구원 책임연구원

관심분야 : 멀티캐스트 트랜스포트, 분산시스템아키텍춰 등