멀티미디어 입출력 서버를 위한 오디오 변환 필터

(Audio Transformation Filter for Multimedia I/O Server)

조 병 호 † 장 유 탁 † 김 우 진 †† 김 기 종 ††† 유 기 영 ††††

(Byoung Ho Cho)(Yu Tak Jang)(Woo-Jin Kim)(Ki Jong Kim)(Ki-Young Yoo)

요 약 본 논문에서는 음성 입력으로 받아들인 멜로디를 MIDI 데이타로 변환하는 필터의 설계 방법과 분산 멀티미디어 환경에서 동작하는 입출력 서버 시스템인 MuX 환경에 적용하는 방법에 관해 기술한다. MuX는 다양한 입출력 디바이스와의 인터페이스를 위해 장치 독립적인 DLM(Dynamic Linking Module)을 사용하는데, 현재 MuX 시스템의 입출력 디바이스 인터페이스로 사용되는 웨이브 형식의 오디오 DLM과 MIDI(Musical Instrument Digital Interface) DLM의 기능을 보강하기 위해서 사람의 음성을 MIDI 데이타로 변환해주는 필터를 설계하고 구현하였다. MIDI 데이타의 입력 방식이 파일이나 MIDI 악기 외에도 사람의 음성 데이타로도 가능하므로 악기 연주에 익숙하지 않은 사람들도 MIDI 데이타를 입력할 수 있고, 미디어의 표현력이 증가되어 다양한 응용에 활용될 수 있다.

Abstract In this paper, we present a design method of a digital filter converting humming voice melody into MIDI data and a method of adapting it to a distributed multimedia I/O server. MuX uses device-independent DLMs(Dynamic Linking Module) for the interface with various I/O devices, and has wave-form audio DLM and MIDI DLM for audio interfaces. In order to expand the audio device interfacing ability of MuX system, we have designed and implemented a filter transforming human voice into MIDI messages. As the methods to input MIDI data are expanded to human voice in addition to MIDI files and MIDI instrument, someone who is not good at playing instruments can also generate the MIDI data, which enables our media interfaces to be used in various applications.

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[1,2,3].

† 학생회원 : 경북대학교 컴퓨터공학과
bhcho@purple.knu.ac.kr
asphalt@purple.kyungpook.ac.kr
†† 비 회 원 : 영진전문대학 컴퓨터정보기술계열 교수
wjkim@yeungjin.ac.kr
††† 정 회 원 : 영진전문대학 컴퓨터정보기술계열 교수
kjkim@yeungjin.ac.kr
†††† 종신회원 : 경북대학교 컴퓨터공학과 교수
yook@bh.kyungpook.ac.kr
논문접수 1999년 9월 20일
심사완료 2000년 8월 18일

초기의 MuX 시스템에서 사용 가능한 입출력 디바이스는 입력용으로 마이크와 카메라, 출력용으로 스피커와 모니터 정도로 상당히 제한되어 기본적인 입출력 디바이스만을 지원하였고 다양한 입출력 디바이스의 추가, 확장을 쉽게 하기 위해서 DLM(Dynamic Linking Module)을 사용하여 디바이스 독립적인 인터페이스를 제공하였다. 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[4,5].

MuX 시스템이 좀더 포괄적인 의미의 멀티미디어 데이타 입출력을 수행하기 위해서는 여러 다른 입출력 디바이스와의 인터페이스를 추가, 확장하는 것이 요구되어지는 실정이었다[3,6,7]. 이에, 입출력 디바이스의 확장

으로 MIDI를 지원하는 기기와의 입출력 인터페이스, MIDI 파일의 입출력을 수행하는 인터페이스를 개발한 바 있다[8]. 이로서, 사용자들은 MIDI를 지원하는 전자 악기를 MuX상에서 사용할 수 있게 되고, 자신의 연주를 원거리에 있는 다른 사람에게 실시간에 보내거나 파일로 저장하였다가 다시 재생할 수 있어 오디오 측면의 풍부한 표현력을 제공받게 되었다. 하지만, MIDI 데이타를 입력하기 위해서는 사용자가 MIDI 악기를 연주하거나 혹은 시퀀서 프로그램을 이용해서 MIDI 데이타를 생성해야 하므로 일반인들이 사용하기에는 어려움이 있다.

본 논문에서는 사용자가 마이크를 통해서 음성으로 멜로디를 입력할 수 있도록 하기 위해서 음성 데이타를 MIDI 데이타로 변환해주는 필터를 설계하고 MuX 환경에서 사용할 수 있도록 새로운 입력 DLM을 추가로 구현함으로서 시스템의 디바이스 인터페이스를 확장하였다. 마이크로부터 입력된 음성 데이타의 피치를 추출하고 잡음이 있는 음정을 보정한 후, 이를 MIDI 데이타로 변환하고 MuX 시스템의 디바이스 인터페이스에 전달하도록 하였다.

이 과정에서 가장 중요한 부분은 입력된 음성 데이타로부터 피치를 검출하는 알고리즘으로 음성 처리시스템에서 필수적인 요소이다[9]. 이는 지금까지도 많은 연구가 진행되고 있으나 잡음이 있는 환경, 대역폭이 제한된 경우, 주파수의 위상특성이 변질된 경우 등과 같은 복잡한 상황에서는 신뢰할 만한 결과를 얻지 못하고 있는 실정이다. 크게 파형의 표본화를 이용하는 시간 영역 검출과 진폭이나 위상 주파수표를 이용하는 주파수 영역, 그리고 시간과 주파수 영역을 같이 이용하는 방법의 세 가지로 나뉘어진다[10,11,12]. 본 논문에서는 MuX의 분산 환경에 적용하기 위해 빠른 수행시간을 필요로 하므로 알고리즘이 대체적으로 간단하고, 실행시간도 빠른 시간 영역 검출법 중의 평균차 함수법(Average Magnitude Difference Function)을 이용하였다[13].

서론에 이어서 2장에서 MuX에서의 디바이스 인터페이스 기술을 소개하고, 3장에서 음성 데이타에서 피치를 추출하는 방법을 설명하고, 4장에서는 추출된 피치를 MIDI 데이타로 변환하는 필터의 설계와 MuX와의 인터페이스 DLM의 구현에 관해 기술하며, 5장에서 실험결과를 보인 후에, 6장에서 결론을 맺는다.

2. 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[4,5].

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 서버 내에 등록된 DLL(Dynamic Linking Library)들로부터 DLM들을 생성하여 DLM 테이블에 등록하고 관리한다. Dlo 클래스는 멀티미디어 입/출력 기능을 가지는 기본 객체로서 DLL내의 모든 객체는 Dlo로부터 상속된다. DLO 객체의 기본적인 속성은 DloAttribute 구조체에 저장된다. Medium 클래스는 멀티미디어 데이타를 MuX와 디바이스, 네트워크 또는 파일 시스템간에 입출력하는 객체의 클래스로 입력 또는 출력을 하나 가진다.

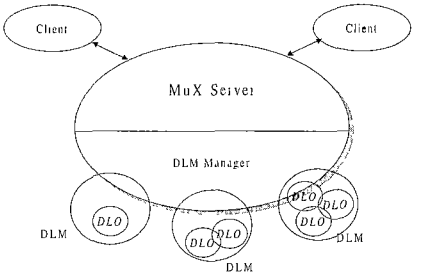

그림 1 MuX의 디바이스 인터페이스

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 위해

서는 Medium 객체로부터 상속받는 새로운 클래스와 DLM을 위한 인터페이스 함수를 포함하는 DLL을 만들어 MuX에 등록하면 된다.

3. 오디오 변환 필터

오디오 변환필터는 입력된 음성 데이타(Wave 포맷)를 MIDI 메시지 형태로 변환하며, 이를 위해 그림 2와 같이 피치추출, 음정인식, 음표그룹화, 미디 메시지변환의 4가지 단계를 수행한다. 피치 추출기는 입력된 음성 신호를 작은 프레임으로 자른 다음, 각 프레임의 주기를 추출하고, 음정인식기는 추출된 각 주기에 해당되는 옥타브와 음정을 주파수 테이블을 참조해서 출력한다. 음표 그룹화에서는 옥타브와 음정의 쌍들을 그룹화하고, MIDI 메시지 변환기에서는 그룹화된 옥타브, 음정 쌍에 대해서 그룹의 길이를 고려하여 음정의 길이를 계산하고 MIDI메시지로 변환한다.

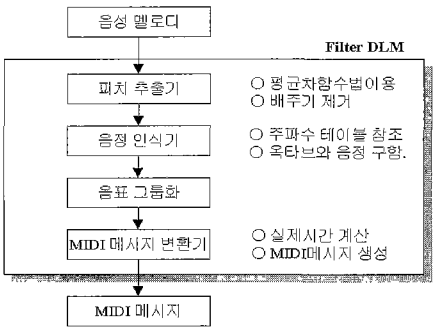

그림 2 오디오 변환기

이런 과정을 거쳐 음성 오디오 입력이 미디 메시지로 변환되는데, 이 변환가를 MuX의 입력 DLO인 MIDI Filter DLO로 구현하고 이것을 관리하는 Filter DLM을 설계, 구현하였다. 기존의 MIDI 관련 DLM과 Filter DLM의 관계는 그림 3과 같다.

3.1 피치 추출기(Pitch Detector)

음성신호는 일정 간격을 두고 서로 유사한 위상값을 가지게 되는데 이를 기본주기라 한다. 기본주기가 작을수록 파형의 반복이 자주 일어나고 높은 소리를 내며 피치가 높다. 반면, 기본주기가 클수록 반복이 적게 일어나고 낮은 소리를 내며 피치가 낮다.

피치 추출기는 입력된 음성 신호를 분석하여 피치 값들을 출력한다. 피치를 추출하는 방법은 MuX의 분산 환경에서 빠른 수행시간을 필요로 하므로 알고리즘이 대체적으로 간단하고, 실행시간도 빠른 시간 영역 검출법 중의 평균차 함수법(Average Magnitude Difference Function)을 이용하였다[13]. 음성 파형들을 일정한 길이의 프레임으로 자른 다음, 각 프레임에 대해 평균차 함수법을 사용한다. 그리고, 좀 더 정확한 결과를 위해서 반주기, 배주기들을 보정한다.

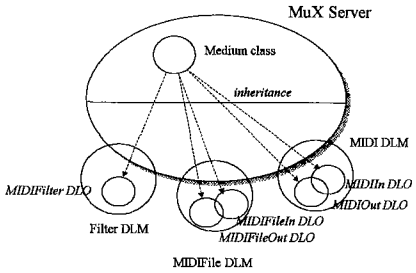

그림 3 Filter DLM

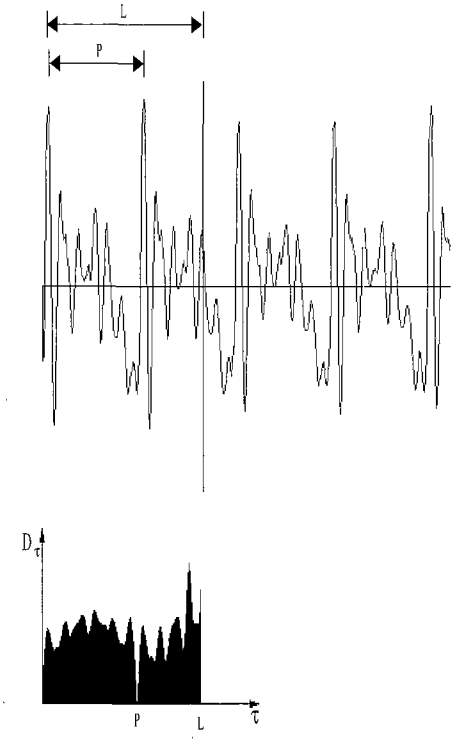

그림 4 평균차 함수법

3.2 평균차 함수법

시간영역에서의 피치 추출법들은 일반적으로 음성 파

형의 주기성을 강조하고 나서 결정논리에 의해 주기성을 검출한다. 평균차 함수법은 각 주기에 대해서 원래 신호와 임의의 주기만큼 떨어진 신호와의 차들을 모두 더한 후 평균값을 내고, 그 중 가장 작은 임의의 주기를 그 창에서의 주기로 가정한다.

L을 프레임의 크기라 하고, S_j는 음성표본의 값이라고 하면, 평균차 함수법은 다음과 같이 구성된다[13].

$$D_\tau = \frac{1}{L-\tau}\sum_{j=1}^{L-\tau}|S_j - S_{j+\tau}|, \qquad \tau = 1, \ldots, \tau_{MAX}$$

만약 τ가 주기 P라면 D_τ는 그 위치에서 그림 4에서와 같이 극소값이 나올 것이다.

3.3 배주기 보정

위의 평균차 함수법에서는 τ가 실제 주기 P의 배수 즉, $\tau = 2P, 3P, \ldots$ 등의 위치에 있더라도 D_τ가 극소값이 나올수 있다(그림 5). 그래서 다음과 같이 배주기를 실제 주기로 보정한다. 평균차 함수법에서 극소값 D_τ가 나왔다면, 그 τ가 배주기인지를 알기 위해서 그 τ를 근사주기 p′ 라 두고 $\frac{p'}{2}$, $\frac{p'}{3}$, $\frac{p'}{4}$ .... 의 주기에 대해서 D 값 즉, $D_{p'/2}$., $D_{p'/3}$,....와 D_τ 를 비교하여 문턱값보다 작다면 그 주기를 참주기 P 라 둔다. 의사 코드는 다음과 같다.

if $D_i - D_\tau < T$ *then* $P \leftarrow i$, $i = p'$, $\frac{p'}{2}$, $\frac{p'}{3}$, $\frac{p'}{4}$...., P_{min}

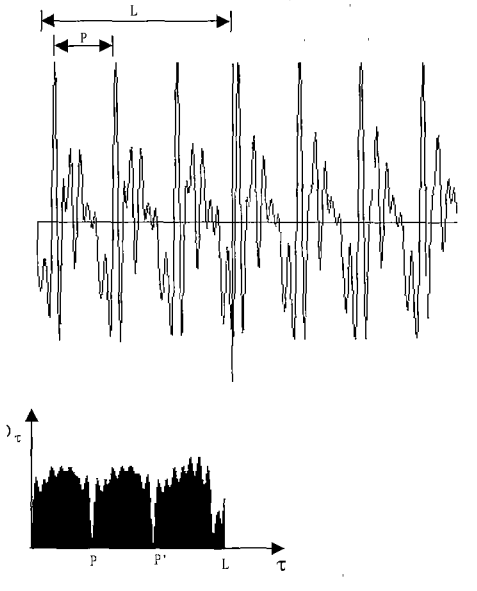

그림 5 배주기가 나오는 경우

3.4 잡음 검출(Noise Detection)

해당 프레임에 음성신호가 0 에 가깝거나, 음의 시작이나 끝에서처럼 신호가 불안하면 평균차 함수법의 결과를 그대로 이용한다면 오류가 발생한다. 본 연구에서는 이를 잡음이라고 간주하고 해당 주기를 null이라고 둔다.

첫 번째, 해당 프레임의 신호가 미약하거나 잡음만이 낀 경우에는 평균차 함수법을 이용하기 전에 해당 프레임내의 신호들의 합을 구하여 문턱값보다 적은 경우를 잡음으로 간주한다. 두 번째, 신호가 불안한 경우 평균차 함수법을 이용한 결과 극소값이 뚜렷하게 나오지 않거나, 주기가 아주 작게 나올 수 있다. 이 경우에는 주기의 최소값을 두어 이보다 적을 경우 잡음으로 간주한다.

3.5 음정 인식기(Note Identifier)

음정인식기는 추출된 각 주기에 대해 해당되는 옥타브와 음정을 주파수 테이블을 참조해서 출력한다. 각 음정은 표 1과 같이 고유한 주파수를 가진다. 주파수는 주기와 역수의 관계를 가지므로, 앞 단계의 피치 추출기에서 검출된 주기들의 집합에 대해서 다음과 같이 주파수를 구한다.

$$f = \frac{sampling\ rate}{\tau}$$

구해진 주파수들은 표 1과 비교 연산하여 가장 근사치인 옥타브와 음정을 구할 수 있다. 옥타브와 음정을 합쳐서 C3(세 번째 옥타브의 '도')와 같이 표현한다. 그리고, 잡음으로 처리되어 주기가 null인 경우에 대해서는 주파수도 null로 간주한다. 예를 들어, sampling rate가 11025이고 τ 의 집합이 {50, 51, 51, 43, 42} 일 때, f 는 각각 220.5, 216.17, 216.17, 256.39, 262.5 가 되며, 해당되는 옥타브와 음정은 A3, A3, A3, C4, C4 이다.

표 1 음정의 표준주파수표

옥타브	음정	주파수(Hz)	옥타브	음정	주파수(Hz)
1	C	32.703	3	C	130.81
	D	38.708		D	140.83
	E	42.202		E	164.81
	F	43.654		F	174.81
	G	46.998		G	196.00
	A	55.000		A	220.00
	B	61.739		B	246.94
2	C	65.400	4	C	261.60
	D	73.415		D	293.66
	E	82.407		E	329.83
	F	87.307		F	349.23
	G	97.999		G	392.00
	A	110.00		A	449.00
	B	123.43		B	493.92

3.6 음표 그룹화(Note Grouping)

음표그룹화는 위 단계에서 출력된 옥타브, 음정의 집합에 대해서 유사정도를 고려하여 그룹화하여 옥타브/음정과 길이의 쌍으로 표현한다. 피치추출, 음정인식을 통해 프레임당 하나의 음정이 발생하므로, (총녹음시간 × 샘플링비 / 프레임의 길이) 만큼의 음정들의 집합이 생성된다. 실제로 음표 그룹화 부분의 결정논리들을 강화한다면 앞 단계의 출력이 그리 좋지 않더라도 smoothing을 통해 괜찮은 출력으로 바뀔 수 있다.

본 연구의 그룹화 과정에서는 먼저 인접한 같은 음정들은 하나로 묶어 주는데 이 과정에서 잡음(null 값)도 하나의 음정으로 간주하여 그룹화한다. 이런 다음 보정과정을 거치는데 먼저, 아주 짧은 시간에 음정의 변화가 생길 수 없다는 가정에 바탕으로 길이의 문턱값을 두어 이보다 작다면 주위 음정으로 통합한다. 그리고, 음의 시작이나 음의 끝에서는 신호가 불안하기 때문에 구해진 음정이 틀릴 확률이 높다. 그러므로, null 값이 그룹화 되어지고 난 다음의 음정은 null로 간주하고, null 값의 그룹이 나오기 전의 음정은 바로 이전 음정으로 간주하여 처리한다. 음표 그룹화 알고리즘은 다음과 같다.

단계 1. 연속된 동일한 음정들을 음정과 길이의 쌍 $\{ N_i, l_i \}$으로 변환

단계 2. if $l_i < threshold$ 이면,

if N_{i-1} == N_{i+1} 이면, $N_i = N_{i-1}$

else

if N_{i-1} == null 이면, $N_i = N_{i+1}$

else $N_i = N_{i-1}$

end if.

end if.

end if.

예) 만약 길이의 문턱값이 3이라면, 아래의 음정 집합은 다음과 같이 그룹화된다.

null null null null E5 C4 C4 C4 C4 C4 E2 E2 C4 C4 C4 C4 F5 null null null null

→ {null, 4}, {E5, 1}, {C4, 5}, {E2, 2}, {C4, 4}, {E5, 1}, {null, 4}

→ {null, 4}, {C4, 13}, {null, 4}

4. 미디 메시지 변환기(MIDI message transformer)

그룹화된 각 음표그룹에 대해서 실제시간을 계산해서 시퀀서가 이용할 수 있도록 미디 메시지와 시간정보를 포함한 미디 이벤트들로 변환한다.

4.1 미디메시지 소개

표준 미디 파일 규격(SMF Standard MIDI File)은 시퀀서(sequencer)가 녹음하거나 연주하는 미디 데이타를 저장하기 위해 특별히 고안된 것이다[14]. 이 규격은 상태 바이트와 적절한 데이타 바이트로 이루어진 표준 미디 메시지와 이 메시지를 연주하기 전에 어느 정도 클럭 펄스(clock pulse)를 기다리는지를 표시하는 시간정보, 빠르기, 분해능(PPQN or SMPTE), 박자와 음정 표시 및 트랙과 패턴 이름 등의 다양한 정보들을 저장하는 형식을 나타내준다. 여기에서 사용하는 데이타 구조는 8비트 데이타 구조이다. 이 규격에서는 8비트 데이타 구조에 무엇이 들어가야 하는지를 지정한다.

미디 메시지는 크게 두 종류로 나누어진다[15]. 하나는 채널 메시지(channel message)이고, 또 하나는 시스템 메시지(system message)이다. 채널 메시지는 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 익스클루시브 메시지(system exclusive message)로 구성된다. 본 연구에서는 음 출력을 위해서 사용하는 채널 보이스 메시지(표 2)만 생성한다. 이는 음의 ON-OFF와 음색의 변화 및 악기에 대한 연주 정보의 전송에 관계되는 신호이다.

그림 6은 시퀀서에게 보내지는 메시지 이벤트를 나타낸 것으로, 실제 미디 메시지에 델타타임(delta time)이라는 타임 스탬프를 붙여서 보낸다. 시퀀서는 델타타임이 지난 다음 상태 바이트와 데이타 바이트를 읽어서 메시지를 처리한다.

표 2 채널 보이스 메시지

역할	상태 바이트	데이타 바이트	데이타 바이트
음의 ON	9c	n	v
음의 OFF	8c	n	v
c : 채널번호(0~15) n : 노트번호(0~127) v : 세기(0~127)			

델타타임	상태바이트	데이타바이트

그림 6 미디 이벤트 형식

4.2 MIDI의 시간 표현 방법

MIDI 파일에서는 각 이벤트의 연속적인 시간 관계를 델타타임(delta time)을 통해서 표현한다. 하나의 이벤트는 이전 이벤트와의 시간 차이와 실질적인 이벤트정보로 구성되는데, 이전 이벤트와의 시간 차이는 델타타임

으로 표현되고 실질적인 이벤트는 MIDI 메시지가 된다. 이때, 델타타임의 단위시간은 header chunk에 있는 division 2바이트와 메타 이벤트로 표현되는 3바이트의 tempo정보로 결정된다.

4.2.1 PPQN(Pulse Per Quater Note)

델타타임의 단위시간을 4분 음표 길이의 상대적인 값으로 표현하는 방법이다. 이는 header chunk의 division값 중에서 첫 번째 바이트가 양수일 때에 해당하고 이때 division의 값은 4분음표 하나가 가지는 delta time 값이다. 따라서 나머지 모든 델타타임은 이 값에 상대적인 값이 된다. 예를 들어서 division이 16진수로 0060(hex)이라면 4분음표 하나가 96 펄스를 가진다는 것이다. 만약 델타타임이 0030(hex)이라면 4분음표의 반쪽, 8분음표 길이만큼의 시간이 되는 것이다. 여기서, 이 델타타임의 절대시간은 tempo와 결부되어 결정된다.

Tempo는 메타 이벤트 3바이트로 표현되고 4분음표의 길이를 microsecond 단위로 표현한다. PPQN인 경우에 델타타임의 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 delta time의 한 단위시간은 0.5초 / 96이 된다.

4.2.2 SMPTE(Society of Motion Picture & Television Engineers) 코드

4분음표 길이의 상대적인 값을 표현하는 PPQN과는 달리 SMPTE 방법은 델타타임의 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 델타타임의 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.

4.2.3 델타타임 계산

델타타임의 단위시간이 정해졌다면, 각 그룹의 실제 시간은 다음과 같이 계산되어진다.

$$\text{델타타임} = \left(\frac{\text{프레임의크기}}{\text{샘플링비}} \times \text{그룹의길이}\right) / \text{델타타임의 단위시간}$$

4.3 미디 이벤트 변환

앞에서 언급한 것처럼 채널 보이스 메시지는 음의 ON-OFF와 음색의 변화 및 악기에 대한 연주 정보의 전송에 관계되는 신호이다. 음의 ON-OFF는 피아노 건반에 적용하면, 건반을 눌렀을 때가 ON 상태이고, 떼었을 경우가 OFF상태이다. 표 2에서 노트번호는 피아노 건반의 번호라고 보면 되고, 세기는 건반을 누르는 세기에 해당한다. 실제 미디 메시지를 만들 때는 OFF메시지를 세기를 0으로 한 ON메시지로 대신한다.

본 연구에서는 채널과 세기는 하나로 고정하고, 변하는 노트번호를 옥타브와 음정 쌍에 매핑시킨다. 앞 단계 에서 구한 음정과 길이 쌍을 $\{ N_i, l_i \}$ 이라 두면, 집합 $\{ N_i, l_i \}$ 을 미디 이벤트로 변환하는 방법은 다음과 같다.

1) N_i이 null 인 쌍에 대해서는 미디이벤트를 생성하지 않는다.
2) 맨 처음 쌍인 $\{ N_0, l_0 \}$은 델타타임을 0으로 하여 미디 메시지를 생성한다.
3) 각 $\{ N_i, l_i \}$ 에 대해서, N을 해당 노트번호로 바꾸고, l_{i-1}을 델타타임으로 변환하여 ON에 해당하는 미디 이벤트를 생성한다.

예) 채널을 1번, 세기를 40h로 고정하고, 한 프레임이 0.01초, 델타타임의 기본단위를 0.001초로 했을 때, {C4, 30}, {D4, 40}, {E4, 20} →

0	91h	3Ch	40h	300	91h	3Eh	40h	400	91h	40h	40h

4) 만약 N_{i+1} 이 null 이거나 i 가 마지막 인덱스라면 , l_i을 델타타임으로 변환하여 음 OFF에 해당하는 미디 이벤트를 생성한다.

예) 채널을 1번, 세기를 40h로 고정하고, 한 프레임이 0.01초, 델타타임의 기본단위를 0.001초로 했을 때, {C4, 30}, {null, 20} ,{D4, 40} →

0	91	3Ch	40h	300	91	3Ch	0h	200	91h	3Eh	40h

5. 실 험

5.1 네트워크 상에서 보이스 멜로디의 MIDI 출력

로컬 호스트에 있는 마이크로 입력된 음성 멜로디를 MIDI 메시지로 변환하고 네트워크로 전송하여 원격 호스트의 MIDI음원과 스피커를 통해 소리를 내는 예제이다. 다음의 그림 7은 이 실험의 개념도이다.

5.2 실험 환경 및 결과

본 논문에서 설계하고 구현한 오디오 변환 필터는 Windows NT 4.0환경에서 MuX Windows NT 버전을 사용하고, Microsoft Visual C++ 버전 2.0 언어로 구현하였다. 음성 멜로디 입력을 위한 사운드 카드는 사운드 블러스터를 사용하였고 컴퓨터와 MIDI 디바이스 간의 하드웨어적인 인터페이스들로 MIDI 디바이스 인

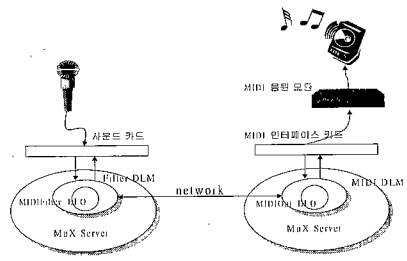

그림 7 네트워크 상의 보이스 오디오의 미디 메시지 변환

터페이스용 카드인 MPU-401, 실제음을 소리내기 위해 MIDI가 지원하는 128개 음의 샘플들과 percussion 소리들을 저장하고 있는 음원 모듈 Rorland Sound Canvas를 사용했다.

음성의 피치 추출과정에서 가장 중요한 것이 배주기의 진위 여부를 판단하는 것인데 본 논문에서는 AMDF의 결과로 나오는 극소값 D_t에 문턱값을 적용해서 배주기를 보정한다. 실험에서 음성녹음을 11016Hz, 8bit로 하기 때문에 AMDF의 결과인 D_t는 0 에서 255 사이의 값을 가지는데 배주기 보정에 필요한 문턱값을 너무 높게 한다면 잘못된 반주기가 선택되어 질 수 있고, 반대로 너무 낮게 한다면 옳은 반주기가 선택이 안되어질 수 있다. 그러므로, 이 값은 실험을 통해서 결정되었으며 8보다 작은 극소값을 가지면 참주기로 본다.

정확한 미디 메시지로 변환하기 위해서는 입력된 음성 신호에서 잡음을 추출하고 제거하는 방법이 사용되어야 하는데 잡음으로 간주되는 두 가지 경우에 대해서 해결책을 제시한다. 첫째, 해당 프레임의 신호가 미약하거나 잡음만으로 구성된 경우를 판단하기 위해서 그 프레임의 신호 값을 모두 더한 값이 10000 이상이면 정확한 피치 추출이 불가능한 잡음으로 간주하고 10 이하이면 거의 소리가 없는 상태로 간주하고 무시한다. 둘째, 프레임 내의 신호가 불안한 경우는 극소값이 뚜렷이 나오지 않거나 주기가 아주 작게 나오므로 주기의 범위를 설정하여 이 범위를 벗어나면 잡음으로 간주한다. 본 논문에서 구하고자 하는 음정의 범위는 보통 사람이 낼 수 있는 소리의 범위, 즉 F2 (87.307Hz)에서 B5 (987.77.Hz)까지이다[16]. 그러므로, sampling rate 가 11016 Hz일 때 나올 수 있는 주기는 sampling rate / 해당 주파수, 즉 11.15(B5)에서 126.18(F2)까지이다. 그래서 AMDF결과 주기가 11 이하이면 B5보다 높은 고음일수 있으나 이를 잡음이라 간주하고 또한 126 이상일 때도 F2보다 저음일 수 있으나 잡음으로 간주한다.

음표의 그룹화 과정에서는 음표 길이의 문턱값을 주어 음 사이의 불안한 신호로 인해 생기는 오류를 보정하였다. 길이의 최소 문턱값은 크기가 클수록 부드러운 음의 진행 결과를 보장해줄 수 있다. 그러나 너무 큰 값은 잘못된 음의 그룹화를 가져올 수 있으므로 샘플링비와 잡음정도를 고려하여 결정하여야 하며, 본 실험에서는 여러 차례 실험을 통해 문턱값을 3으로 정했다. 즉, 음표 그룹화 과정에서 길이가 3 이하인 음은 음 사이의 불안한 신호에서 기인한 것으로 보고 무시한다.

실험은 LAN 환경에서 입력용 오디오 스트림을 변환 필터를 통해 MIDI 스트림으로 바꾸고 이렇게 생성된 단일 MIDI 스트림을 원격의 MIDI 출력 스트림이나 MIDI 파일 출력 스트림으로 연결하도록 하였다. 그 결과 사람이 느낄 수 있는 지연 없이 자연스럽게 연주되고 표준 MIDI 형식의 파일로 저장됨을 확인하였다.

6. 결 론

본 논문에서는 분산 멀티미디어 입출력 서버에서 MIDI 악기의 연주나 시퀀서 프로그램을 사용하지 않고도 MIDI 데이타를 쉽게 입력할 수 있는 메커니즘을 개발하였다. 마이크를 통하여 음성으로 멜로디를 입력하면 이것을 미디 메시지로 변환하여 입력하는 필터 인터페이스를 설계하고 구현하였다.

입력된 음성 멜로디는 웨이브 형태의 디지털 오디오이므로 이것을 프레임 단위로 분해해고 AMDF (Average Magnitude Difference Function) 방법을 사용하여 각 프레임의 피치를 추출하였다. 각 프레임에서 추출된 피치 정보를 이용하여 실질적인 음의 시작과 끝을 구분해내고 정확한 음의 높이를 추출한다. 이 때, 배음의 구분을 위한 알고리즘이 제시된다. 이렇게 추출된 음의 길이와 높이 정보를 이용해서 MIDI 메시지로의 변환과정을 수행한다.

MuX의 디바이스 인터페이스 메커니즘에 따라 Filter DLM을 설계하고 실질적인 변환을 수행하는 MIDI Filter DLO를 구현하여 시스템에 추가함으로서 MuX의 디바이스 인터페이스 기능을 확장하였다. MIDI Filter DLO에서는 웨이브 오디오 인터페이스를 이용해서 음성 데이타를 프레임 단위로 입력받아 피치 정보로의 변환 과정을 수행하고 생성된 MIDI 메시지를 새로운 프레임으로 만들어 돌려준다. 이렇게 구현된 DLM을 이용하여 네트워크 상에서 음성 멜로디를 원격의 호스트의 MIDI 출력 기기로 재생하고, 또 원격의 파일로도 저장하는 실험을 수행하였다.

본 논문의 결과로 구현된 DLM을 여러 응용분야에

활용하는 방안과 기대되는 성과는 다음과 같다. 첫째, MuX의 입출력 기능을 강화하는 MIDI 디바이스 인터페이스의 보강으로 좀더 포괄적인 의미의 입출력 서버로서의 MuX의 활용을 기대할 수 있다. 둘째, MIDI 인터페이스와 편리한 입력 방법을 이용하여 오디오와 관련된 새로운 응용 프로그램의 개발이 가능해진다. 또한, MIDI와 관련된 웹 상에서 운영되는 음악에 관한 여러 응용 프로그램을 개발할 때 편리한 기능을 수행할 수 있을 것이다. 이러한 여러 가지 응용을 위해서는 현재 단일 MIDI 스트림을 확장하여 여러 개의 MIDI 스트림을 통합하는 믹서의 구현이 요구되며, WAN 상에서 생길 수 있는 데이타 지연을 해결하는 연구가 필요하다.

참 고 문 헌

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

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

[3] 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.

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

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

[6] Ghias A., Logan J., Chamberlin D. and Smith B. C., "Query by Humming : Musical Information Retrieval in an Audio Database," Proc. ACM Multimedia '95, 1995.

[7] Shuzo Saito, Kazuo Nakata, *Fundamentals of speech signal processing*, Academinc Press, 1985

[8] 조병호, 강태진, 유기영, "멀티미디어 입출력 서버를 위한 분산 MIDI 인터페이스의 설계 및 구현", 한국정보처리학회 논문지, 5권 7호, 1998.

[9] 배명진, "음성신호의 기본주파수 검출", SCAS Vol. 10, No. 1, pp. 63-69, 1993.

[10] M. Bae, S. Ann, "The High Speed Pitch Extraction of Speech Signals using the Area Comparison Method," KIEE, Vol.22. No.2, pp.101-105, Feb. 1985.

[11] M. Lahat, R. J. Niederjohn, and D. A. Krubsack, "A Spectral Autocorrelation Method for Measurement of the Fundamental Frequency for Noise-corrupted Speech," IEEE Trans., Acoust., Speech, Signal Processing. Vol. ASSP-35, No.6, June. 1987.

[12] M. Bae and S. Ann, "On the Time-Frequency Hybrid Technique for Detectiong the Pitch of Noise Corrupted Speech Signals," J. Acoust. Soc. Korea, Vol.9, No.1, 1990.

[13] Myron J. Ross, Harry L. Shaffer, "Average Magnitude Difference Function Pitch Extractor," IEEE Trans. Acoust., Speech, and Signal Process., Vol. ASSP-22, pp. 353-362, Oct. 1974.

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

[15] Michael Boom, *Music through MIDI*, Microsoft Press, 1988.

[16] 지정규, 오해석, "선율을 이용한 음악정보 검색 시스템의 설계 및 구현", 한국정보처리학회 논문지, 5권 1호, 1998.

조 병 호

1995년 경북대학교 컴퓨터공학과 학사. 1997년 경북대학교 컴퓨터공학과 석사. 1997년 ~ 현재 경북대학교 컴퓨터공학과 박사과정. 관심분야는 멀티미디어 시스템, 지문인식, 분산 시스템, 암호학

장 유 탁

1998년 경북대학교 컴퓨터공학과 학사. 2000년 경북대학교 컴퓨터공학과 석사. 2000년 ~ 현재 (주)닉스테크 소프트웨어 기술 연구소 재직중. 관심분야는 멀티미디어 시스템, JAVA, 암호학, XML

김 우 진

1983년 경북대학교 전자공학과 학사. 1989년 경북대학교 전자공학과 석사. 1995년 ~ 현재 경북대학교 컴퓨터공학과 박사과정. 1991년 ~ 현재 영진전문대학 컴퓨터정보기술계열 부교수 재직중. 관심분야는 데이터베이스 시스템, 분산 시스템

김 기 종

1994년 경북대학교 전자공학과 학사. 1999년 경북대학교 컴퓨터공학과 석사. 1994년 ~ 현재 영진전문대학 컴퓨터정보기술계열 전임강사 재직중. 관심분야는 멀티미디어 시스템, 데이터베이스 시스템, 분산 시스템, 암호학

유 기 영

정보과학회논문지 : 컴퓨팅의 실제
제 6 권 제 5 호 참조