논문 2007-02-01

휴대용 멀티미디어 기기에서 메타데이터 활용을 강화한 파일 시스템 구조

(A File System Architecture for Enriched Metadata in Portable Multimedia Devices)

윤 현 주[1]
(Hyeon-Ju Yoon)

Abstract : In this paper, we developed a file system architecture for portable multimedia devices. To enhance user convenience, the information about the stored files should be easily retrieved and organized. We defined NMD (Networked MetaData), which can organize the files in networked fashion by attaching user-defined attributes and relation between files. The NMD is stored in flash memory to utilize its nonvolatile property and low-power consumption, while multimedia files are stored in hard disk, an inexpensive mass storage. The experimental implementation showed that this architecture was able to save about 10% power compared to the hard disk NMD-store.

Keywords : portable multimedia device, enriched metadata, dual file system

Ⅰ. 서 론

DMB 수신, 동영상 및 음악 재생, 내비게이션 등의 기능을 갖춘 복합형 멀티미디어 휴대 기기의 사용이 늘어나고 있다. 이들은 대용량의 동영상 데이터를 비롯한 멀티미디어 데이터의 재생을 주요 기능으로 하며, DMB 방송의 녹화(저장), 음성 녹음 등의 멀티미디어 데이터 생성 기능도 일부 지원한다. 또한 터치스크린이나 간단한 버튼에 의한 조작 기능과 소프트웨어 키보드에 의한 문자 입력 등의 제한된 사용자 인터페이스를 제공한다. 그리고 호스트 컴퓨터와의 연결을 통해 보다 다양한 파일 조작 기능을 할 수 있다.

그런데 휴대 기기의 저장 장치가 커지고 파일 수가 많아질수록 사용자가 필요로 하는 파일을 빠른 시간 내에 검색하고 그룹핑하는 것에 대한 요구가 증가하는데, 현재의 휴대 기기는 파일 시스템의 구조와 그를 조작하는 사용자 인터페이스가 제한되어 있어 파일에 대한 조작과 검색을 쉽게 할 수 없다는 단점이 있다. 예를 들어, 최근 들어 많이 개발되는 DMB 겸용 멀티미디어 기기의 경우 녹음/녹화 기능도 같이 제공하는데, 단순히 버튼을 조작하는 것만으로 쉽게 녹화를 할 수 있지만 녹화된 동영상 파일의 이름은 "mYTN_102123.mp4"와 같이 임의로 작성되어 사용자가 나중에 특정 프로그램이나 녹화분을 찾고자 할 때는 쉽지가 않다. 그리고 출연배우나 제목 등 키워드를 이용하여 파일을 디렉터리 구조로 정리하거나 그에 따라 검색을 할 때에도 터치스크린과 펜 정도의 제한된 입력 장치로 인해 이를 수행하기가 쉽지 않아, 대부분 호스트 컴퓨터에서의 응용프로그램 연결을 통해야 한다. 휴대용 멀티미디어 기기의 저장 장치로 사용되는 하드디스크는 2.5인치 크기에서도 200 GB 용량의 제품들이 상용화되어 있어 200개 이상의 영화, 약 6만여 개의 음악 데이터를 저장할 수 있다고 보면, 호스트 컴퓨터 없는 휴대 상황에서 원하는 파일이나 데이터를 접근하여 읽고 쓰기 위해서는 기존의 디렉터리 구조 이상의 파일 시스템 구조가 필요하다.

하드디스크의 용량이 커져 저장하는 데이터양이 많아지고 웹 검색과 같은 환경에 익숙한 사용자들의 요구에 따라, 데스크톱 환경에서 사용할 수 있는 검색 소프트웨어가 일반화되고 있고[1], 이제는 파일 시스템과 데이터베이스를 직접 연결하는 새로운 형태의 파일 시스템을 장착한 운영체제도 속속 발표되고 있다[2,3]. 그러나 휴대형 멀티미디

어와 같은 임베디드 기기에서는 일반 데스크톱에서 고려해야 하는 효율이나 속도의 문제 외에 전력 소모량이나 사용자 인터페이스 등의 제약점을 같이 고려해야 한다.

본 논문에서는 메타데이터나 콘텐츠, 파일 간의 연관성을 이용한 정보를 구축하는 확장 메타데이터 구조인 Networked Metadata (NMD)를 제안하고, 이러한 연관성을 여러 응용 프로그램에서 효율적으로 이용할 수 있도록 파일 시스템에서 지원하는 방법에 대하여 제안하였다. 또한 저전력, 고속으로 이러한 기능을 실행할 수 있도록 플래시 메모리를 활용하는 파일 시스템 구조를 제안하고 프로토타입을 구현하여 그 성능을 평가하였다.

본 논문의 2장에서는 강력한 검색 기능을 결합하는 기존의 파일 시스템에 대해서 알아보고 모바일 멀티미디어 기기에서 요구되는 특성을 기술한다. 3장에서는 본 논문에서 제안하는 NMD와 파일 시스템 구조에 대해 설명하고, 4장에서 프로토타입 구현 및 간단한 성능 평가 결과를 보인다. 5장에서는 결론과 향후 연구 방향을 제시한다.

II. 관련 연구

최근 컴퓨터 또는 휴대형 시스템에 탑재되는 저장장치의 용량이 커지고, 파일의 종류 및 크기 또한 다양해지면서 기존의 단순한 메타데이터 구조에 기반한 파일 저장 또는 검색을 넘어서는 기능에 대한 요구가 커지고 있다. 웹 검색이 일상화되면서 컴퓨터에 저장된 데이터 및 파일에 대해서도 웹 검색과 같은 다양한 방법의 검색 및 파일 그룹핑 등을 원하는 것이다.

구글 데스크톱을 비롯한 데스크톱 검색 소프트웨어들은 개인용 또는 업무용 컴퓨터를 위한 검색 프로그램으로 컴퓨터에 저장된 이메일, 문서 파일, 사진, 웹 기록, 비디오, 음악 파일 등에 대해 탐색을 위한 색인을 만들고 빠르고 다양한 검색을 지원한다. 무료 제품이거나 AOL, Yahoo, Google 등과 같은 포털 또는 검색 사이트의 서비스 소프트웨어로 제공되는 경우가 많으나, 기업용 상용 제품도 많이 출시되어 있다[1]. 이들은 데스크톱에서의 검색 작업을 보다 신속하고 풍부하게 해 준다는 장점이 있으나, 독립적인 소프트웨어이므로 다른 응용들에서 그 기능을 유기적으로 사용할 수 없다는 한계가 있다. 또한 일부 제품은 포털 서버와의 연계 등으로 인한 보안 문제가 대두되기도 하였다.

한편, 마이크로소프트, 애플, 썬 마이크로시스템즈 등, 상용 컴퓨터와 운영체제를 개발하는 회사의 최근 또는 차기 운영체제들에서 메타데이터를 데이터베이스화하고 콘텐츠에 대한 강력한 검색 기능을 포함하는 파일 시스템 탑재가 일반화되고 있다. WinFS[3]는 롱혼에 탑재할 목적으로 개발되던 파일시스템으로서 콘텐츠를 포함한 다양한 요소로 메타데이터를 구성하고 XML 및 SQL 기술에 기반한 검색과 통합 관리를 지원한다. 얼마 후 출시될 Windows Vista [4]에서는 WinFS를 포함하지는 않았지만 데스크톱 검색 기능을 강화하여, 폴더 구조에 관계없이 빠른 검색이 가능할 뿐만 아니라 파일 속성이나 관련 데이터를 추가하거나 편집하는 기능이 있어 보다 효과적인 검색을 할 수 있다.

애플 컴퓨터에서 2005년에 내놓은 새로운 OS Tiger (MacOS/X 10.4)는 시스템 상의 파일들을 빠르고 효율적으로 검색할 수 있는 기능을 내장한 최초의 상용 운영체제이며, 메타데이터와 콘텐츠 색인을 만들 수 있는 도구인 Spotlight[2]를 함께 제공하고 있다. Spotlight는 OS의 파일시스템인 HFS+에 유기적으로 통합된 검색기술로서, 파일이 생성되거나, 저장, 이동, 복사, 제거될 때마다 파일 시스템은 파일에 대한 인덱스와 카탈로그 작업을 백그라운드에서 수행하여 검색에 사용할 수 있게 한다. 또한 검색기술을 개발자들이 사용할 수 있도록 API를 제공하여, 개발자의 응용프로그램에서 파일 검색, 플러그인 적재 등을 위하여 검색 기술을 제한 없이 사용할 수 있다.

LiFS(Linking File System)[5]에서는 기존 시스템의 메타데이터 색인 및 검색 기능 외에 파일 간의 관계(relation)를 설정하여 링크 메타데이터를 만들 수 있도록 하였다. 그리고 이러한 색인 및 링크 정보를 빠르게 검색하고 활용하기 위하여, 휘발성인 RAM의 한계를 극복하고 속도가 느린 하드디스크의 약점을 보완할 수 있는 차세대 비휘발성 메모리[6]를 포함하는 시스템에 활용할 것을 제안하였다.

이들은 모두 대용량의 다양한 데이터를 다루어야 하는 차세대 파일 시스템에 대한 방향성을 제시하고 있으나, 데스크톱 시스템을 기반으로 하고 있어 임베디드 또는 휴대형 기기에 대한 적용 및 요구 분석은 별로 이루어지지 않은 상황이다.

본 논문에서 제안하는 확장 메타데이터를 지원하는 파일 시스템은 RAM과 플래시 메모리, 하드디스크를 모두 사용하는 멀티미디어 휴대기기에서 운용되는 것을 목표로 하며, LiFS에서 제안한 아

이디어 중, 메타데이터 DB를 플래시 메모리 등의 비휘발성 메모리에 저장하여 기존의 메인 메모리 DB[7]보다 전력 소모를 줄이면서 다양한 관계 설정과 질의 검색을 가능하게 하는 방법이다.

III. 파일시스템 구조

서론에서 설명한 바와 같이 휴대형 멀티미디어 기기에서 다양한 형태의 파일 검색 및 조작 기능을 지원하기 위해서 다음과 같은 기능들이 요구된다.

- 각 파일의 메타데이터와 콘텐츠로부터 정보를 추출하고 조직화하는 기능
- 파일 간의 관계를 다양하게 표현하고 저장하는 기능
- 사용자가 메타데이터 정보를 추가할 수 있는 기능
- 저전력, 고속으로 메타데이터 검색 및 멀티미디어 데이터 입출력 실행
- 한정된 형태의 입출력 방식에 적합한 인터페이스

이 중 각 파일의 메타데이터나 콘텐츠로부터 정보를 추출하고 조직화하는 기능은 본 논문의 범위를 벗어나므로 제외하도록 한다. 또한 한정된 형태의 입출력 방식에 적합한 인터페이스는 본 연구에서 제안하는 파일 시스템의 응용 프로그램에 해당하는 것으로, 본 논문에서는 그를 지원할 수 있는 API를 정의하고 프로토타입 구현으로 가능성을 보여 주도록 한다.

1. Networked MetaData 구조

NMD (Networked MetaData)는 기존의 메타데이터와 더불어 다양한 형태의 파일 정보를 기록하고 조직할 수 있게 하여 사용자에게 편의성을 제공하려는 것이다. NMD에 포함되는 정보 내용은 다음과 같다.

- 기존 파일 시스템의 메타데이터 (예: Unix/Linux의 inode, NTFS의 attribute)
- 파일 헤더 및 콘텐츠로부터 자동 추출된 메타데이터 (예: 사진 파일의 해상도, 문서 파일의 단어, 비디오 파일의 영상 클립)
- 사용자가 직접 입력하는 메타데이터 (예: 친구와 같이 찍은 사진에 친구 이름 입력)
- 파일시스템으로부터 생성되는 파일 간 관계 (relation) (예: 상위 폴더와 하위 폴더 간 관계)
- 사용자의 입력 또는 콘텐츠로부터 생성되는 파일 간 관계 (예: 연속극을 녹화한 파일들 간에 순서 관계 정의)

NMD 구조는 기본적으로 네트워크형 데이터 모델을 따른다. NMD는 다음과 같이 세 가지 기본 요소에 의해 정의되며, 이들은 방향 그래프 형태로 표현될 수 있다.

- 파일 (file): 파일은 여러 가지 속성(attribute)으로 구성되는 메타데이터와 내용(contents)을 가진다.

속성 (attribute): 속성은 key-value 쌍으로 표현되며, 키는 파일 구조 및 시스템에 의해 정의되는 predefined key와 사용자에 의해 정의되는 arbitrary key로 구분된다. 일반적인 메타데이터에 나타나는 속성 외에 관계를 구성하는 속성이 추가된다.

- 관계 (relation): 파일들 간의 관계 또는 속성 간의 관계를 정의한다.

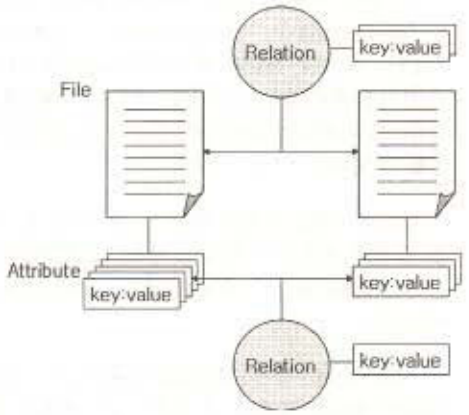

그림 1. NMD 기본 요소
Fig. 1. NMD Elements

관계는 암묵적 (implicit) 관계와 명시적 (explicit) 관계로 나누어진다. 암묵적 관계는 메타데이터 또는 파일시스템 구조에 의해 생성되는 관계로 디렉터리 포함 관계, 같은 형식을 가진 파일, 같은 속성 값을 가진 파일의 집합 등이 이에 속한다. 구현 시 암묵적 관계는 파일 생성이나 변경 시 자동으로 설정되어 별도로 기록된 다음 관계 정보만을 나중에 이용하는 방식과, 키워드에 의한 검색 시 파일시스템을 실제 검색하여 도출하는 검색 지향 방식의 두 가지 형태가 가능하다. 본 연구의 목적은 휴대형 기기에서 사용하는 것이므로 검색 때마다 직접 파일시스템(하드디스크)을 접근하는 것은 비효율적이고 따라서 파일의 생성 또는 변경

때마다 자동 생성하는 방식을 가정한다. 명시적 관계는 사용자 또는 응용프로그램에 의해 지정되어 이루어지는 관계이다.

파일들 간의 관계는 비대칭과 대칭의 두 가지 형태가 있다. 디렉터리 포함 관계나 순서가 있는 동영상 시리즈물 등의 관계는 비대칭이며, 같은 작곡가의 음악 파일 간의 관계는 대칭적이다. 파일들 간의 관계는 다음과 같이 정의할 수 있다.

FileRelation = (source, target, {set of attributes}, symmetry)

파일 관계의 예는 다음과 같다.

- (a.dvi, a.tex, {dependency:source}, FALSE): a.dvi의 source는 a.tex다. 암묵적 관계이다.
- (my_mbc_102304.mp4, my_mbc_113203.mp4, {drama:sequel, title:joomong}, FALSE): 드라마 주몽 동영상의 순서 관계 정의로 사용자가 지정할 수 있다. 파일의 이름은 녹화 시 임의로 생성된다.
- (goldberg.mp3, toccata.mp3, {artist:bach}, TRUE): 두 음악 파일의 작곡가는 Bach이다. 메타데이터 분석을 통해 얻어질 수 있는 암묵적 관계이며, 메타데이터에 기록되지 않은 경우 사용자가 명시적으로 지정할 수 있다.

속성들 간의 관계는 비대칭, 즉 어떤 속성이 다른 속성의 부속적인 성질을 가질 때 정의한다.

AttrRelation = (superAttr, subAttr:{set of values})

예를 들어, 문서 파일에는 hwp, doc, pdf 등 다양한 형태의 포맷이 존재한다. 이 관계를 표현하면 다음과 같다.

AttrRelation = (type:document, format:{hwp, doc, pdf})

이러한 속성 관계를 정의함으로써, 키워드에 의한 파일이나 관계 검색 시에 상위 속성을 명시하면 하위 속성들까지 자동으로 검색하게 하여 보다 다양하고 편리하게 이용할 수 있다.

2. NMD API 설계

NMD를 응용프로그램 등에서 활용하기 위해서는 파일시스템에 통합되거나 연동해서 동작하는 구현이 필요하며, 그 인터페이스가 정의되어야 한다. 앞에서 정의한 NMD에 대한 API를 다음과 같이 정의한다.

(1) 관계 생성 및 삭제

Function Name	Operation
mk_file_rel (source, target, attrlist, symmetry)	source와 target 파일 간에 특정 속성에 의해 정의되는 관계 설정
mk_attr_rel (super, sublist)	상위 속성과 하위 속성들 간의 관계 설정
del_file_rel (source, target, attrList)	source와 target 파일 간에 특성 속성에 의해 정의되는 관계 삭제
del_attr_rel (super, sublist)	상위 속성과 하위 속성들 간의 관계 삭제

(2) 관계의 속성 조작

Function Name	Operation
get_rel_attr_set (source, target)	source와 target 파일 간에 정의된 속성을 모두 추출

Function Name	Operation
set_rel_attr (source, target, key, value)	source와 target 파일 간에 key와 value로 정의되는 관계 설정 또는 변경
del_rel_attr (source, target, key, value)	source와 target 파일 간에 key와 value로 정의되는 관계 삭제
get_rel_set (source)	특정 파일이 source가 되는 관계 모두 검색
get_match_rel_set (source, attrList)	특정 파일이 source가 되는 관계 중, 속성이 attrList인 것을 모두 검색

(3) 속성 조작

Function Name	Operation
create_attr (key, [valueRange])	key와 특정 값을 갖는 속성 생성
get_attr_rel(key, value)	key와 value로 정의되는 속성의 속성 관계에 따른 하위 속성 검색

(4) 데이터 저장 모델 및 질의문 API

실제 관계 및 속성 등을 검색하는 질의문(query)의 API는 구체적인 데이터 저장 모델과 구현, 그리고 응용에 따라 매우 달라질 수 있다. 추상 모델은 방향 그래프로 표현되지만, 저장 모델은 방향 그래프의 직접 구현, 관계형 DB와 같은 테이블 구조, XML 스키마 표현 등을 활용할 수 있으며, 그래프 탐색을 이용하는 질의문을 API와 정규식 형태로 직접 구현하거나 SQL, XSL, XQL 등을 활용하여 그에 적합한 질의문을 정의할 수 있다.

3. NMD를 지원하는 파일 시스템 구조

본 논문에서 제안하는 휴대기기용 파일시스템의 전체 구조는 <그림 2>와 같다. 저렴한 비용으로 대용량의 멀티미디어 데이터의 효율적인 입출력을 가능하게 하기 위하여 멀티미디어 데이터를 대용량 하드디스크에 저장하고, 본 논문에서 제안하는 NMD 등 랜덤 접근 데이터를 위한 비휘발성 저장 장치로 플래시 메모리를 사용한다.

NMD의 크기는 파일의 개수와 종류, 그리고 관계 및 속성 등 사용자 정의 메타데이터의 크기에 의해 결정된다. 최소 수십 KB에서 수백 MB 이상으로 확장될 가능성이 있으며, 주로 사용자가 멀티미디어 데이터를 재생하기 전에 원하는 파일을 검색하기 위한 용도로 사용한다는 특징이 있다. 따라서 빠른 검색 속도를 제공해야 하고, 기존 메타데이터라 할지라도 런타임 시에 개별 파일을 접근해서 정보를 추출하기보다는 미리 NMD를 구성해서 저장해 둘 것이 요구된다. 즉, NMD 저장소는 대용량, 비휘발성이어야 하고, 고속 접근이 가능해야 한다. 또한, 배터리를 사용하는 휴대형 기기에서는 전력 사용량도 심각하게 고려해야 한다.

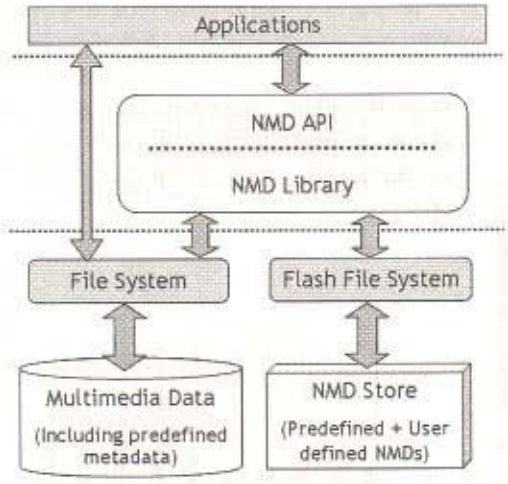

그림 2. 확장된 메타데이터를 지원하는 파일시스템 전체 구조

Fig. 2. File system architecture supporting Networked Metadata

휴대형 멀티미디어 기기는 대용량의 멀티미디어 데이터를 저장하기 위해 하드디스크를 사용하고, OS 및 사용자 프로그램을 플래시 메모리에 저장하여 사용하는 경우가 많다. 하드디스크는 탐색 시간이 오래 걸리고 기계적 작동이 필요한 만큼 전력 소모량도 많지만, 가격 대비 저장용량이 매우 크고 멀티미디어 데이터와 같은 순차 탐색이 자주 요구되는 경우에는 속도에 문제가 없고 전력소모량도 최소화할 수 있기 때문에 유용하다. 그러나 NMD 저장소는 그 특성상 랜덤 접근에 의한 검색이 더 많이 요구되므로 하드디스크에서는 속도가 저하될 우려가 있고, 전력 소모량도 커진다.

DRAM 등 메인 메모리의 용량이 커지고 가격도 하락하는 상황에서는 MMDB와 같이 NMD 저장소를 런타임 시에 모두 메인 메모리에 두고 실행하는 것도 고려해볼 수 있다. 그러나 [8]의 연구에서 밝힌 바와 같이, 메인 메모리는 플래시 메모리 사용 시보다 최소 10배, 최대 1000배 이상 전력 소모량이 많기 때문에 휴대형 기기에서는 그다지 적합하지 않다고 판단된다.

본 논문에서는 비휘발성이면서 대용량 저장이 가능하고, 랜덤 검색 속도와 전력 소모량 면에서도 우수한 플래시 메모리를 NMD 저장소로 두고, 멀티미디어 파일들을 하드디스크에 저장하는 이종 파일 시스템 구조를 제안한다. NMD를 구현하는 API와 라이브러리는 NMD 저장소에 대한 기본 연산을 지원하는 것으로 기존 파일 시스템 및 플래시 파일 시스템과 연동하여 작동하며, 사용자를 위한 응용프로그램 개발에 활용될 수 있다.

IV. 프로토타입 구현과 성능 분석

1. 프로토타입 구현

본 연구에서는 네트워크 DB 개념에 가까운 NMD의 프로토타입을 구현하기 위하여 HyperMMSP2[9] 보드 상에 현재 활용하기가 용이한 관계형 DB 엔진인 SQLite[10]을 사용하여 라이브러리와 간단한 응용프로그램을 구성하였다. HyperMMSP2 보드의 주요 사양은 <표 1>과 같고, <그림 3>은 라이브러리 구현 구조를 보여 준다.

표 1. HyperMMSP2 보드 사양

Table 1. Specification of HyperMMSP2 board

구분	항목	내용
H/W	CPU	MMSP2 MP2520F
	SDRAM	128MB
	Nand Flash	64 MB
	Display	4" Color TFT LCD
	Multimedia	Audio/Video Codecs, Mic/Speaker jack

	Interfaces	USB, IDE, CF, Serial, JTAG
S/W	OS	Embedded Linux Kernel 2.4.19
	File system	YAFFS
	GUI	TinyX

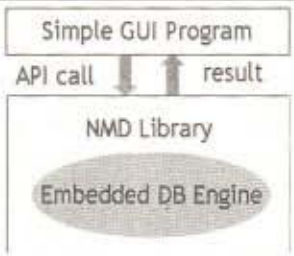

그림 3. NMD 라이브러리 프로토타입 구현
Fig. 3. Implementation of NMD Library Prototype

SQLite 엔진은 이를 활용하는 다른 프로그램에 임베딩되어 사용되며, 엔진 사이즈가 작아 휴대형 기기와 같은 임베디드 시스템에 활용하기 적합하다. 본 연구에서 제안한 API를 SQLite 사용에 적합하게 변형하고 SQLite 엔진을 포함하는 라이브러리로 구성하여 응용프로그램에서 사용할 수 있도록 구현하였으며, SQLite에서 사용하는 데이터베이스 파일은 YAFFS[11]를 통해 플래시 메모리에 입출력되도록 구성하였다.

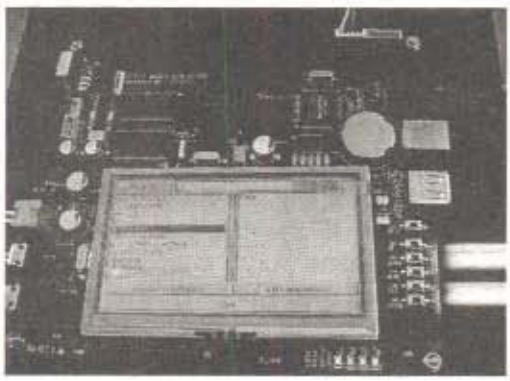

그림 4. MMSP2 보드에 구현한 모양
Fig. 4. Implementation snapshot

구현된 라이브러리는 앞 절에서 설계한 내용 중 파일 간의 관계를 설정하는 것과 속성 간의 관계를 설정하는 부분을 매우 낮은 수준(low-level)에서 다루었다. 각 관계 및 파일은 동등한 형태의 entry 구조에 저장되며, 이들 간의 관계는 부모-자식 형태로 표현된다. API는 관계의 생성, 삭제, 변경, 검색 등의 기능을 정의하고 있다. 응용의 일종인 GUI 프로그램은 자주 사용되는 속성을 미리 설정해 두고 사용자가 그 관계를 정의하거나 파일에 할당할 수 있도록 한다.

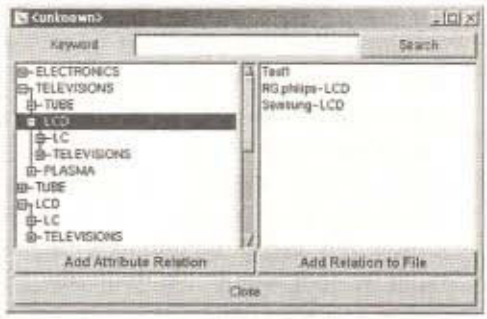

그림 5. 속성 관계를 설정하는 GUI 프로그램
Fig. 5. A GUI program to help setting relations and attributes

2. 성능 평가

본 논문에서 제안한 NMD 지원 이중 파일 시스템의 성능을 전력 소모량과 검색 속도 면에서 하드디스크에 저장하는 경우와 비교하여 측정하였다. 테스트에 사용된 프로그램은 DB로부터 100개의 관계를 검색하여 결과를 가져오도록 하였고, 검색의 대상이 되는 DB의 크기를 변화시키며 실험하였다.

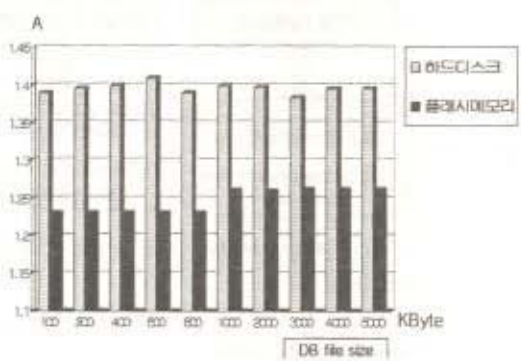

그림 6. 하드디스크 DB와 플래시 메모리 DB 실행 시의 전류량
Fig. 6. Power consumption comparison

<그림 6>은 DB 크기에 따른 전력 소모량의 차이를 보여 준다. 전력 소모량 측정의 경우, 두 경

우 모두 보드 전체의 전류량을 측정하였으므로 CPU나 LCD 등에서 사용하는 일정량의 전력은 같은 수준을 유지한다. 전압은 거의 일정하게 4.98~4.99V를 유지하므로 전류량으로 표시하였다. 예상한 바와 같이 플래시 메모리에 NMD를 저장할 경우 하드디스크보다 적은 전력을 소모하며, (평균전류량 × 전압)으로 계산하면 약 10% 정도의 전력이 절감됨을 볼 수 있다.

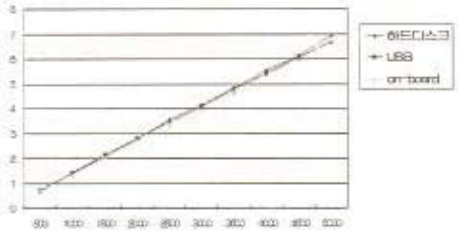

그림 7. 하드디스크 DB와 플래시 메모리 DB 실행 시의 검색 속도

Fig. 7. Data retrieval time

<그림 7>은 NMD DB를 각각 하드디스크, USB 타입 플래시 메모리, 보드 내장 플래시 메모리에 저장했을 때의 검색 속도를 비교한 것이다. 예상과는 달리 속도 차이가 거의 없음을 볼 수 있으며, 5KB 정도의 DB에서도 검색에 수초 이상 걸려 실용적이지 못한 결과를 보인다. DB의 크기에 따라 속도가 선형으로 증가하는 것은 SQLite가 하나의 DB 파일을 관리하며 자주 저장 장치를 접근하는 것에 기인하는 것인데, 이는 검색을 위한 절차를 바꾸어 메인 메모리에 캐쉬되는 분량을 늘림으로서 개선할 수 있다. 최종 구현을 SQLite로 결정한다면 다양한 최적화 기법을 적용할 수 있을 것이다. 또한 하드디스크와의 속도 차이가 없는 것은 일차적으로 연속된 영역에 대한 저장과 검색을 반복하는 실험 데이터의 문제로 인해, 하드디스크에서 탐색 시간과 회전 시간에 의한 지연이 거의 없었기 때문으로 분석된다. 플래시 메모리 전용 파일 시스템인 YAFFS는 기존 파일 시스템과 FTL(Flash Translation Layer)[12]로 구현된 USB 타입 메모리 입출력과 비교해 그다지 성능이 우월하지 않음을 알 수 있다.

V. 결 론

본 연구에서는 휴대용 멀티미디어 기기에서 편리한 검색 기능을 구현할 수 있도록 하는 파일 시스템 구조를 제안하였다. 개별 파일의 메타데이터 및 사용자 정의 메타데이터를 데이터베이스를 이용해 저장하고, 이를 하드디스크 대신 휴대용 기기에서 주로 사용하는 플래시 메모리에 저장함으로써 비휘발성 특성을 살리면서 휴대용 기기에 필수적인 저전력 실행이 가능하도록 하였다.

한편, 플래시 메모리 자체의 읽기/쓰기 속도에 비해 매우 떨어지는 검색 성능을 높이기 위하여 DB 파일에 대한 플래시 메모리 저장 구조에 대한 연구가 추가로 계속되어야 한다. 플래시 메모리는 RAM이나 하드디스크 등의 기존 기억 장치와 다른 장치 특성을 가지고 있어 전용 파일 시스템들이 연구되고 있으며, B-Tree와 같이 특정 데이터 저장 구조에 특화된 플래시 저장 구조에 대한 연구[13]도 있다. 본 논문에서는 데이터 저장 모델을 구체적으로 지정하지 않았으며, 이의 설계와 함께 그 검색 성능을 최적화할 수 있는 플래시 저장 구조 연구가 필요하다.

또한 본 논문에서 제안한 이중 파일 시스템은 휴대형 기기와 NMD에 최적화된 파일 시스템으로 통합될 필요가 있다. 그리고 프로토타입 구현에서는 생략했지만, 추후 실용화를 위해서는 파일 포맷에 따른 메타데이터 및 파일 콘텐츠로부터 자동으로 정보를 추출하거나 인덱싱하는 기법들을 결합하여 구현하는 것이 필수적이다.

참고문헌

[1] "Desktop Search", http://en.wikipedia.org/wiki/Desktop_search, 2006.

[2] Apple Developer Connection, "Working with Spotlight", http://developer.apple.com/macosx/tiger/spotlight.html, 2004.

[3] R. Grimes, "Revolutionary File Storage System Lets Users Search and Manage Files Based on Content," http://msdn.microsoft.com, 2004.

[4] http://www.microsoft.com/korea/windowsvista/default.aspx, 2006.

[5] A. Ames, N. Bobb, S. C. Brandt, A. Hiatt,

"Richer File System Metadata Using Links and Attributes," Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies, Monterey, CA, Apr. 2005.

[6] 유병곤, 류상욱, 윤성민, "유비쿼터스용 유니버설 메모리 기술," 전자통신동향분석, 제20권 제1호, 2005년 2월.

[7] M. Wang, "Main Memory Databases," http://www.cs.cmu.edu/~natassa/courses/15-823/S01/lecturees/18_mmdb.pdf, 2001.

[8] C. Park, J.-U. Kang, S.-Y. Park, J.-S. Kim, "Energy-Aware Demand Paging on NAND Flash-based Embedded Storages," International Symposium on Low Power Electronics and Design, 2004.

[9] http://www.hybus.net, 2006.

[10] http://www.sqlite.org/, 2006.

[11] YAFFS, http://www.aleph1.co.uk/yaffs/index.html, 2006.

[12] "Understanding the Flash Translation Layer (FTL) Specification," Intel, Dec. 1998.

[13] C.-H. Wu, L.-P. Chang, T.-W. Kuo, "An Efficient B-Tree Layer for Flash-memory Storage Systems," Proceedings of the 9th International Conference on Real-Time and Embedded Computing Systems and Applications(RTCSA), 2003.

저 자 소 개

윤 현 주

1988년 서울대학교 컴퓨터공학과 학사. 1990년 한국과학기술원 전산학과 석사. 1997년 한국과학기술원 전산학과 박사. 2005년~현재: 금오공과대학교 컴퓨터공학부 재직. 관심분야: 분산시스템, 임베디드 소프트웨어, 센서 네트워크.

Email: juyoon@kumoh.ac.kr