논문 2011-06-25

안드로이드 플랫폼 상에서 동기화가 고려된 통합 커서의 설계 및 구현

(Design and Implementation of an Unified Cursor Considering Synchronization on the Android Mobile Platform)

김 경 환*, 하 주 호, 원 종 필, 이 의 성, 김 주 민, 손 진 호
(Kyung-Hwan Kim, Jo-Ho Ha, Jong-Pil Won, Uee-Song Lee, Joo-Min Kim, Jin-ho Son)

Abstract : Android platform provides a content provider and a cursor mechanism to access the internal SQLite engine. Content providers not only store and retrieve data but also make it accessible to applications. Applications can only share data through content provider, since there's no common storage area that Android packages can access. Cursor is an interface that provides random read-write access to the result set returned by a database query. However, this cursor possesses two major limitations. First, a cursor does not support a join clause among cursors, since the cursor can only access a single table in the content provider. Second, the cursor is not capable of creating user-customized field in the predefined content providers. In this paper, we propose the unified cursor architecture that merges several cursors into a single virtual cursor. Cursor translation look-aside buffer (TLB), column windowing mechanism and virtual data management are the three major techniques we have adopted to implement our structure. And we also propose a delayed synchronization method between an application and a proposed unified cursor. An application can create a user-customized field and sort multiple tables using a unified cursor on Android platform.

Keywords : Android, Database, Embedded software, Mobile, Multimedia message

I. 서 론

최근 스마트폰의 성능이 향상되고, 사용 가능한 메모리의 용량이 증대됨에 따라 음악, 동영상, 사진과 같은 다양한 멀티미디어 파일이 스마트폰에 저장되는 추세이다. 또한 Twitter, Facebook과 같은 SNS(Social Networking Service)의 대중화로 기존과는 비교할 수 없을 정도로 많은 메시지와 연락처가 스마트폰에 저장되고 있다. 따라서 이러한 데이터들을 고속으로 검색하고 관리하기 위해, 스마트폰에서도 SQLite와 같은 내장형 데이터베이스가 기본적으로 탑재되고 있다. 모바일 시장에서 급성장세를 보이고 있는 Android 플랫폼의 경우에도 데이터베이스 엔진으로써 SQLite를 채택하고 있으며, 응용 프로그램과의 인터페이스를 위해 Content Resolver, Content Provider, Cursor 와 같은 메커니즘을 제공한다 [1].

Android의 Content Provider는 Linux 시스템 상에서 서로 다른 응용 프로그램 간에 데이터 공유를 위한 메커니즘으로써, 데이터베이스를 제공하는 응용 프로그램의 수정 없이는 데이터베이스의 스키마를 변경할 수 없는 문제를 가지고 있다. 또한 서로 다른 프로세서의 Content Provider를 통해 제공되는 데이터들 간에 결합(join), 정렬을 수행할 수 없는 문제점을 가지고 있다.

본 논문에서는 이러한 문제점들을 해결하기 위해 여러 개의 물리적인 커서들을 가상의 단일 커서로 변환 시켜주는 통합커서 구조를 제안하며, 응용

* 교신저자(Corresponding Author)
논문접수 : 2011. 02. 28., 수정일 : 2011. 03. 19.,
채택확정 : 2011. 03. 28.
김경환, 하주호, 원종필, 이의성, 김주민, 손진호 :
LG전자 미래IT융합연구소

프로그램과의 데이터 동기화 효율을 높이기 위한 지연식 통보 방식도 제안한다. 제안하는 통합커서는 커서 변환 색인 버퍼(Translation Look-aside Buffer: TLB), 커서 간 컬럼 사상 기법, 가상 컬럼 데이터 관리 기법으로 구성된다. 커서 변환 색인 버퍼는 다수의 물리적 커서를 단일의 가상 커서로 변환한기 위한 테이블이다. 커서 간 컬럼 사상 기법은 가상 커서의 컬럼들을 개별적인 물리적 커서의 컬럼들로 변환시키는 메커니즘이다. 가상 컬럼 데이터 관리 기법은 데이터베이스의 스키마 변경 없이 임의의 컬럼을 추가하고, 해당 컬럼 접근 시에 가상의 데이터를 생성하기 위한 방법이다. 마지막으로 지연식 통보 방법은 데이터베이스 내의 데이터 변동 시에 일정 시간의 지연을 가지고 응용 프로그램에 통보를 해주는 메커니즘이다.

본 논문의 구성은 다음과 같다. 2장에서는 안드로이드 시스템의 데이터베이스 구조 및 동기화 과정에 대해 살펴보고, 3장에서는 통합 커서의 구조 및 지연식 동기화 방법에 대해 제안한다. 4장에서는 제안한 통합커서를 이용해 통합 메시지 응용 프로그램을 구현한 것을 보이며, 마지막으로 5장에서는 결론을 맺는다.

II. 안드로이드 플랫폼의 데이터베이스 구조 및 동기화 방법

1. 데이터베이스 계층 구조

안드로이드 시스템은 대량의 데이터를 효율적으로 다루기 위해 SQLite 데이터베이스 엔진을 제공한다. SQLite는 2000년에 Richard Hipp 박사에 의해 개발된 무료 데이터베이스 엔진으로써, 안정적이고 용량이 작아 소규모의 임베디드 시스템 용 데이터베이스에 적합하다 [2]. SQLite는 아이폰, 심비안 등의 모발일 환경에 많이 채용되어 있으며 휴대용 MP3 플레이어에도 많이 이용되고 있다 [3]. 안드로이드 시스템은 SQLite 엔진을 이용해 취득된 데이터를 응용 프로그램에 전달하기 위해 그림 1과 같은 계층 구조를 가지고 있다.

구성도를 살펴보면 크게 응용 프로그램 계층, 프레임워크(framework) 계층, 물리적 데이터베이스 계층의 세 부분으로 나누어지는 것을 알 수 있다. 응용 프로그램 계층은 안드로이드 SDK(Software Development Kit)를 이용해서 개발되는 프로그램들로써 메시지 송수신 프로그램, 멀티미디어 재생 프로그램, 사진 편집 프로그램 등이 그 예가 될 수 있다. 프레임워크 계층의 주요 구성요소는 Content Resolver와 Content Provider이다. Content Resolver는 AndroidManifest.xml 파일에 명시된 이용 가능한 Content Provider 들의 접근 권한(permission)과 URI(Uniform Resource Identifier)를 이용해 특정 Content Provider를 반환하는 역할을 한다. Content Provider는 서로 다른 응용 프로그램 간에 데이터 접근을 위한 안드로이드 표준 인터페이스로써 Content Resolver를 통해서만 접근이 가능하다. 데이터베이스 계층은 플래시 메모리 파일 시스템인 YAFFS2(Yet Another Flash File Sytem)와 SQLite 엔진으로 구성되며, 실제 데이터가 저장되는 물리적인 저장장치에 해당한다.

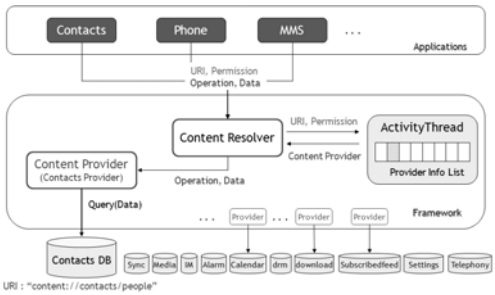

그림 1. 안드로이드의 데이터베이스 접근 구조
Fig. 1. Database access method of android

2. 안드로이드 시스템의 커서 인터페이스 구조

안드로이드 시스템에서 데이터베이스로의 접근은 그림 1과 같이 Content Resolver를 통해서만 이루어진다. Content Resolver는 파라미터로 전달되는 URI를 이용해 특정 Content Provider를 찾아내며, Content Provider 내의 함수들을 접근하는 통로 역할을 하게 된다. 따라서 Content Resolver 객체의 query, insert, delete, update 함수를 호출하게 되면 Content Provider 객체의 query, insert, delete, update 함수가 호출된다. 안드로이드 시스템은 Content Resolver를 통해서만 Content Provider에 접근 가능하게 함으로써, 서로 다른 응용프로그램 간에 권한 문제없이 데이터 공유를 가능하게 하였다.

응용 프로그램이 Content Resolver의 query 함수를 호출하게 되면 Content Provider의 query 함수가 호출된다. Content Provider의 query 함수는 반환 값이 Cursor 객체로써, 안드로이드 시스템은

일반적인 커서의 기능(포인터 이동) 외에 부가 기능들을 추가해 Cursor 클래스를 구현해 놓았다. 응용 프로그램은 Cursor 객체를 이용해 데이터 셋 내의 포인터 이동뿐만 아니라 데이터베이스의 스키마 정보와 전체 데이터 셋의 크기를 알 수 있다. 또한 2.3절에서 설명하는 데이터베이스와 응용 프로그램 간의 데이터 동기화를 위한 Observer의 등록도 Cursor 객체의 멤버 함수를 이용한다.

3. 데이터베이스와 응용 프로그램 간의 동기화

안드로이드 시스템은 응용 프로그램과 데이터베이스간의 데이터 동기화를 위해서 Observer 메커니즘을 제공한다. 일반적인 Observer 메커니즘은 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체들한테 연락이 가고, 자동으로 내용이 갱신되는 방식으로 일대다(one-to-many) 의존성을 정의한다. 일대다 관계는 주제(Subject)와 Observer에 의해 정의가 되며, Observer는 주제에 의존하게 된다. 주제의 상태가 바뀌면 Observer에게 연락이 오며, 연락 방법에 따라 Observer에 있는 값이 새로운 값으로 갱신된다 [4].

그림 2는 Android 시스템에서 Observer를 이용한 데이터 동기화 메커니즘을 보여주며, 전체 과정은 네 단계로 구분할 수 있다. 첫 번째 단계는 감시하려는 데이터베이스의 ContentProvider URI를 등록하는 단계이다. Observer의 등록은 register ContentObserver 함수와 registerDataSet Observer 함수를 이용하며, ContentProvider의 URI가 파라미터로 이용된다. 안드로이드 시스템에서는 Observer를 ContentObserver와 DataSet Observer로 구분한다. ContentObserver는 개별 데이터의 삽입, 삭제, 갱신과 같은 상태 변화를 감지하며, DataSetObserver는 데이터베이스 질의 시에 반환되는 Cursor 객체의 close, requery, deactivate와 같은 상태 변화를 감시한다. 두 번째 단계는 다른 프로세스로부터 데이터가 변경되는 단계이다. 다른 프로세스로 부터 데이터베이스에 insert, update, delete와 같은 데이터 조작이 수행된다. 세 번째 단계는 데이터베이스의 변경 후에 notifyChange 함수를 호출하는 단계이다. notifyChange 함수는 안드로이드 프레임워크 단에 구현되어 있는 함수로써, 특정 ContentProvider의 URI를 감시하는 모든 Observer들에 변경사항을 통보하는 역할을 한다. 마지막 단계는 Observer내의 onChange 함수가 호출되는 단계이다. onChange 함수는 Observer 생성 시에 재정의가 필요한 callback 함수로써, Observer가 데이터베이스 변경 사항을 통보 받으면 자동으로 호출된다.

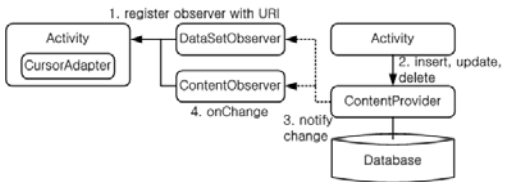

그림 2. 데이터 동기화 과정
Fig. 2. Process of data synchronization

Ⅲ. 동기화가 고려된 통합 커서

1. 통합 커서의 전체 구조

본 논문에서 제안한 통합 커서는 서로 다른 스키마로 구성된 데이터베이스들을 단일의 가상 스키마를 가지는 데이터베이스인 것처럼 보이게 만들어 준다. 따라서 통합 커서를 이용하는 응용 프로그램은 다양한 데이터베이스들의 개별 스키마에 대한 고려 없이, 단일 인터페이스를 통해 서로 다른 데이터베이스 상의 컬럼들을 접근 할 수 있다. 또한 통합커서를 이용해 가상의 컬럼들을 생성할 수 있고, 기존 데이터베이스의 스키마 변경 없이 기존 스키마에 결합할 수 있다. 그림 3은 통합 커서를 사용하는 응용 프로그램 입장에서 보게 되는 가상 데이터베이스의 그림이다. 물리적 데이터베이스들의 커서 인터페이스들이 통합 커서를 통해 가상 데이터베이스의 단일 커서처럼 보이게 되는 것을 알 수 있다.

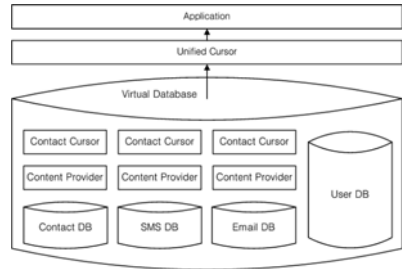

그림 3. 통합 커서를 이용한 가상 데이터베이스 접근
Fig. 3. Access a virtual database using the unified cursor

통합커서 내에서의 처리 과정은 커서 변환 과정, 컬럼 변환 과정, 데이터 fetch의 3 단계로 구분되

며, 데이터베이스에 CRUD(Create, Read, Update, Delete) 연산이 발생할 경우 지연식 동기화 방법을 통해 데이터의 변동사항을 통보 받게 된다. 통합 커서의 전체 구조는 그림 4와 같으며, 주요 컴포넌트들은 Cursor TLB, Column Windowing Manager, Virtual Field Manager, Sync Handler Manager, Cursor Interface Layer이다. Cursor TLB 블록은 커서 변환 색인 버퍼이며, 가상 커서와 물리적 커서 간의 변환 작업을 담당한다. Cursor TLB 내의 단일 TLB 아이템들 간에는 정렬이 가능하며, 이를 통해 서로 다른 데이터베이스에 존재하는 데이터들 간에 정렬 기능을 수행할 수 있다. Column Windowing Manger는 커서 간 컬럼 사상 기법이 구현되어 있는 컴포넌트로써 컬럼 간의 인덱스 멥핑 테이블을 관리한다. 이 테이블을 통해 가상 컬럼의 특정 인덱스 접근이 물리적 컬럼의 실제 인덱스 값으로 변환된다. Virtual Field Manager는 가상 컬럼의 데이터를 관리하는 역할을 하는 컴포넌트로써, 데이터베이스에 사상된 컬럼의 데이터와 사용자가 정의한 컬럼의 데이터가 구분되어 처리된다. 사상된 컬럼들의 데이터 fetch는 기존의 데이터베이스에 있는 데이터들을 이용하고, 사용자 정의 컬럼들의 데이터 fetch는 가상 컬럼 데이터 관리 기법에 의해 제공된다. Sync Handler Manager는 지연식 동기화 기법을 위한 컴포넌트로써 데이터베이스 변동 시에 응용 프로그램에 변경사항을 통보하는 역할을 한다. Sync Handler Manager를 통해 다수의 데이터베이스로부터 생성되는 데이터 변경 통보들이 필터링(filtering) 되어 응용 프로그램에 전달된다. Cursor Interface Layer는 안드로이드 플랫폼의 표준 Cursor 인터페이스 변환 부분으로써 통합 커서를 안드로이드 Cursor 인터페이스를 통해 접근할 수 있게 해 준다.

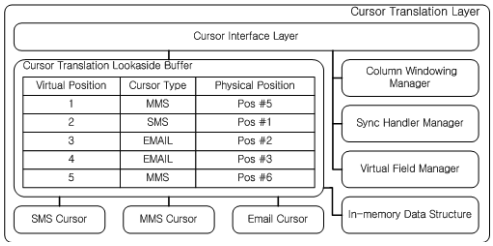

그림 4. 통합 커서의 전체 구조도
Fig. 4. Architecture of a unified cursor

2. 커서 변환 색인 버퍼

2.1 커서 변환 색인 버퍼의 구조

안드로이드 시스템은 Content Resolver의 질의에 대한 반환 값으로써 Cursor라는 객체를 반환한다. Cursor는 질의의 결과 추출된 데이터들을 접근하기 위한 포인터로써 안드로이드에서는 순차접근과 임의접근을 제공한다 [5]. 또한 Cursor는 내부 데이터베이스 엔진과 응용 프로그램을 분리시켜 독립적인 인터페이스를 제공해주는 역할도 하게 된다. 본 논문에서는 여러 개의 물리적인 커서를 가상의 커서로 변환하기 위해 커서 변환 색인 버퍼(Cursor TLB) 메커니즘을 고안하였다. 일반적으로 TLB는 가상 메모리를 물리적인 메모리로 변환하는 하드웨어 장치로써 컴퓨터의 MMU(Memory Management Unit)를 구성하는 한 부분이다 [6].

그림 5는 CursorTLB의 구조와 가상커서가 물리적 커서로 변환되는 과정을 보여준다. 그림 5의 Cursor TLB 부분을 보면, CursorTLB의 단일 아이템은 인덱스 값, 물리적 커서의 종류, 데이터 셋 내에서의 레코드 위치로 구성되는 것을 알 수 있다. 인덱스 정보는 응용 프로그램이 가상 커서에 접근하는 실제 위치에 대한 정보로써, 인덱스를 키 값으로 물리적 커서의 종류와 레코드의 위치가 구해진다. 커서 종류는 여러 개의 물리적 커서를 구분하는 용도로 사용되며, 레코드 내의 위치와 조합하여 물리적 커서 내의 실제 데이터를 구하는데 이용된다. 전체 변환 과정은 다음과 같다. 응용 프로그램에서 가상 커서의 0번째 레코드를 요청하게 되면, Cursor TLB의 변환 과정을 통해 인덱스 0번에 있는 물리적 커서가 SMS인 것과, 0번째 위치에 있는 레코드를 요청한다는 것을 알게 된다. 따라서 SMS 데이터베이스의 0번째 레코드가 반환된다. 가상 커서의 5번째 데이터를 접근하는 경우에도 동일한 메커니즘을 통해, 커서의 종류가 MMS 이고 1번째 위치에 있는 레코드를 요청한다는 것을 알 수 있다.

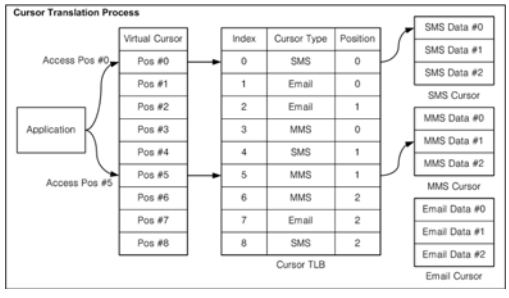

그림 5. 커서 변환 색인 버퍼
Fig. 5. Cursor translation look-aside buffer

2.2 커서 변환 색인 버퍼를 이용한 정렬

안드로이드 시스템은 ContentResolver에 질의 시에 특정 컬럼을 기준으로 정렬을 할 수 있는 기능을 제공한다. 통합 커서에서도 물리적 커서 내의 데이터 간에 정렬 기능을 제공하기 위해 CursorTLB 내의 개별 아이템들 간에 정렬 기능을 구현 하였다. CursorTLB 내의 개별 아이템들이 정렬되어 있으면, 서로 다른 데이터베이스에 존재하는 데이터들도 정렬된 상태로 반환이 가능하다. 통합 커서를 이용한 데이터 접근은 커서 변환 색인 버퍼를 통해 물리적 커서로 변환되어 데이터베이스에 접근하므로, 서로 다른 데이터베이스 내의 데이터들이 정렬된 것과 같은 효과를 줄 수 있다.

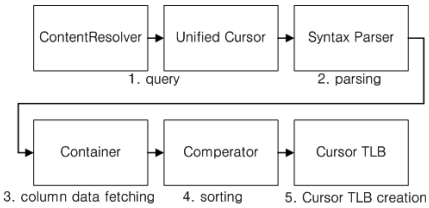

그림 6. Cursor TLB를 이용한 정렬 과정 흐름도
Fig. 6. Cursor TLB sorting process

그림 6은 Cursor TLB를 이용해 정렬이 수행되는 전체 흐름도를 보여주며, 전체 과정은 5단계로 구분된다. 첫 번째 단계로 응용 프로그램이 통합커서에 정렬을 원하는 컬럼의 이름과 정렬 순서를 파라미터로 질의를 하게 된다. 질의는 ContentResolover의 멤버 함수인 query 함수를 이용하게 된다. 두 번째 단계로 통합커서는 전달받은 파라미터들을 분석하여 가상의 컬럼인지 물리적 컬럼인지를 구분하고, 정렬 순서를 파악한다. 세 번째 단계로 해당 컬럼의 모든 데이터들을 Container에 저장한다. 가상 컬럼인 경우에는 3.4절의 가상 데이터 관리 기법에 따라 데이터의 fetch가 이루어지고, 물리적 컬럼에 사상된 경우에는 각각의 데이터베이스에서 직접 데이터의 fetch가 이루어진다. 네 번째 단계로 Container에 저장된 데이터들 간에 정렬이 수행된다. 정렬의 구현은 Java 언어의 Comparable 인터페이스와 Collection 객체 내의 sort함수를 이용하였다. 따라서 Cursor TLB 클래스는 Comparable 클래스를 implements 하고, Cursor TLB 클래스 내에 compareTo 함수를 구현하였다. 마지막으로 파라미터를 파싱(parsing)해 구한 정렬 순서에 따라 Cursor TLB가 구성되게 된다. 데이터 정렬이 완료된 후에는 통합커서를 통해 데이터에 접근하게 되면, 정렬된 순서에 따라 3.4절의 과정을 거쳐 데이터의 fetch가 이루어지게 된다.

그림 5의 Cursor TLB 부분을 보면, 가상 커서의 데이터 인덱스에 해당하는 커서의 종류가 SMS, MMS, Email 순으로 순차적으로 나오는 것이 아니라 랜덤하게 나오는 것을 알 수 있다. 이는 그림 5의 Cursor TLB가 특정 Column에 따라 정렬이 이루어진 후의 결과 값을 보여주고 있기 때문이다.

3. 커서 간 컬럼 사상 기법

안드로이드 플랫폼에서 응용 프로그램은 Cursor 객체에서 제공해 주는 인터페이스를 이용하여 데이터베이스 내의 특정 레코드에 있는 특정 컬럼의 데이터로 접근이 가능하다. 통합 커서도 기존의 Cursor 객체와 동일한 인터페이스를 제공하기 위해 가상 커서의 컬럼들을 물리적 커서의 컬럼들로 변환시켜주는 커서 간 컬럼 사상 기법 (Cursor Column Windowing)을 고안하였다. 커서 간 컬럼 사상 기법을 이용해 사용자는 통합 커서의 객체 생성 시에 기존 스키마에는 없는 가상 컬럼들을 추가로 생성할 수 있고, 가상 컬럼들을 다수의 물리적 컬럼들로 사상시킬 수 있다. 이후에는 통합 커서의 가상 컬럼들을 이용하여 데이터베이스에 접근하더라도, 내부적으로는 다수의 물리적 커서의 컬럼들로 변환되어 데이터베이스에 접근하게 된다.

안드로이드 시스템에서 커서를 통해 데이터베이스에 접근하기 위해서는 질의 결과로 얻어지는 데이터 셋 내에서의 컬럼들 이름과 컬럼들의 인덱스 값을 알아야 한다. 이에 대한 정보를 제공하기 위해 안드로이드 시스템은 Cursor 객체 내에 다양한 멤버 함수들을 정의하고 있다.

표 1. 레코드 접근을 위한 의사 코드
Table 1. Pseudo code of record access

```
Cursor c;
String columnNames[] = null;
int columnIdx;
String columnData = null;

c = context.getContentResolver().query(SMS_URI,
projection, selection, null, sorting);
columnNames = c.getColumnNames();
columnIdx = c.getColumnIndex(columnNames[0]);
columnData = c.getString(columnIdx);
```

표 1은 Cursor 인터페이스를 이용해 특정 데이터에 접근하는 의사 코드를 보여준다. getColumnNames 함수를 이용해 질의 결과 얻어진 데이터 셋에 존재하는 모든 컬럼들의 이름을 알 수 있고, 이들 중 특정 컬럼의 이름에 해당하는 인덱스를 알기 위해서는 getColumnIndex 함수를 이용한다. 특정 컬럼의 인덱스가 구해지면, Cursor내의 getString, getInt, getLong과 같은 멤버 함수들을 이용하여 해당 컬럼의 데이터를 원하는 데이터 형태로 구할 수 있게 된다.

그림 7은 가상 커서의 컬럼들과 물리적 커서의 컬럼들 간의 사상정보를 보여준다. 응용 프로그램에서 통합 커서의 객체 생성시에 가상 커서의 컬럼들로 msg_id, msg_type, msg_subject, msg_body, msg_attachment의 다섯 개 컬럼들을 생성했고, SMS 메시지 데이터베이스와 Email 메시지 데이터베이스로부터 이들에 해당하는 컬럼들을 사상시켰다. 그림 7에서 “Virtual Column Name“은 가상 컬럼의 이름을 내타내고, "SMS Column Name"과 "Email Column Name"은 각각 SMS 메시지의 데이터베이스와 Email 메시지의 데이터베이스 내에 존재하는 컬럼의 이름들이다. fullIdx와 dataIdx는 가상 컬럼과 물리적 컬럼을 사상시키기 위해 사상 정보를 보관하는 컨테이너이다. fullIdx 컨테이너는 사용자가 생성한 가상 컬럼들을 관리하기 위한 array 형태의 자료구조로써, 가상 컬럼과 1대 1로 사상된다. fullIdx 컨테이너에 저장된 인덱스 값들은 -1인 경우에 사용자 정의 컬럼을 나타내며, 이외의 값인 경우에는 dataIdx 컨테이너에서 참조해야 하는 위치정보를 나타낸다. dataIdx 컨테이너는 물리적 커서가 가리키는 데이터 셋 내에서 컬럼들의 인덱스 정보를 보관하는 역할을 한다. 이들 인덱스 값을 이용해 실제 데이터베이스 내의 특정 컬럼에 접근하게 된다.

커서 간 컬럼 사상 테이블이 생성되는 전체 메커니즘은 2단계로 나누어진다. 첫 번째 단계로 컬럼 선별 작업이 이루어진다. 가상 컬럼은 사용자 정의 컬럼과 물리적 커서에 사상되는 컬럼으로 구분되며, 통합 커서의 객체 생성 시에 사용자의 파라미터 정보를 이용하여 컬럼들의 선별 작업이 수행된다. 그림 7의 SMS, Email 메시지 컬럼들 중 문자열이 존재하는 컬럼들은 사상되는 컬럼들이고, “”와 null로 주어진 컬럼들의 이름은 사용자 정의 컬럼들이다. 사용자 정의 컬럼들은 데이터베이스의 스키마 상에는 존재하지 않는 가상의 컬럼들로써, 이 컬럼들에 대해서는 3.4절에서 상세히 설명한다. 두 번째 단계로 dataIdx 테이블 생성 작업이 이루어진다. 물리적 커서에 사상되는 컬럼들의 정보를 이용하여 데이터베이스에 질의를 하게 되면 데이터 셋에 대한 Cursor 객체가 얻어진다. 이렇게 구한 Cursor 객체의 getColumnIndex 함수를 호출하여 각 컬럼들의 컬럼 인덱스들을 dataIdx 컨테이너에 채우게 된다. 그림 7에서 "_id"의 dataIdx 값 0은 SMS 데이터베이스에 질의 결과로 얻어진 데이터 셋 내에서 “_id" 컬럼의 실제 인덱스 값이 0이라는 것을 나타낸다. 세 번째 단계로 fullIdx 테이블 생성 작업이 이루어진다. fullIdx 테이블의 생성은 통합 커서 객체 생성시에 입력받은 가상 컬럼의 이름들과 물리적 커서의 이름들을 이용한다. 가상 커서의 컬럼 이름이 “” 이거나 null 인 경우에는 컨테이너의 값를 -1로 채우고, 그 외의 경우에는 두 번째 단계에서 구한 dataIdx 컨테이너의 위치 정보를 채우게 된다.

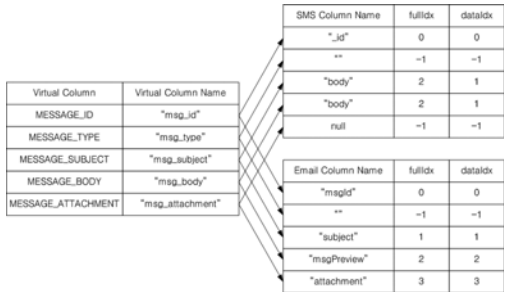

그림 7. 커서 간 컬럼 사상 테이블

Fig. 7. Translation table between virtual cursor and physical cursor

커서 간 컬럼 변환의 전체 과정은 다음과 같다. 그림 7에서 응용 프로그램이 MESSAGE_BODY에 접근하게 되면 가상 컬럼의 이름이 msg_body인 것을 알게 된다. SMS 메시지라고 가정하면 msg_body에 해당한는 물리적 컬럼의 이름은 body이고, fullIdx값은 2인 것을 알 수 있다. fullIdx값 2는 dataIdx 컨테이너에서 2번째 인덱스의 값을 취하라는 의미이다. dataIdx [2]의 값인 1이 물리적 컬럼 이름인 body의 데이터 셋 내에서의 실제 인덱스 값에 해당한다. 그림 7에서 가상 컬럼인 msg_subject와 msg_body가 물리적 컬럼의 body에 동일하게 사상 된 것은 body 컬럼의 데이터를 제목에도 표시하기 위해서이다.

4. 가상 컬럼 데이터 관리 기법

안드로이드 플랫폼은 데이터베이스에서 이용 가

능한 컬럼들의 정보를 제공하기 위해, Cursor 인터페이스 내에 getColumnNames 함수를 제공한다. getColumnNames 함수는 데이터베이스의 스키마에 정의되 모든 컬럼들의 이름을 Java String의 Array 형태로 반환하며, AbstractCursor를 implements 하는 Cursor 인터페이스에서는 반드시 구현되어야 하는 함수이다. 안드로이드 시스템에서 Cursor 인터페이스 방식은 단순히 Provider를 접근하기 위한 통로로써, 데이터베이스에서 제공되는 컬럼들 이외에는 별도로 컬럼들을 추가할 수 없는 문제점을 가지고 있다. 안드로이드 시스템에서 제공해주는 저수준의 SQLite 제어 함수를 이용하여 컬럼을 추가할 수 있는 방법이 있으나, ContentProvider가 구현된 응용 프로그램에서만 수정이 가능한 한계를 가지고 있다 [7]. 따라서 이 경우에도 URI를 통해 다른 응용 프로그램의 ContentProvider에 접근하는 응용 프로그램들은 임의로 컬럼 생성을 할 수가 없다.

이러한 문제점을 해결하기 위해 본 논문에서는 가상 컬럼 데이터 관리 기법을 고안하였다. 가상 컬럼 데이터 관리 기법은 가상 컬럼 생성부와 컬럼 데이터 가공부로 나누어진다. 가상 컬럼 생성부는 3.3절의 커서 간 컬럼 사상 기법을 이용하며, 가상 컬럼 생성 시에 물리적 컬럼의 이름을 “”와 null로 사상한다. “”로 생성된 물리적 컬럼은 임의 데이터 가공이 필요하다는 표시이며 컬럼 데이터 가공부의 함수 구현을 통해 데이터를 반환하게 된다. null로 생성된 물리적 컬럼은 별도의 데이터 가공이 필요 없이 0이나 null로 반환되는 컬럼을 의미한다. 실제로 서로 다른 데이터베이스를 통합하는 응용 프로그램에서 다른 데이터베이스에는 없는 컬럼들에 대해 0으로 처리하는 경우가 빈번하여 null 처리 함수를 따로 작성하였다.

그림 8은 가상 컬럼의 데이터가 처리되는 전체 흐름을 보여준다. 응용 프로그램에서 getString, getInt, getFloat 등과 같은 함수를 이용해 데이터를 요청하게 되면 통합 커서 내의 CursorTLB를 이용하여 물리적 커서를 알아낸다. 물리적 커서를 찾아내면, 컬럼 변환 테이블을 이용하야 해당 컬럼이 사상된 컬럼인지 사용자 정의 컬럼인지 분별하는 단계를 거치게 된다. 이후 사상된 컬럼이면 데이터베이스의 데이터를 이용하고, 사용자 정의 컬럼이면 데이터 가공 함수를 이용하여 데이터를 생성한다. 데이터 가공 함수는 다양한 형태로 구현이 가능하다. 램에 상주하고 있는 데이터를 바로 전달하거나 데이터를 임의로 조작하여 반환하는 것이 가능하며, 기존에 데이터베이스에 존재하던 데이터들뿐만 아니라 네트워크 상이나 파일 상에 존재하는 데이터를 이용하는 것도 가능하다.

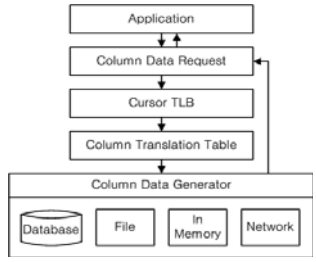

그림 8. 가상 컬럼 데이터 처리 과정
Fig. 8. Virtual column data process

5. 지연식 데이터 동기화 방법

통합커서는 여러 개의 물리적 커서를 통합하는 단일의 가상 커서이다. 따라서 통합하는 물리적인 커서에서 발생하는 통보의 수만큼 응용 프로그램에 통보의 상태를 알려주어야 한다. 통합 하려는 물리적 커서가 n개이고, 각각의 커서가 m개씩의 통보를 발생시키면 통합커서는 총 nm개의 통보를 발생시키게 된다. 이러한 전체 통보 방식은 지나친 query overhead로 인해 사용자 입력에 대한 처리 시간 지연을 유발하고, 응용 프로그램과 데이터베이스 간의 데이터 일관성을 유지하지 못하게 한다. 따라서 데이터베이스에서는 데이터가 갱신되었으나 응용 프로그램에는 반영이 늦어지는 문제가 발생한다.

본 논문에서는 이러한 문제를 개선하기 위해 지연식 통보 방법(delayed notification)을 고안하였다. 지연식 통보 방식에서는 초기 통보 후에 일정 시간의 통보 유예 시간을 두어, 통보 유예 시간에 들어오는 통보들을 타임아웃(time-out)이 발생한 시점에 하나의 통보로 처리한다. 따라서 유예 시간 동안 다수의 통보가 발생하여도 응용 프로그램에는 1개의 통보만 전달되게 된다. 그림 9는 전체 통보 방식과 지연식 통보 방식의 차이점을 보여주며, 본 논문의 구현에서는 통보 유예 시간을 100ms로 설정하였다. 통보 유예 시간 100ms는 본 논문에서 구현한 응용 프로그램에 최적화 되어 있다.

안드로이드 시스템에서 데이터베이스 변경시의 통보 과정은 2.2절에서 설명하였다. 본 논문에서 제안한 지연식 통보 방법은 통보 과정을 하위 계층과 상위 계층으로 구분하여 처리한다. 하위 계층은 Helper 객체가 담당하며, Helper 객체는 3가지의 기능을 수행한다.

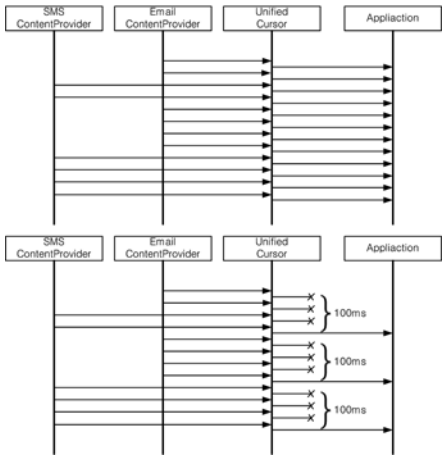

그림 9. 전체 통보 방식과 지연식 통보 방식 비교
Fig. 9. Delayed notification method

첫 번째 기능은 Observer들의 등록과 상위 계층(SyncHandler)으로의 메시지 이벤트 통보이다. 안드로이드는 Observer가 감시하는 Uri가 변경될 때마다 onChange 함수를 호출해 준다. 이벤트 통보는 onChange 함수 내에 구현되어 있으며, ID값이 0인 메시지를 SyncHandler로 발생시킨다. 두 번째 기능은 각각의 데이터베이스에 질의하기 위한 Query문들의 생성이다. Query문의 생성은 통합 커서의 객체 생성시에 전달받은 파라미터를 파싱해서 생성된다. 세 번째 기능은 3.4절에서 설명한 가상 컬럼 데이터 기법을 통한 데이터의 관리이다. Helper 객체 내에 실제 이용되는 가상 데이터들이 구현되어 있다. 상위 계층은 SyncHandler가 담당한다. SyncHandler는 하위 계층(Helper)으로부터 전달 받은 메시지들과 SyncHandler 자신이 발생시키는 메시지들을 처리하며, 통보 유예 시간 후에 응용 프로그램에 레코드 변경 사항을 통보하게 된다.

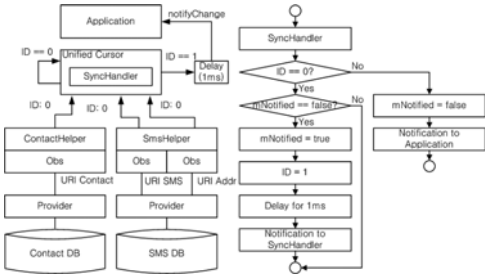

그림 10. 전체 통보 방식과 지연식 통보 방법 비교
Fig. 10. Delayed notification method

그림 10의 오른쪽 순서도는 SyncHandler의 전체 처리 과정을 보여주며, 전체 처리 과정은 다음과 같다. SyncHandler 객체는 첫 번째 계층인 Helper 객체와 SyncHandler 객체 자신으로부터 메시지를 받을 수 있다. Helper 계층으로부터 오는 메시지는 ID가 0인 값을 가지고 있고, mNotified가 false인 상태에서 통보를 받을 경우에는 mNotified를 true로 설정하게 된다. mNotified가 true인 상태에서 Helper 객체로부터 통보를 받게 되면 해당 통보들은 모두 무시가 되며, 유예 시간 동안의 지연 후에 ID값이 1인 메시지 이벤트를 SyncHandler 자신에게 발생시킨다. SyncHandler는 ID값이 1인 통보를 받게 되면 mNotified를 false로 설정하고, 응용 프로그램에 데이터베이스 변경사항을 통보하게 된다. 따라서 유예 시간을 주기로 1번의 통보만 발생되며, 유예 시간 동안 발생하는 다른 통보들은 모두 무시된다.

Ⅳ. 구현 및 실험

본 장에서는 LG전자의 옵티머스 3D를 이용한 통합 메시지함의 구현을 보이고, 지연식 동기화 방법을 이용하여 전체 통보 횟수를 줄인 실험 결과를 기술한다. 옵티머스 3D는 Dual-core 1GHz의 ARM Cortex-A9 CPU와 4Gb의 SDRAM, 4.3인치 LCD, 3D 듀얼 카메라를 장착한 안드로이드 2.2(프로요) 기반의 상용 모바일 폰이다.

1. 통합 메시지 리스트의 구현

그림 11은 본 논문에서 제안한 통합커서를 이용해 통합 메시지 리스트를 구현한 화면이다. 통합 커서는 서로 다른 데이터베이스들을 가상의 데이터베이스로 변환시키는 기능을 수행하므로, 서로 상관성이 없는 데이터들을 추출하기 위해 Music Player, Contact 응용 프로그램 그리고 메시지 응용 프로그램의 데이터들을 통합의 대상으로 선정하였다. 이들 응용 프로그램들은 안드로이드 플랫폼 배포 시에 기본적으로 탑재되어 있다.

그림 11에서 메시지 리스트 내의 한 개 아이템은 메시지의 종류, 메시지의 내용, 메시지가 저장된 시간으로 구성된다.

그림 11. 통합 메시지 리스트의 실행 화면
Fig. 11. Screen of the unified message list

이를 구현하기 위해 통합커서 단에 msgType, msgBody, timeStamp를 가상 컬럼으로 생성하였다. 그리고 이들 각각의 컬럼들은 실제 데이터베이스의 물리적 컬럼과 사상되어 있거나, 사용자 정의 컬럼으로 구성되어 있다. 가상 컬럼인 msgBody와 timeStamp는 물리적 컬럼과 사상된 컬럼들이다. msgBody 컬럼의 경우 Music Player에서는 title 컬럼으로, Contact 응용 프로그램에서는 display_name 컬럼으로, 메시지 응용 프로그램에서는 body 컬럼으로 사상되어있다. timeStamp 컬럼의 경우에는 Music Player에서는 date_added 컬럼으로, Contact 응용 프로그램에서는 sync3 컬럼으로, 메시지 응용 프로그램에서는 date 컬럼으로 사상되어있다. 가상 컬럼인 msgType 의 경우에는 실제 데이터베이스 상에는 존재하지 않는 컬럼으로써, 3.4절에 언급한 가상 컬럼 데이터 관리 기법을 이용해 레코드 내의 데이터를 생성하였다.

본 논문에서 제안한 통합커서를 이용할 경우, 서로 다른 테이블의 데이터 간에도 정렬이 가능하다는 것을 보이기 위해 각각의 데이터들을 msgType과 msgBody를 기준으로 오름차순 정렬을 수행하였다. 그림 11의 왼쪽 화면은 msgType 컬럼을 기준으로 정렬을 수행한 것이며, 각각의 메시지들이 메시지의 종류에 따라 정렬된 것을 알 수 있다. 그림 11의 오른쪽 화면은 msgBody 컬럼을 기준으로 정렬을 수행한 것이며, 각각의 메시지들이 본문 내용에 따라 오름차순으로 정렬된 것을 알 수 있다.

2. 지연식 동기화 방법 실험

지연식 동기화 방법의 성능 측정은 SMS 메시지와 MMS 메시지에 대해 지연식 동기화 방식을 사용하지 않았을 경우와 사용할 경우의 통보 개수 차이를 조사하였다. SMS, MMS 메시지들은 각 통신사들의 무선망을 통해 모바일 폰에 수신되므로, 많은 양의 메시지를 보내더라도 망 상태에 따라 임의적으로 메시지가 수신된다. 이때 수신된 SMS 메시지는 하나의 메시지 수신시마다 3번의 데이터베이스 갱신이 일어나고, MMS 메시지의 경우에는 9번의 데이터베이스 갱신이 일어난다.

그림 12를 보면 SMS 메시지의 경우 하나의 메시지가 수신되는 시간이 짧고, 100ms 미만의 짧은 시간동안 3개의 메시지가 집중적으로 발생하는 것을 알 수 있다. 이에 비해 MMS 메시지의 경우, 하나의 메시지가 수신되는 시간은 길지만 실제 데이터베이스의 갱신은 짧은 시간동안 이루어지는 것을 알 수 있다. 지연 시간의 설정에 따라 통보 회수가 달라지나, 100ms 를 가지는 지연식 동기화 방법을 이용할 경우 평균 67%의 SMS 통보수를 줄일 수 있었고, MMS의 경우에는 평균 50% 가량의 통보수가 줄어드는 것을 알 수 있었다.

SMS Message Arrived Time (3msg)	
Msg#1 – Noti#1	02-19 13:29:02.376
Msg#1 – Noti#2	02-19 13:29:02.392
Msg#1 – Noti#3	02-19 13:29:02.392
Msg#2 – Noti#1	02-19 13:29:09.056
Msg#2 – Noti#2	02-19 13:29:09.087
Msg#2 – Noti#3	02-19 13:29:09.173
Msg#3 – Noti#1	02-19 13:29:10.392
Msg#3 – Noti#2	02-19 13:29:10.400
Msg#3 – Noti#3	02-19 13:29:10.407

MMS Message Arrived Time (1msg)	
Msg#1 – Noti#1	02-19 13:42:25.165
Msg#1 – Noti#2	02-19 13:42:25.204
Msg#1 – Noti#3	02-19 13:42:51.243
Msg#1 – Noti#4	02-19 13:42:51.259
Msg#1 – Noti#5	02-19 13:42:51.275
Msg#1 – Noti#6	02-19 13:42:52.095
Msg#1 – Noti#7	02-19 13:42:53.735
Msg#1 – Noti#8	02-19 13:42:56.235
Msg#1 – Noti#9	02-19 13:42:56.310

그림 12. SMS, MMS 메시지 수신시 통보 회수
Fig. 12. Notification times of SMS and MMS

V. 결 론

본 논문에서는 여러 개의 물리적인 커서를 하나의 통합된 커서로 변환시키는 메커니즘을 제안하였고, 빠른 사용자 반응성을 보장하기 위한 지연식 동기화 기법도 고안하였다. 또한 제안한 메커니즘을 이용하여 통합 메시지 수신함을 안드로이드 플랫폼 상에 구현하였다. 통합 메시지 수신함은 통합 커서를 이용함으로써, 여러 개의 물리적 데이터베이스들을 단일의 가상 데이터베이스처럼 이용 할 수 있었다.

통합커서의 구현을 위해 가상 커서를 물리적 커서로 변환시키는 커서 변환 색인 버퍼를 고안하였으며, 가상 컬럼을 물리적인 컬럼으로 사상시키는 커서 간 컬럼 사상 기법도 개발하였다. 또한 사용자 정의 컬럼을 생성하고, 이를 관리하기 위한 가상 컬

럼 데이터 관리 기법도 고안하였다. 이를 통해 기존의 안드로이드 커서 메커니즘에서는 제공해 주지 않는 다양한 기능들을 수행할 수 있었다. 다중 Provider 간의 데이터 정렬이 가능했고, Join의 기능 수행 및 사용자 정의 컬럼의 추가가 가능했다. 또한 빈번한 데이터베이스 변경시에도 일정한 수준의 통보수를 유지함으로써, 빠른 사용자 반응성을 보장할 수 있었다.

향후 제안된 메커니즘을 다양한 응용 프로그램에 적용하기 위해서는 SQLite 엔진에서 제공해 주는 다양한 질의 문들을 범용적으로 해석할 수 있게 통합 커서 내의 파서 부분이 수정되어야 할 것이다. 또한 좀 더 빠른 사용자 반응성을 보장하기 위해 컬럼 데이터의 일부를 Caching 하는 방법이 필요할지도 모른다.

참고문헌

[1] http://d.android.com/guide/

[2] http://en.wikipedia.org/wiki/Sqlite

[3] 김상형, 안드로이드 프로그래밍 정복, 한빛미디어(주), 2010.

[4] Erich Gamma, Richard Helm, Ralph Johnson, "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, Professional, first edition, 1994.

[5] A. Eisenberg, J. Melton, "SQL Standardization: The Next Steps", ACM SIGMOD Record, 29(1), pp. 63-67, 2000.

[6] R. Rashid, A. Tevanian, M. Young, D. Golub, R. Baron, D. Black, W. Bolosky, J. Chew. "Machine-Independent virtual memory management for paged uniprocessor and Multiprocessor Architectures", IEEE tansactions on Computers, Vol.37(8), pp. 896-908, Aug. 1988.

[7] Mark L., "Beginning android", first edition, Apress, 2009.

[8] http://download.oracle.com/javase/7/docs/api/

[9] 진희규, 이기용, 우경구, "멀티미디어 CE 기기를 위한 빠른 질의 복구 기법", 정보과학회논문지, Vol.35, No.3, pp. 286-205, 2008.

[10] Silberschatz A., Korth H., Sudarshan, S. "Database System Concepts", 4th edition, McGrawHill, 2001.

[11] http://en.wikipedia.org/wiki/CRUD

저 자 소 개

김 경 환 (Kyung-hwan Kim)

2001년 : 서울대학교 전기공학부 학사.
2008년 : 서울대학교 전기컴퓨터공학부 석사.
현재, LG전자 미래IT융합 연구소 선임연구원.

관심분야 : Linux Kernel, Robotics, Real-time OS.
Email : kyunghwan.kim@lge.com

하 주 호 (Ju-ho Ha)

2000년 : 서울대학교 전기공학부 학사.
2007년 : 서울대학교 전기컴퓨터공학부 석사.
현재, LG전자 미래IT융합 연구소 선임연구원.

관심분야 : 임베디드 소프트웨어, 안드로이드 시스템.
Email : juho.ha@lge.com

원 종 필 (Jong-pil Won)

2003년 : 한국항공대학교 정보통신공학과 학사.
2005년 : 한국항공대학교 정보통신공학과 석사.
현재, LG전자 미래IT융합 연구소 선임연구원.

관심분야 : 안드로이드 OS, 안드로이드 응용개발, 무선네트워크 프로토콜, 미래인터넷, 컴퓨터 비젼.
Email : jongpil.won@lge.com

이 의 성 (Uee-song Lee)

1998년 : 서강대학교 컴퓨터공학과 학사.
2000년 : 서강대학교 컴퓨터공학과 석사.
현재, 미래IT융합연구소 책임연구원.

관심분야 : 임베디드 소프트웨어, 소프트웨어공학, 뇌 과학.
Email : ueesong.lee@lge.com

김 주 민 (Joo-min Kim)

1998년 : 숭실대학교 전자공학과 학사.
2001년 : 숭실대학교 전자공학과 석사.
현재, 미래IT융합연구소 책임연구원.

관심분야 : Real-time OS, Artificial Intelligence, Natural Interface.
Email : joomin.kim@lge.com

손 진 호 (Jin-ho Son)

1991년 : 성균관대학교 전기전자컴퓨터공학과 학사.
1993년 : 성균관대학교 전기전자컴퓨터공학과 석사.
2004년 : 성균관대학교 전기전자컴퓨터공학과 박사.
현재, 미래IT융합연구소 상무.

관심분야 : 임베디드 소프트웨어, 컴퓨터 비젼, Robotics, Bio-IT.
Email : jinho.sohn@lge.com