http://dx.doi.org/10.3745/KIPSTD.2011.18D.5.403

안드로이드 플랫폼에서 애플리케이션 간 객체 전송을 개선하기 위한 메타서비스의 설계 및 구현

최 화 영[†] · 박 상 원[††]

요 약

최근 안드로이드 기반의 스마트 폰이 널리 사용되고 있으며, 이를 이용한 다양한 응용이 개발중이다. 그 중 한 애플리케이션에서 생성한 데이터를 다른 애플리케이션으로 전달하는 경우가 빈번하게 발생한다. 그러므로 한 애플리케이션에서 발생하는 메타데이터를 다른 애플리케이션으로 쉽게 전달하거나 공유할 수 있는 방법이 필요하다. 안드로이드에서 애플리케이션이 생성한 메타데이터는 Java 객체로 만들어진다. 안드로이드에서 기존의 데이터 전달 방법으로는 클립보드와 인텐트 그리고 컨텐트 프로바이더가 있다. 하지만 이 방법들은 모두 레코드 형태로 데이터를 전달하도록 되어 있다. 그러므로 개발자는 객체를 전달하기 위해 마샬링과 언마샬링하는 과정을 직접 구현해야 한다. 본 논문에서는 애플리케이션에서 만든 임의의 타입의 객체를 전송할 수 있는 메타서비스를 설계하고 구현하였다. 메타서비스를 이용할 경우 클립보드와 컨텐트 프로바이더에서 객체를 전송하기 위해 구현해야하는 복잡한 과정을 줄여서 버그를 줄이고 코드를 간단히 작성할 수 있도록 하여 생산성을 증진시킬 수 있도록 하였다.

키워드 : 안드로이드, 클립보드, 객체 전송, 애플리케이션, 모바일 플랫폼

Design and Implementation of a MetaService for Improving Object Transfer among Applications on Android Platform

Hwayeong Choe[†] · Sangwon Park[††]

ABSTRACT

Recently, smart phones based on Android platform have been widely used, and lots of applications have been developed. Data generated from an application are frequently transferred to other applications. Therefore, a method that can easily transfer or share meta-data among applications is required. Generally meta-data created by android applications are java objects. The android platform uses clipboard, intent and content provider in order to transfer data among the applications. However, those ways are designed to transfer data as a record. So these methods have to marshall the object to a record and unmarshall the record to an object. In this paper, we designed and implemented the MetaService which can transfer any type of object made by applications. When the MetaService is used, we can reduce complex implementations such as clipboards and content providers and we can reduce many bugs. Therefore, we can make the applications simple and increase productivity.

Keywords : Android, Clipboard, Object Transfer, Application, Mobile Platform

1. 서 론

최근 화두가 되고 있는 여러 모바일 플랫폼 중 안드로이드 플랫폼은 오픈 소스 플랫폼으로 개발 접근성이 높아 많은 연구 및 개발이 진행되고 있다. 안드로이드 플랫폼을 기반으로 한 개발이 활발해지면서 안드로이드 애플리케이션, 라이브러리, 플랫폼 최적화 등 다양한 기술 및 연구 결과가 나오고 있는 실정이다. 그 중 스마트 폰에서 추출할 수 있는 다양한 애플리케이션에서 발생하는 정보를 담고 있는 메타데이터를 활용한 연구가 진행되고 있다. 메타데이터는 스마트 폰의 애플리케이션에서 발생하는 여러 데이터에 대한 정보를 담고 있는 데이터이다. 스마트 폰에서 발생하는 메타데이터는 사용자의 스마트 폰 활용 방법에 따라 개인화되고 특화된 데이터로 활용가치가 높다. 메타데이터를 사용하

※ 본 논문은 2011년 한국외국어대학교 교내학술연구비의 지원과 지식경제부 산업기술원천개발사업(10035348, 모바일 플랫폼 기반 계획 및 학습 인지 모델 프레임워크 기술 개발)의 지원으로 수행되었음.
† 준 회 원 : 한국외국어대학교 정보통신공학과 석사과정
†† 종신회원 : 한국외국어대학교 정보통신공학과 부교수
논문접수 : 2011년 6월 24일
수 정 일 : 1차 2011년 7월 26일, 2차 2011년 8월 12일
심사완료 : 2011년 8월 13일

여 새로운 부가가치를 창출할 수 있기 때문에 사용자 중심의 애플리케이션 개발과 메타데이터를 활용한 서비스 제공을 위한 연구가 진행되고 있다[1, 2, 3]. 메타데이터를 활용하기 위해 애플리케이션에서 발생하는 메타데이터를 보다 효과적으로 전송하고 처리할 수 있는 메커니즘이 필요하다. 메타데이터는 레코드 형태뿐만 아니라 다양한 형태로 존재할 수 있다. 따라서 다양한 형태의 데이터를 객체로 전달할 경우 편리하게 데이터를 전송할 수 있다.

다양한 기기간의 데이터를 전달하기 위한 메커니즘으로 Java의 RMI(Remote Method Invocation)[4][5]와 CORBA (Common Object Request Broker Architecture) [6] 그리고 COM(Component Object Model)[7]이 있다. 이러한 방법들은 동일 기기나 다른 기기 내의 다른 프로세스간의 데이터 전송이나 호출을 할 수 있는 방법이다. 스마튼 폰과 같은 모바일 기기에서 기기간 데이터를 전송하거나 함수를 호출하는 것은 보안 등의 문제로 사용하는 것이 거의 불가능하다. 또한 앞에서 언급한 방법들의 플랫폼은 안드로이드에서 적용하기가 곤란하다.

여러 모바일 플랫폼 중 안드로이드 플랫폼에서는 애플리케이션 간 객체를 전송하기 위해 클립보드[8], 인텐트(intent)[9] 그리고 컨텐트 프로바이더(content provider)[10]와 같은 메커니즘을 제공한다. 안드로이드 플랫폼에서 애플리케이션 간의 객체 전송은 매우 제한적이다. 다른 애플리케이션의 객체를 사용하기 위해서는 정확한 경로와 적절한 권한을 설정하여 접근이 가능하다. 이것은 다소 제한적인 방법이다. 또한 애플리케이션 간의 데이터나 객체 전송을 위한 구현이 복잡하다.

안드로이드에서 제공하는 클립보드는 getText(), setText() 등의 간단한 API를 사용하여 스트링 타입의 데이터를 전달할 수 있다[8]. 그리고 최근까지 레코드 형식의 데이터 전송을 지원하지 않았으나 API 레벨 11부터 레코드 형식의 데이터를 전송하는 것을 지원한다[11]. 하지만 레코드 형식의 데이터를 전송하기 위해 전송한 데이터를 컨텐트 프로바이더를 사용하여 원하는 데이터를 커서를 이용해 얻어야 하는 부분을 개발자가 구현해야 한다. 그리고 레코드 형식의 데이터 전송도 스트링 타입의 데이터와 마찬가지로 매우 간단한 메서드를 통해 처리할 수 있는 방법이 필요하다.

인텐트[9]는 안드로이드 컴포넌트 간의 통신 수단으로 어떤 액션이 수행되어야 할 메시지를 전달하는 메커니즘이다. 인텐트는 애플리케이션 간의 데이터를 전송하는 방법의 하나로 사용할 수 있다. 인텐트로 객체를 전환하기 위해서는 레코드 형식의 데이터를 객체로 변환하는 과정이 필요하다. 이러한 작업은 레코드 형식의 데이터를 객체로 변환하고 다시 레코드 형식의 데이터로 바꾸는 과정을 모두 구현해야 한기 때문에 개발의 효율성을 떨어뜨린다. 또한 인텐트는 주목적이 데이터 전송을 위한 메커니즘이 아니기 때문에 애플리케이션 간의 여러 가지 타입의 데이터를 객체로 전송하는 용도로 사용하기 부적합하다. 그리고 컨텐트 프로바이더 또한 객체를 전송하기 위해 객체의 마샬링(marshalling)과 언마샬링(unmarshalling) 과정을 거쳐 데이터를 변환하고 전달한다.

앞에서 언급한 애플리케이션 간의 데이터를 전송하기 위한 안드로이드의 메커니즘을 개선하기 위하여 본 논문에서는 편리한 데이터와 객체 전송을 위해 메타서비스를 설계하고 구현하였다. 컨텐트 프로바이더는 애플리케이션 간의 데이터 공유를 위한 메커니즘이기 때문에 공유 데이터에 대한 보안 문제에 대해 고려해야 한다. 하지만 본 논문에서 제안하는 메타서비스의 경우 데이터를 공유하는 것이 본래의 목적이 아니다. 클립보드의 목적은 복사, 붙여 넣기하는 데이터를 단순히 전달받아 사용하기 위한 것이고 컨텐트 프로바이더의 목적은 특정 애플리케이션이 생성한 데이터를 공유하기 위한 것이다.

본 논문에서 제안하는 메타서비스는 안드로이드에서 제공하는 클립보드의 간단한 API 스타일을 참고하여 스트링 타입의 데이터뿐만 아니라 객체도 전달 가능하게 하는 메커니즘이다. 메타서비스는 애플리케이션 간의 객체를 전송하기 위한 구현 과정의 복잡성을 줄여주고, 애플리케이션 간의 스트링 타입 데이터뿐만 아니라 객체의 전송을 지원한다. 2장 관련연구에서는 본 논문에서 구현되는 객체 전송을 위해 사용할 수 있는 메커니즘에 대해 설명하고 3장에서는 메타서비스와 기존 메커니즘의 차이점과 특징에 대해 기술한다. 그리고 4장에서는 메타서비스를 사용하기 위한 서비스 등록 방법과 메타서비스에서 지원하는 객체 타입에 대해 설명한다. 5장에서는 메타서비스의 구조 및 기능에 대해 자세히 다룬다. 그리고 6장에서는 메타서비스를 활용한 예제 프로그램을 보인다. 마지막으로 7장에서 향후 연구 방향에 대해 설명하고 결론을 맺는다.

2. 관련 연구

여러 모바일 플랫폼은 서로 다른 데이터 형식을 사용하기 때문에 기기 간 또는 기기 내의 애플리케이션 간의 객체를 전달하고 공유하는 것이 어렵다. 플랫폼과 기기의 종류가 많고 지원하는 객체의 형식도 다르기 때문에 객체 전달 및 공유 문제를 해결하기 위해 객체 전송 방법에 대한 연구가 진행되고 있다. 여러 대의 모바일 기기(PDA)들 간에 공개하는 데이터와 비공개 데이터를 서로 교환하기 위해 프로토콜을 개발[12]하거나 여러 기기 간의 데이터를 전달하는 방법으로 클라우드를 이용[13]하는 방법을 사용한다. 모바일 기기 간의 메타데이터를 공통된 형태로 저장 및 제공하기 위한 모바일 플랫폼[14]도 연구되고 있다.

2.1 분산 프로그래밍 기법

서로 다른 영역에 분산되어 있는 객체를 전송하고 처리하기 위한 메커니즘으로 Java의 RMI와 CORBA 그리고 COM이 있다. RMI[4, 5]은 기존의 자바 소켓으로 통신 프로그래밍을 하여 분산되어 있는 객체간의 메시지 교환하던 방식을

개선하여 분산된 객체를 마치 로컬 객체처럼 사용 가능하게 한다. 즉, 네트워크 상으로 다른 자바 애플리케이션과 통신할 수 있는 자바 애플리케이션을 만들기 위해 사용한다. 그리고 CORBA[6]는 OMG(Object Management Group)에서 정의한 OMA (Object Management Architecture)의 핵심이 되는 부분으로 객체지향기술을 기반으로 분산된 애플리케이션을 통합할 수 있는 표준 기술이다. CORBA는 클라이언트와 서버의 개발 언어가 서로 다르다 하더라도 이와 상관없이 개발자는 IDL(Interface Definition Language)에 정의된 인터페이스 메서드를 호출하여 사용할 수 있다. 원격지에 존재하는 메서드를 마치 로컬에 있는 메서드를 호출하는 것과 같이 사용할 수 있다. 그리고 COM[7]은 서로 다른 언어로 만들어진 애플리케이션 또는 소프트웨어의 컴포넌트들이 상호 통신할 수 있도록 이진 코드 레벨에서의 표준과 서비스를 말한다. COM은 일종의 규약으로 클라이언트와 서버 간에 상호 통신할 수 있도록 표준화된 방법을 규정한 사항이다. 서로 다른 시스템에서 COM을 통한 클라이언트-서버 간 통신으로 물리적 위치에 상관없이 COM 컴포넌트를 사용할 수 있다.

2.2 안드로이드에서의 프로세스간 데이터 전송 기법

여러 모바일 플랫폼 중 안드로이드 플랫폼은 클립보드와 인텐트 그리고 컨텐트 프로바이더 등의 데이터 전송 메커니즘을 제공한다. 안드로이드 플랫폼에서 클립보드를 사용하기 위해서 ClipboardManager[8] 클래스를 사용한다. ClipboardManager는 API 레벨 1에서 Char- Sequence getText, Boolean hasText, void setText (charSequence text)와 같은 3가지 메서드를 지원한다. 각 메서드는 클립보드에 텍스트를 저장하고, 텍스트가 저장되어 있는지에 대한 여부 검사 그리고 데이터를 가져오는 역할을 한다. 클립보드는 안드로이드의 시스템 서비스와 연결하여 사용하며 위의 3가지 메서드를 지원하기 때문에 매우 간단하게 구현할 수 있다. 또한 API 레벨 11부터 ClipData 타입을 지원하면서 데이터 전송을 위한 getPrimaryClip, hasPrimaryClip과 같은 메서드를 제공한다[11]. 하지만 객체 전송을 위해 객체를 레코드로 변환하는 마샬링하고 레코드를 객체로 변환하는 언마샬링하는 과정을 컨텐트 프로바이더를 사용하여 개발자가 부가적으로 구현해야한다. API 레벨 11 이전의 클립보드는 스트링 타입의 데이터만을 지원했기 때문에 애플리케이션 간에 전송할 수 있는 데이터가 제한적이라는 한계점이 있었다. 그리고 API 레벨 11에서 그러한 한계점을 보완하기 위한 ClipData의 지원으로 인해 데이터를 처리할 수 있지만 객체를 처리하는 부분은 여전히 개발자의 몫으로 남아있기 때문에 기존의 클립보드의 방법보다 쉽게 객체를 다른 애플리케이션에 전달할 수 있게 하는 메커니즘이 필요하다.

안드로이드 플랫폼에서 지원하는 데이터 전송 메커니즘으로 인텐트와 컨텐트 프로바이더가 있다. 인텐트는 안드로이드에서 특정 액티비티에 메시지를 전달하는 방법으로 메시지에 데이터를 실어 데이터를 전송하는 방법의 하나로 사용이 가능하다. 인텐트를 전달하기 위해 호출할 대상을 정확하게 찾을 수 있도록 가급적이면 상세한 정보가 작성되어야 한다. 이러한 인텐트의 정보는 Action, Category, Data, Component, Type, Extra와 같은 인텐트 요소에 담을 수 있다. 애플리케이션 간의 객체 전송을 위해서는 인텐트 요소를 모두 정의하여 사용해야 한다. 하지만 인텐트는 어떤 액션이 수행되어야 하는지에 대한 메시지를 전달하기 위해 사용하는 메커니즘으로 데이터 전송이 주목적인 메커니즘이 아니다. 인텐트의 요소들을 사용하여 인텐트의 정보를 구성하는 방식을 사용하기 때문에 데이터를 전송하기 위해 대상 액티비티의 정보를 모를 경우 전송이 불가능하다. 또한 인텐트로 값을 전달할 때 객체를 전달할 수 없다.

안드로이드는 보안 정책상 애플리케이션이 생성한 데이터는 기본적으로 해당 애플리케이션 내에서만 접근이 가능하다. 애플리케이션 외부로 데이터를 공개하여 접근이 가능하도록 하기 위해서 컨텐트 프로바이더를 제공한다[10]. 컨텐트 프로바이더는 약속된 URI를 통해 어떤 컨텐트 프로바이더와 통신할 것인지 결정된다. 안드로이드 시스템은 URI를 이용하여 컨텐트 제공자를 찾고 컨텐트 프로바이더를 로드한 뒤, 컨텐트 프로바이더의 메서드를 호출한다. 따라서 서로 약속한 URI를 사용해야 한다. 컨텐트 프로바이더를 사용하여 애플리케이션의 데이터를 다른 애플리케이션에서 접근 가능하도록 하기 위해서는 해당 애플리케이션의 컨텐트 프로바이더를 작성해야한다. 또한 데이터를 저장하는 저장소를 이용하기 때문에 파일 또는 데이터베이스를 저장소로 사용할 경우 데이터를 로드하고 저장하기 위한 질의를 처리해야 한다.

2.3 C로 작성된 코드의 Java 프로그램으로의 변환

모바일 기기에서 C 기반의 코드를 Java로 변환하거나[15] C 코드를 안드로이드 코드로 변환하는 기법들이 연구되었다[16]. 이 기법들은 기존에 만들었던 프로그램을 다른 플랫폼으로 변환하는 것으로 본 연구에서 논의하고 있는 애플리케이션 간의 객체 전송과는 다른 관점에서 진행된 연구들이다. 이 연구들에서 제시하는 기법은 프로그램의 다양한 요소들을 변환하는 기법을 제시하고 있지만, 본 연구에서는 객체를 애플리케이션, 즉 프로세스 범위를 넘어서서 전달할 경우 이를 간편하게 하는 방법을 제시하고 있다.

3. 메타서비스의 설계

본 논문에서 제안하는 메타서비스는 클립보드와 컨텐트 프로바이더의 객체 전송방법을 개선하기 위한 메커니즘이다. 3장에서는 메타서비스가 클립보드와 컨텐트 프로바이더와 비교했을 때 어떤 차이점과 이점이 있는지 기술한다. 또한 메타서비스를 설계하기 위해 사용자가 활용할 수 있는 예시를 보이고 메타서비스의 형태를 설명한다.

```
   // Copy EditCopy text to the ClipBoard
1: private void copyToClipBoard() {
2:     ClipboardManager clipMan = (ClipboardManager)
           getSystemService(Context.CLIPBOARD_SERVICE);
3:     clipMan.setText(string);
4: }

   // Paste ClipBoard Text to EditPaste
5: private void pasteFromClipBoard() {
6:     ClipboardManager clipMan = (ClipboardManager)
           getSystemService(Context.CLIPBOARD_SERVICE);
7:     String str = clipMan.getText();
8: }
```

(그림 1) 안드로이드의 클립보드 사용(API Level 1)

3.1 기존의 클립보드와 컨텐트 프로바이더

안드로이드에서 제공하는 클립보드 스트링 타입의 데이터를 전송하기 위한 방법이 매우 간단하지만 객체를 전송하기 위한 방법은 복잡하다. 본 논문에서 제안하는 메타서비스는 안드로이드의 클립보드의 스타일과 흡사한 형태로 사용할 수 있다. 더불어 객체의 전송도 스트링 타입의 데이터를 전송하는 형태로 구현이 가능하다.

(그림 1)은 안드로이드의 클립보드 사용 예이다. 클립보드는 (그림 1)에서 보는 바와 같이 getSystemService를 호출하여 클립보드 시스템 서비스에 열결한다. get-System Service는 android.os에 포함되어 있는 Service Manager 클래스를 이용하여 시스템 서비스에 연결해주는 메서드이다. ServiceManager 클래스는 클립보드 외에 다른 시스템 서비스 연결에도 사용되는 클래스이다. 이와 같이 서비스 연결이 완료되면 클립보드의 메서드를 사용하여 간단히 스트링 타입의 데이터를 전송할 수 있다. (그림 1)에서 보는 바와 같이 전송하고자 하는 텍스트를 setText를 사용하여 전송하고 getText를 사용하여 전송된 텍스트를 받을 수 있다. setText와 getText와 같은 간단한 API로 스트링을 전송할 수 있기 때문에 사용이 간편하다. 하지만 객체를 전송하기 위한 메서드는 없기 때문에 객체 전송을 위한 지원이 필요하다. 본 논문에서 제안하는 메타서비스에서는 클립보드의 텍스트 전송과 유사한 방법으로 간단한 메서드를 사용하여 객체의 전송도 지원하도록 하였다.

안드로이드의 클립보드는 (그림 2)에서 보는 바와 같이 API 레벨 11부터 데이터 전송을 위한 ClipData 타입을 지원한다. 텍스트만을 전송하는 것이 아닌 클립을 전송할 수 있도록 확장되었다. 하지만 이 방법은 컨텐트 리졸버에 데이터를 요청하고 커서를 사용하여 레코드를 읽어들여야 한다. 따라서 레코드에 접근하기 위한 코드를 프로그래머가 직접 작성해야 한다. 또한 클립보드에서도 URI를 사용하여 컨텐트 제공자를 찾는 방식이기 때문에 URI를 알 수 없는 경우나 약속된 URI가 없는 경우 애플리케이션 간 객체 전송이 불가능하다.

안드로이드에서 클립보드의 목적의 단순한 복사, 붙여넣기를 간편하게 하기 위한 메커니즘이다. 따라서 컨텐트 프로바이더에서 약속된 URI를 사용하고 저장소에서 원하는 객체를 찾기 위한 작업은 최소화 시킬 필요가 있다. 따라서 클립보드의 텍스트 전송을 위한 메서드와 마찬가지로 개발자가 아주 손쉽게 사용할 수 있도록 객체 전송 또한 지원되어야 한다.

```
1: ClipboardManager clipboard =
   (ClipboardManager)getSystemService(Context.CLIPBOARD
    _SERVICE);
2: ContentResolver cr = getContentResolver();

   // Set ClipData Info
3: ClipData clip = clipboard.getPrimaryClip();
4: ClipData.Item item = clip.getItemAt(0);
5: Uri uri = item.getUri();

6: ArrayList<Rect> arr = new ArrayList<Rect>();
7: if (Data.CONTENT_ITEM_TYPE.equals(cr.getType(uri))) {
8:     Cursor c = cr.query(uri, PROJECTION, null, null, null);
9:     while(c.moveToNext()) {
10:        if (c.moveToFirst()) {
11:            x1 = c.getString(1);
12:            y1 = c.getString(2);
13:            x2 = c.getString(3);
14:            y2 = c.getString(4);
15:            arr.add(new Rect(x1, y1, x2, y2);
16:        }
17:    }
18:}
```

(그림 2) 안드로이드의 클립보드 사용 (API Level 11)

컨텐트 프로바이더를 이용하여 객체를 전달할 경우에는 (그림 2)의 8~15번째 라인과 같은 코드를 작성해야 한다. 그러므로 클립보드에서 보는 바와 같이 객체를 레코드로 마샬링하는 코드를 객체마다 작성해야 하는 번거로움이 있다.

3.2 메타서비스의 활용 및 분석

메타서비스를 사용하는 형태는 안드로이드에서 제공하는 클립보드에서 객체를 전송하는 형태와 유사하다. (그림 3)과 (그림 4)는 본 논문에서 제안하는 메타서비스를 사용하여 객체를 저장하고 로드하는 예시이다.

```
1: MSManager msmgr =
   MSManager.getMSManager(context);

2: int w = 20; int y = 20;
3: Rect rect1 = new Rect(0, 0, w, y);
4: Rect rect2 = new Rect(0, 0, w/2, y/2);

5: List<Rect> items = new ArrayList<Rect>();
6: items.add(rect1);
7: items.add(rect2)

8: msmgr.setObject(items);
```

(그림 3) 메타서비스를 이용하여 객체를 메타서버로 전달

메타서비스를 사용하고자 (그림 3)의 1번째 라인처럼 애플리케이션은 모두 MSManager를 생성해야 한다. MSManager 생성은 getMSManager를 호출하여 생성할 수 있다. 2~4번째 라인과 같이 애플리케이션에서 생성한 여러 개의 Rect 객체를 생성하고 전달하기 위해 8번째 라인의 setObject를 호출하여 리스트 객체를 담아 MetaServer로 전송한다. 이와 같은 작업을 기존의 방법으로 코드를 작성하는 경우 (그림 2)의 6~18번째 라인과 같이 객체를 레코드의 형태로 변환하는 작업을 수행해야 하지만 메타서비스를 이용할 경우 (그림 3)의 8번째 라인과 같이 간단한 코드로 이를 처리할 수 있어 프로그램을 효율적으로 작성할 수 있어 애플리케이션을 개발할 때 생산성을 증진시킬 수 있다.

```
1: MSManager msmgr =
   MSManager.getMSManager(context);
2: ArrayList<Rect> arr;
3: arr = (ArrayList<Rect>)msmgr.getObject();
```

(그림 4) 메타서비스를 이용한 애플리케이션에서의 객체 로드

(그림 4)는 애플리케이션에서 전송된 객체를 로드하는 예시이다. getObject를 호출하여 간단하게 Meta-Server에 존재하는 객체를 애플리케이션으로 로드한다.

기존 방법에서는 (그림 2)의 7~15번째 라인과 유사하게 레코드를 읽어 객체로 변환하는 복잡한 코드를 작성해야 하지만 메타서비스를 이용할 경우 (그림 4)의 3번째 줄과 같이 간단하게 전달된 객체를 얻을 수 있다.

기존 방법을 이용하여 객체를 전달할 경우 전달할 때는 객체를 레코드 형태로 변환하는 마샬링 코드를 작성해야 하고, 전달받을 때는 레코드를 객체로 변환하는 언마샬링 코드를 작성해야 한다. 하지만 메타서비스를 이용하면 (그림 3)과 (그림 4)에서 보는 바와 같이 앞에서 언급한 변환과정을 거칠 필요가 없기 때문에 버그를 줄일 수 있고 코드를 간략하게 작성할 수 있어 애플리케이션을 개발할 때 생산성을 증진시킬 수 있다.

메타서비스를 사용하면 (그림 3, 4)와 같이 애플리케이션 간의 객체를 전송할 때 setObject, getObject를 사용한다. 해당 애플리케이션 내에서 객체 전송이 가능하고 다른 프로세스의 애플리케이션에서도 객체 전송이 가능하다. (그림 3)과 (그림 4)의 코드는 각각 다른 프로세스에서 객체를 전송하는 과정을 나타낸 코드이다. 메타서비스를 사용할 경우 그림에서 보는 바와 같이 다른 프로세스에서도 객체의 전송이 원활하게 이뤄지기 때문에 매우 간편한 방법으로 객체를 처리할 수 있다. 그리고 위와 같이 매우 가독성이 높은 간단한 코드로 객체를 전송할 수 있다. 3.1절의 클립보드와 컨텐트 프로바이더의 예시와 비교해 보았을 때, 보다 간결한 코드로 객체 전송을 위한 동작을 구현할 수 있는 것을 확인할 수 있다. 따라서 기존의 컨텐트 프로바이더를 사용하여 객체를 마샬링하고 언마샬링하는 과정을 구현하는 복잡한 과정을 거치지 않고 매우 간결한 코드로 객체를 전송할 수 있다. 또한 이는 객체 전송을 위한 애플리케이션 개발에 매우 효율적으로 활용할 수 있다.

4. 메타서비스의 구현 방법

본 논문에서 제안한 메타서비스는 안드로이드 플랫폼에서 사용하기 위해 서비스로 구현하였다. 서비스는 안드로이드에서 RPC(Remote Procedure Call)를 가능하게 하는 메커니즘이다. 메타서비스를 안드로이드의 서비스로 제공하는 방법에는 2가지가 있다. 첫 번째로 안드로이드 시스템에 직접 시스템 서비스로 등록하는 방법과 두 번째로 시스템 서비스가 아닌 안드로이드의 원격 서비스[17]로 메타서비스를 등록하는 방법이 있다[18]. 안드로이드의 시스템 서비스로 등록하는 방법은 플랫폼을 수정해야하기 때문에 안드로이드 플랫폼의 커널을 다시 빌드해야 한다. 따라서 커널을 재빌드할 수 없는 환경에서 메타서비스를 사용하기 위해 두 번째 방법은 메타서비스를 안드로이드 시스템의 원격 서비스를 사용하는 방법이 있다. 원격 서비스는 안드로이드 시스템에서 백그라운드로 수행되는 서비스로 여러 프로세스가 IPC (Interprocess Communication)을 통해 프로세스간 원격지의 메서드를 호출할 수 있게 하는 기능을 한다.

4.1절에서는 메타서비스를 안드로이드 서비스로 등록하기 위해 어떤 안드로이드 서비스가 제공되고 있는지에 대해 설명하고 4.2절에서는 메타서비스를 안드로이드 서비스로 등록하는 2가지 방법과 활용 방법에 대해 기술한다. 4.2.1절은 안드로이드 시스템 서비스로 메타서비스를 등록하는 방법을 설명하고 4.2.2절에서는 원격서비스로 메타서비스를 제공하는 방법에 대해 기술한다. 그리고 마지막으로 4.3절에서는 메타서비스에서 지원하는 객체 타입에 대해 설명한다.

4.1 원격지 함수를 호출하기 위한 안드로이드 서비스

안드로이드 플랫폼에서 하나의 프로세스는 다른 프로세스의 메모리에 접근할 수 없다. 프로세스 간 통신을 위해서는 운영체제가 이해할 수 있는 형태로 객체를 분해하고 프로세스의 경계를 넘어 객체를 전송하는 과정이 필요하다. 이러한 과정을 개발자가 구현한다면 개발자는 많은 양의 코드를 작성해야 한다. 안드로이드 플랫폼은 RPC(Remote Procedure Call)를 이용하여 원격지의 메소드를 호출할 수 있도록 스텁(stub)코드를 자동으로 생성해 주는 AIDL(Android Interface Definition Language)이라는 툴을 제공한다[17]. 안드로이드에서 AIDL을 사용하여 IPC 서비스를 구현하기 위해 맨 처음 단계로 AIDL 파일을 생성한다. AIDL 파일은 클라이언트에서 사용 가능한 메서드와 필드를 정의하는 인터페이스를 정의한 파일이다. 그리고 정의된 인터페이스의 메서드를 구현한다. AIDL 컴파일러는 AIDL 파일에 정의된 인터페이스를 통해 Java 언어로 된 인터페이스를 자동으로 생성해준다. 이 인터페이스는 해당 인터페이스를 상속하는 스텁이라는 내부의 추상 클래스를 포함한다. 사용자는 이 스텁을 확장시켜 AIDL 파일에 선언한 메서드가 자신이 원하는 동작

을 수행하도록 구현한다. 그리고 마지막으로 인터페이스를 서비스를 사용하는 클라이언트에게 제공한다. 이와 같이 안드로이드 플랫폼에서는 서비스를 사용하기 위한 방법을 제공해준다.

앞서 언급한 과정을 거쳐 메타서비스는 생성한 파일을 사용하여 서비스에 등록하고 사용자에게 인터페이스를 제공할 수 있다. 메타서비스는 서버 역할을 하는 부분과 각각의 애플리케이션이 클라이언트 역할을 하기 때문에 안드로이드 시스템에 상주하고 있어야 한다. 따라서 메타서비스는 AIDL을 사용하여 안드로이드 시스템의 서비스로 등록하여 사용한다.

4.2 메타서비스를 활용하기 위한 등록 방법

메타서비스를 활용하기 위해서 안드로이드의 서비스로 등록하는 과정이 필요하다. 메타서비스를 서비스로 등록하는 방법에는 2가지가 있다. 안드로이드의 시스템 서비스로 제공하는 방법과 안드로이드의 원격 서비스로 제공하는 방법이 있다. 4.2절에서는 이 2가지 방법에 대해 기술한다.

4.2.1 메타서비스를 시스템 서비스로 제공

메타서비스를 안드로이드 시스템 서비스에 직접 등록하기 위해서는 서비스 stub 구현 파일과 stub 파일 그리고 서비스 인터페이스인 aidl 파일이 필요하다. 안드로이드 프레임워크에는 시스템이 부팅할 때에 서비스 등록을 담당하는 SystemServer.java 파일이 있다. 메타서비스를 시스템 서비스로 등록하기 위해서는 안드로이드 커널에 있는 SystemServer.java 파일을 수정해야 한다. SystemServer.java 파일에 메타서비스를 추가하기 위해 아래와 같이 addService()를 사용한다.

```
ServiceManager.addService("metaservice",
                          new MetaService(context));
```

위의 코드를 SystemServer.java 파일에 추가한 후 안드로이드 커널 전체를 빌드하는 과정을 거치면 안드로이드 플랫폼에 메타서비스 등록을 완료할 수 있다. 안드로이드 시스템에 등록된 서비스는 SystemManager 클래스를 사용하여 간편하게 사용할 수 있다. 그리고 시스템 서비스로 등록이 완료된 메타서비스는 (그림 5)와 같이 사용할 수 있다.

(그림 5)와 같이 안드로이드의 시스템 서비스로 등록한 메타서비스를 사용할 경우 ServiceManager 클래스를 이용하여 getService로 메타서비스를 얻은 후에 안드로이드 시스템 서비스와 같은 형식으로 사용이 가능하다. 하지만 앞에서 상기한 바와 같이 모바일 기기에 시스템 서비스로 등록하여 사용하는 방법은 안드로이드 커널 전체를 빌드해야 하는 과정이 필요하다. 따라서 커널 빌드 과정이 없으면 메타서비스를 사용할 수가 없다. 시스템 서비스로써 등록된 메타서비스의 테스트 및 활용은 새로운 커널을 포팅할 수 있는 임베디드 장치에서 가능하다. 따라서 안드로이드 커널을 재빌드하여 사용할 수 없는 환경에서는 이 방법을 사용할 수 없다.

```
IBinder b;
b = ServiceManager.getService("metaservice");
IMetaService ms = IMetaService.Stub.asInterface(b);
ms.setText("test");
```

(그림 5) 시스템 서비스로 등록한 메타서비스 사용

4.2.2 메타서비스를 원격 서비스로 제공

메타서비스를 사용하기 위한 방법으로 안드로이드의 원격 서비스를 활용한 서비스 등록 방법을 사용해야 한다. 본 논문에서는 안드로이드 시스템 서비스로 등록은 가능하지만 시스템 서비스로 등록하여 사용하는 방법은 커널을 재빌드해야 한다. 따라서 메타서비스를 활용가능 하도록 안드로이드의 원격 서비스로 제공한다. 4.2.1에서 언급한 바와 같이 안드로이드 시스템 서비스로 등록하는 것은 일반 스마트 기기에서 사용할 수 없기 때문에 원격 서비스로 등록하는 것이 활용성이 높다. 애플리케이션에서 메타서비스를 사용하기 위해 해당 애플리케이션과 서비스 간의 바인딩과정이 필요하다. 바인딩은 클라이언트 프로그램이 서비스가 제공하는 동작들을 수행하기 위해 상호 연결하는 과정으로 MSManager 랩퍼 클래스가 담당하는 역할이다

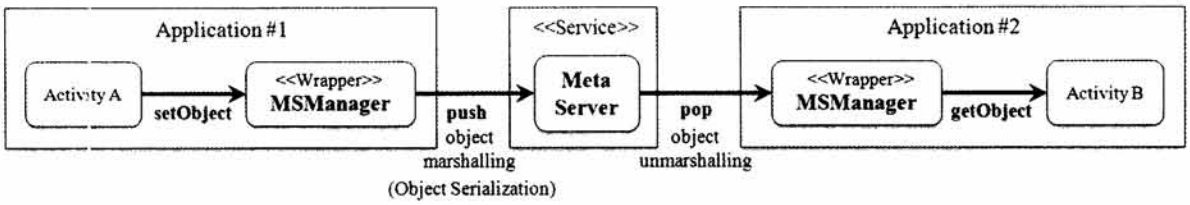
```
MSManager msmgr = MSManager.getMSManager(context);
Object object = new Object();
msmgr.setObject(object);
```

(그림 6) 원격서비스로 등록한 메타서비스 사용

MSManager는 클라이언트인 애플리케이션에서 메타서비스(원격 서비스형태)로 연결하기 위한 메서드를 제공한다. 애플리케이션에서 메타서비스의 서비스를 호출하고 연결하는 과정을 각각 구현하는 것이 가능하지만 편의성을 증대시키기 위해 애플리케이션과 메타서비스를 연결하는 과정과 객체를 마샬링하고 언마샬링하는 작업을 MSManager로 구현하였다. MSManager 클래스를 이용하여 개발자는 메타서비스에 연결 할 수 있다. 메타서비스의 메서드를 시스템 서비스로 사용하는 형태와 유사한 형태로 편리하게 사용할 수 있다.

4.3 메타서비스에서 지원하는 객체 타입

Java 언어는 모든 입출력 데이터를 스트림 형태로 주고받는다. 즉, 전송이 가능하도록 객체를 바이트 스트림으로 저장하고 파일이나 데이터베이스에 저장한 후 추후 객체를 사용하기 위해 바이트 스트림을 객체로 만들어 재구성해야 한다. 따라서 안드로이드 애플리케이션에서 사용하는 Java



(그림 7) 메타서비스 구조

에서 데이터를 처리할 경우 객체를 스트림으로 변경하는 작업이 필요하다. 이러한 작업을 객체 직렬화(serialization)라고 한다[19, 20, 21].

객체 직렬화가 가능한 타입은 기본형과 문자열 그리고 배열과 같은 기본적인 타입의 객체로 전송할 때 바이트 스트림을 사용하여 전송한다. 모든 타입의 객체를 직렬화할 수 있는 것은 아니다. 따라서 메타서비스에서는 객체 직렬화 가능한 객체뿐만 아니라 직렬화 가능하지 못한 객체도 애플리케이션 간의 전송이 가능하도록 2가지 타입의 객체를 처리할 수 있도록 지원한다. 직렬화 가능하지 못한 객체에 대해서는 지속성(Persistent) 객체 타입을 지원하여 애플리케이션 간의 데이터 전송이 가능하도록 하였다. 지속성 객체와 직렬화 객체 2가지에 대한 스트림 연산이 다르기 때문에 각각 객체의 타입을 구분할 수 있도록 시그니처(signature)를 사용하였다. 객체의 타입을 구분하여 전송하는 알고리즘은 5장 메타서비스의 구현 부분에서 자세히 다룬다.

안드로이드 애플리케이션을 구현할 때 사용되는 여러 객체에 대해서 개발자는 지속성 객체와 직렬화 객체 2가지를 사용하여 객체를 생성하고 메타서비스로 데이터 전송을 요구할 수 있다. 또한 메타서비스를 사용하는 애플리케이션은 2가지 객체 타입에 대해 처리가 가능하기 때문에 개발자 입장에서 각각의 객체 타입에 따라 전송하기 위해 구현하였던 스트림처리를 신경 쓰지 않아도 된다는 이점이 있다.

5. 메타서비스의 구현

5장에서는 메타서비스의 전체적인 구조와 각각의 기능에 대해 언급하고 객체 전송 알고리즘에 대해 기술한다. 그리고 메타서비스에서 제공하는 API에 대해 설명한다.

5.1 메타서비스의 구조

(그림 7)은 메타서비스의 구조를 나타낸 것이다. 메타서비스는 클라이언트-서버 구조이다. 메타서비스는 전송받은 객체를 처리하기 위한 MetaServer와 MetaServer로 객체를 전송하기 위한 클라이언트 역할의 애플리케이션으로 구성된다. MetaServer는 안드로이드의 원격 서비스로 안드로이드 시스템에 존재한다. 애플리케이션에서 생성한 객체를 전송하기 위해 객체를 MetaServer에 전송하고 여러 애플리케이션에서 MetaServer에 있는 객체를 로드하는 형태이다. 애플리케이션과 MetaServer 간 통신을 위해 애플리케이션은 MSManager(Meta Service Manager)를 사용한다. MSManager는 메타서비스를 사용하기 위해 MetaServer에 연결을 요청한다. 그리고 연결이 성립되면 해당 애플리케이션은 MetaServer로 객체 전송이 가능하다. 메타서비스는 가독성을 높이기 위해 클립보드와 유사한 형태로 getObject, setObject, getText, setText와 같은 메서드를 제공한다. 객체를 전송하는 과정은 setObject를 사용하고 객체를 전달받는 과정은 getObject를 사용한다. 4가지 메서드 모두 메타서비스를 사용하기 위해서 MetaServer와 연결하는 과정이 필요하다.

각각의 애플리케이션은 서로 다른 프로세스로 동작한다. 그리고 각각의 애플리케이션은 MSManager를 포함한다. MSManager는 랩퍼 역할과 MetaServer와의 연결을 담당한다. (그림 7)의 애플리케이션1은 객체를 전송하기 위해 MSManager 객체를 얻기 위해 MSManager 클래스의 getMSManager를 호출한다. 애플리케이션 컨텍스트를 인자로 받아 MetaServer와 바인딩하는 과정을 거친다. 연결이 완료되면 MetaServer로 객체를 푸시(Push)할 수 있다. 이때 MSManager는 MetaServer로 전송될 객체를 마샬링한다. 이 과정이 끝나면 MSManager는 MetaServer에게 변환된 객체를 스트림으로 전송한다. 전송된 스트림은 MetaServer에 유지된다. MetaServer는 전송받은 마샬링된 객체를 가지고 있고 여러 애플리케이션에서는 원하는 시점에 언제든지 로드할 수 있다.

객체를 로드하는 과정은 앞서 언급한 객체를 전송하기 위한 방법과 마찬가지로 애플리케이션은 getMS- Manager 메서드를 호출하여 MetaServer와 애플리케이션을 연결하는 과정을 거친다. 해당 애플리케이션에서 객체를 전달받기 위해 MSManager는 MetaServer에게 객체를 요청한다. 이 때 객체는 MetaServer에 미리 전송되어 있던 객체이다. MetaServer는 객체를 pop하여 애플리케이션으로 전송한다. 객체를 해당 애플리케이션으로 전송하는 과정에서 MSManager는 스트림을 애플리케이션에서 요청한 객체로 변환하는 언마샬링 과정을 거친 후 애플리케이션으로 객체를 전송한다.

MetaServer로 전송받은 객체는 또 다른 객체가 전송되었을 경우 사라진다. 이는 안드로이드에서 제공하는 클립보드와 동일하게 일회성 객체 전송을 목적으로한 것이다. MetaServer에 전송된 객체는 다른 객체가 저장되기 전에 사라지지 않기 때문에 여러 애플리케이션에서 로드해서 사용할 수 있다. 서로 다른 프로세스에서 객체를 전송하기

```
1: setObject(Object obj)
2: if obj instance of Persistent then
3:     ByteArrayOutputStream baos =
       new ByteArrayOutputStream()
4:     DataOutputStream out = new DataOutputStream(dos)

5:     out.write(PERSISTENT_SIGNITURE)
6:     out.write(obj.getClass().getName())

7:     Persistent p = (persistent)obj
8:     p.write(out)
9:     MetaServer의 setByteArray(baos.toByteArray()) 호출

10: else obj instance of Serializable then
11:    ByteArrayOutputStream baos =
       new ByteArrayOutputStream()
12:    ObjectOutputStream out =
       new ObjectOutputStream(baos)

13:    out.write(SERIALIZABLE_SIGNITURE)
14:    out.writeObject(obj)

15:    MetaServer의 setByteArray(baos.toByteArray()) 호출
16: end if
17: end setObject
```

(알고리즘 1) MetaService의 setObject(): object marshalling

위해 구현해야 했던 객체의 마샬링과 언마샬링 과정을 메타서비스에서 담당함으로써 객체의 전송을 매우 편리하게 하였다.

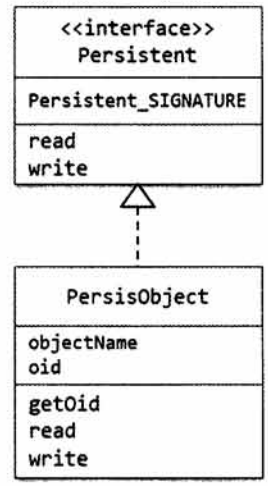
5.2 메타서비스의 객체 전송 알고리즘

(그림 7)에서 보는 바와 같이 MSManager는 객체를 MetaServer로 푸쉬(Push)한다. 이때 객체를 전송하기 위해 스트림 형태로 변환하는 마샬링 과정이 필요하다.

(알고리즘 1)은 setObject 호출로 객체를 MetaServer로 전송하는 과정이다. 객체의 타입은 Persistent와 Serializable 타입 2가지로 구분하여 전송한다. 4.3에서 언급했던 2가지 객체 타입에 대해 전송하는 알고리즘이다. 각각 객체 타입을 검사한다. Persistent 타입의 객체는 DataOutputStream을 사용하고 Serializable 타입의 객체는 ObjectOutputStream을 사용한다. 그리고 각각의 스트림에 담기 전에 시그니쳐를 사용하여 전달받는 쪽에서 객체의 타입을 검사할 수 있도록 (알고리즘 1)의 4, 12번째 줄과 같이 각각의 시그니쳐를 기록한다.

Persistent 타입의 객체는 사용자가 Serializable 객체가 아닌 객체를 전송하기 위한 타입으로 메타서비스에서는 Persistent 인터페이스를 제공한다. (그림 8)과 같이 Persistent 객체를 만들기 위해서는 Persistent 인터페이스를 구현하여 사용할 수 있다. Persistent 인터페이스는 데이터 스트림처리를 위한 read와 write 메서드를 제공한다.

(알고리즘 2)는 (그림 7)에서 객체를 언마샬링하는 과정에 쓰이는 알고리즘이다. getObject를 호출하여 수행되는 과



(그림 8) Persistent Interface

정으로 객체 마샬링 과정에서 스트림에 기록하였던 시그니쳐를 추출하여 Persistent와 Serializable 2가지 객체의 타입을 구분한다. 각각 다른 스트림형식으로 전송하였기 때문에 타입에 맞게 다시 스트림을 객체로 복구해야 한다. (알고리즘 2)의 4~12번째 라인까지는 객체의 시그니쳐가 Persistent일 경우 처리 과정이다. 객체의 클래스 이름을 얻고 새로운 인스턴스를 할당 받아 객체를 설정한다. 설정한 객체에 전송받은 객체를 담아 반환한다.

그리고 (알고리즘 2)의 13~17번째 라인까지는 Serializable 타입일 경우 처리과정으로 readObject를 호출하여 객체를 읽어들이고 반환한다. setObject는 전송하고자 하는 객체를

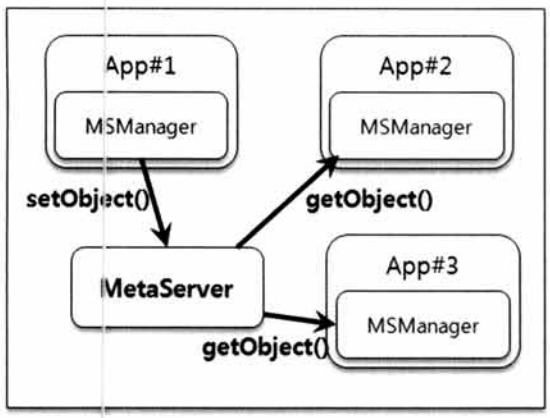
```
1: getObject()
2: MetaServer ms;
3: byte[] byte = ms.getByteArray()
4: objectType ← byte의 앞의 SIGNITURE만 추출

5: if objectType is Persistent then
6:     ByteArrayInputStream bais =
       new ByteArrayInputStream(byte)
7:     DataInputStream in = new DataInputStream(bais)
8:     String className = in.read()
9:     Class c = Class.forName(className)
10:    Object obj = c.newInstance()
11:    Persistent p = (Persistent)obj
12:    p.read(in)
13:    return P

14: else objectType is Serializable then
15:    ByteArrayInputStream bais = new
ByteArrayInputStream(byte)
16:    ObjectInputStream in = new ObjectInputStream(bais)
17:    Object obj = in.readObject()
18:    return Object
19: end if
20: end getObject
```

(알고리즘 2) MetaService의 getObject(): object unmarshalling

바이트 스트림으로 변환하는 마샬링 과정을 거쳐 변환된 스트림을 MetaServer로 전송한다. 그리고 getObject는 MetaServer로 받은 객체를 언마샬링하는 과정을 거쳐 애플리케이션으로 전달한다. MetaServer는 객체를 Persistent 하게 가지고 있기 때문에 애플리케이션이 객체를 전송하고 멈추거나 사라졌을지라도 객체를 사라지지 않고 존재하여 다른 애플리케이션에서 객체를 전송받을 수 있다.

5.3 메타서비스에서 제공하는 API

메타서비스는 안드로이드 플랫폼에서 제공하는 AIDL을 사용하여 구현되었다. 따라서 애플리케이션에서 메타서비스를 사용하기 위해서는 메타서비스에서 제공하는 인터페이스를 포함하고 있어야 한다. 메타서비스의 MSManager는 개발자가 객체 전송하기 위해 사용할 수 있는 메서드를 제공한다.



(그림 9) 메타서비스에서 제공하는 API

(그림 9)는 메타서비스에서 애플리케이션 간 객체를 전달할 때 사용하는 메서드를 나타낸 것이다. 메타서비스의 MSManager는 메타서비스와 애플리케이션에서 호출되는 메서드를 연결시켜주는 랩퍼 클래스이며 Meta- Server와의 연결을 담당하는 클래스이다. 개발자는 애플리케이션의 객체를 메타서비스를 이용해 전송하고 싶을 때 MSManager를 포함해야 한다. 객체를 전송하는 과정이 있기 전에 MetaServer와 해당 애플리케이션을 연결하는 과정이 필요하다. 이 때 getMSManager 메서드를 호출한다. 그리고 MetaServer와의 연결이 성립되면 (그림 9)에서 보는 바와 같이 전달하고 싶은 객체는 setObject 메서드를 사용하고 객체를 전달 받기 위해 getObject 메서드를 이용한다. 그리고 hasObject메서드를 제공하여 객체 전송 부분을 구현할 때 MetaServer에 객체의 저장 유무를 검사할 수 있다. 안드로이드 클립보드에서 스트링 타입의 데이터를 전송하는 것은 동일하게 동작하고 그와 유사한 형태로 객체를 전송하는 것을 구현할 수 있다. 안드로이드에서 제공하는 기존의 클립보드와 유사한 형태를 가지고 있어 간단한 사용이 가능하고 스트링 타입의 데이터뿐만 아니라 객체도 손쉽게 전송할 수 있다.

<표 1>은 메타서비스와 기존의 컨텐트 프로바이더 및 클립보드와 객체 전달 방법을 비교한 것이다. 표에서 보는 바와 같이 컨텐트 프로바이더와 클립보드는 데이터를 레코드 형태로 전송하기 때문에 이를 객체로 변환하는 과정이 필요하다.

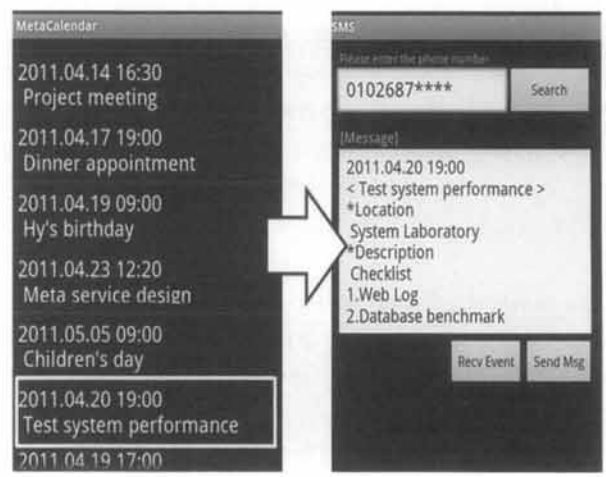
〈표 1〉 데이터 전달 기법 비교

기법	데이터 전달 방법	객체 마샬링 지원
클립보드	레코드	지원 안함
컨텐트 프로바이더	레코드	지원 안함
메타서비스	객체	지원

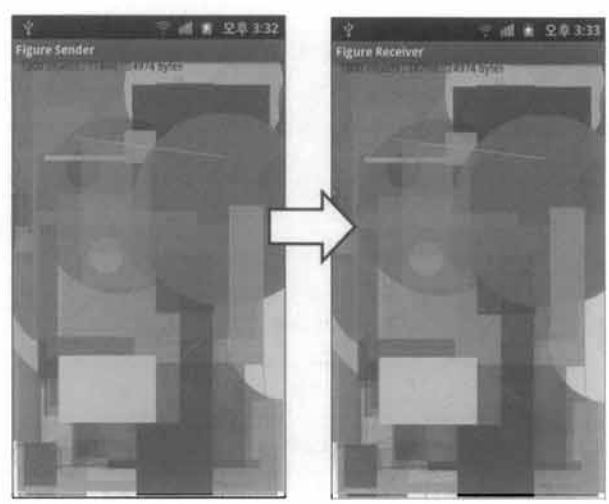
6. 메타서비스의 평가

6.1 메타서비스의 사용 예

5.2절에서 명시한 API를 사용하여 메타서비스의 동작을 확인하기 위해 예제 프로그램을 구현하였다.



(그림 10) 메타서비스를 이용한 애플리케이션 간 객체 전송



(그림 11) 메타서비스를 이용한 애플리케이션 간 객체 전송

(그림 10)과 (그림 11)은 객체(Object)를 전달하는 예제 프로그램이다. (그림 10)은 안드로이드의 내장캘린더의 정보를 가져와서 보여주는 캘린더 애플리케이션에서 선택한 일정에 대한 정보를 SMS 애플리케이션으로 전달하는 화면이다. 3.2절에서 보였던 메타서비스의 활용 예시 코드를 기반으로 작성한 프로그램이다. (그림 10)의 두 개의 애플리케이션은 객체를 전송하기 위해 메타서비스를 이용한다. 캘린더 애플리케이션의 리스트에서 메시지로 원하는 일정을 전송하고 싶을 때 원하는 일정의 타이틀을 클릭하면 setObject 메서드가 호출되어 메타서비스의 MetaServer로 객체를 전송한다. 객체는 일정 정보를 담고 있는 리스트 객체이다. (그림 10)의 우측의 SMS 애플리케이션에서는 MetaServer에서 담겨 있던 객체를 전송받는다. 이 객체는 캘린더에서 가져온 객체를 전송 받은 것이며 getObject 메서드를 사용한다.

(그림 11)은 애플리케이션에서 생성한 그림 객체를 다른 애플리케이션으로 메타서비스를 사용하여 전송하는 예제이다. (그림 10)의 예제 프로그램과 마찬가지로 set-Object 메서드와 getObject 메서드를 사용하여 객체를 전송한다. 위의 예제 프로그램처럼 원하는 객체를 메타서비스에서 제공하는 setObject, getObject와 같은 API를 사용하여 처리할 수 있다. 개발자가 객체를 전송하기 위해 따로 객체를 마샬링하고 언마샬링하는 부분을 구현하지 않아도 되기 때문에 보다 손쉽게 객체를 전송하는 부분을 구현할 수 있다. 이와 같이 메타서비스의 사용으로 애플리케이션에서 발생하는 객체를 전송하기 위한 처리를 편리하게 할 수 있다. 또한 애플리케이션 간의 객체를 전송하는 방법이 간단하기 때문에 다양한 객체를 처리할 때 효율적이다.

6.2 메타서비스의 성능 평가

메타서비스에서 객체를 전송하거나 받을 때의 시간을 측정하였다. 실험에서 (그림 11)에서 보인 그림 객체를 무작위로 생성하여 전송하였다. 그 결과는 <표 2>와 같다.

실험은 Galaxy S2에서 실험하였으며, 각 객체수 마다 5회씩 반복하여 평균을 구하였다. <표 2>에서 100개의 그림 객체의 크기는 평균 2,773 바이트였다. 전송하는데 사용한 그림 객체 하나의 크기는 평균적으로 16 바이트였는데 객체에 대한 메타 정보를 기술하기 위하여 순수한 객체 크기보다 전달되는 데이터의 크기가 증가한 것을 알 수 있다. 이것을 메타서비스로 전송하는데 29ms가 소요되었다. 그리고 이 객체들을 클라이언트 애플리케이션에서 수신하는데 걸린 시간은 46ms였다. 전송할 때는 객체를 마샬링해서 전달하면 되지만, 송신할 때는 전달된 바이트를 언마샬링한 후 객체를 생성해야 하기 때문에 전송하는데 걸린 시간보다 더 많은 시간이 소요된다. 동일한 데이터를 컨텐트 프로바이더에서 전송하였을 경우에는 객체를 전송하는데 많은 시간이 소요되었다. 그 이유는 컨텐트 프로바이더의 경우 데이터를 데이터베이스에 저장하는데 스마트 폰의 저장매체는 NAND 플래시 메모리를 사용하기 때문이다. 플래시 메모리는 읽기 연산보다 쓰기 연산이 10배 가량 느리고, 쓰기를 수행할 경우 블록을 소거하는데 소거 연산은 쓰기 보다 10배 가량 느리다[22]. 그러므로 많은 객체를 컨텐트 프로바이더를 이용하여 전송할 경우 송신할 때 시간이 많이 걸리는 것을 알 수 있다. 컨텐트 프로바이더로부터 데이터를 읽어들이는 연산은 상대적으로 빨리 수행되는데 이것은 플래시 메모리에서 읽기 연산은 빨리 수행되기 때문이다.

실험에서 보는 바와 같이 메타서비스에서 객체를 송신하거나 수신하는데 걸리는 시간은 상당히 짧다는 것을 알 수 있다. 객체들은 보통 유저 인터페이스에서 선택 후 전달하는 것이 일반적인 사용방법이므로 메타서비스는 스마트 폰에서 사용하기에 적합한 성능을 보인다는 것을 알 수 있다.

7. 결 론

최근 스마트 폰에서 발생하는 사용자 컨텍스트와 메타데이터를 활용하여 부가가치를 창출하는데 목적을 둔 연구가 활발히 진행중이다. 사용자 컨텍스트와 메타데이터를 활용하기 위해서 여러 스마트 폰 또는 스마트 폰 내의 여러 애플리케이션 간의 원활한 객체 전송을 위한 메커니즘이 필요하다. 스마트 폰의 애플리케이션에서 발생하는 데이터와 컨텍스트를 전송하기 위해 다른 프로세스간의 통신을 위한 메커니즘이 필요하다. 따라서 본 논문에서는 다양한 모바일 플랫폼 중 안드로이드 플랫폼에서 애플리케이션 간 객체 전송을 효율적으로 지원하기 위한 메타서비스를 설계하고 구현하였다. 안드로이드 플랫폼에서 제공하는 데이터 전송 메커니즘인 클립보드, 인텐트 그리고 컨텐트 프로바이더는 객체 전송을 위해 객체를 마샬링하고 언마샬링하는 과정을 개발자가 구현해야 하기 때문에 번거롭고 구현 과정이 복잡하

〈표 2〉 그림 객체를 전송에 소요된 시간 측정 결과

객체 개수	데이터 크기 (bytes)	메타서비스		컨텐트 프로바이더	
		송신 시간(초)	수신 시간(초)	송신시간(초)	수신시간(초)
10	544	0.008	0.008	0.624	0.010
100	2,773	0.029	0.046	6.344	0.023
1,000	25,001	0.128	0.244	69.708	0.037

다. 따라서 여러 애플리케이션에서 발생하는 객체를 손쉽게 전송할 수 있는 메커니즘이 필요하다.

본 논문의 연구 결과는 다음과 같다. 첫째, 다양한 모바일 플랫폼 중 안드로이드 플랫폼에서 애플리케이션 간 객체 전송을 효율적으로 지원하기 위한 메타서비스를 설계하고 구현하였다. 둘째, 메타서비스의 MetaServer에서 객체를 전달받고 전송하기 때문에 애플리케이션 개발자는 객체의 타입을 고려하지 않고 간단하게 객체를 전송하는 과정을 처리할 수 있도록 하였다. 셋째, 메타서비스를 사용함으로써 객체를 마샬링하고 언마샬링하는 부분에 대해 구현하는 번거로움을 제거하여 보다 효율적인 애플리케이션 개발을 할 수 있도록 하였다. 넷째, 메타서비스에서 제공하는 API는 가독성이 높은 메서드로 구성하여 보다 손쉽게 구현이 가능하도록 하였다.

이와 같은 메타서비스를 애플리케이션 간의 객체 처리를 위한 메커니즘으로 활용하면 한 애플리케이션에서 생성한 객체를 다른 애플리케이션으로 쉽게 전달할 수 있다. 이러한 기능을 이용하면 안드로이드 플랫폼에서 객체를 전송하기 위한 표준 인터페이스로 활용할 수 있고, 이 기능을 더욱 확장하면 윈도우의 COM처럼 한 애플리케이션에서 만든 데이터를 다른 애플리케이션에서 활용할 수 있게 될 것으로 예상된다.

참 고 문 헌

[1] M. Raento, A Oulasvita, R. Petit, H. Toivonen, "ContextPhone: A Prototyping Platform for Context-Aware Mobile Applications", IEEE Pervasive Computing, Vol.4, No.2, pp.51-59, 2005.

[2] J. H. Hong S. I. Yang, S. B Cho, "A Context-aware Messenger for Sharing User Contextual Information", Journal of KIISE : Computing Practices and Letters, Vol.14, No.9, pp.906-910, Aug., 2007(in Korean).

[3] A. Krause, et al., "Context-Aware Mobile Computing: Learning Context-Dependent Personal Preferences from a Wearable Sensor Array", IEEE Transactions on Mobile Computing, Vol.5, No.2, Feb., 2006.

[4] A. KEN, G. JAMES, H DAVID, Java Programming Language Forth Edition "java.rmi-Remote Method Invocation", Addison-Wesley, 2008.

[5] C. Nester, M. Philippsen, B. Haumacher, "A more efficient RMI for Java", In Proceedings of the ACM Java Grande Conference, pp.152-159, 1999.

[6] "Common Object Request Broker Architecture (CORBA) Specification, OMG, Ver.3.1, Jan., 2008.

[7] Box D., "Essential COM", Addison-Wesley Professional, 1997.

[8] ClipboardManager, Android Developers: http://developer.android.com/reference/android/content/ClipboardManager.html

[9] Sayed Hashimi, Satya Komatineni, Pro Android 2 "intent", Apress, 2010.

[10] Sayed Hashimi, Satya Komatineni, Pro Android 2 "Content Provider", Apress, 2010.

[11] ClipData, Android Developers: http://developer.android.com/reference/android/content/ClipData.html

[12] Agarwal, S., Starobinski, D., Trachtenberg, A., On the Scalability of Data Synchronization Protocols for PDAs and Mobile Devices, IEEE Network, Vol.16, 2002. 8.

[13] Byung-Gon Chun, Petros Maniatis, Intel Research Berkeley, Augmented smartphone applications through clone cloud execution, HotOS'09 Proceedings of the 12th conference on Hot topics in operating systems, 2009.

[14] Joonmyung Kang, Takkyun Ko, Sinseok Seo, Baekjae Sung, John Strassner, Jong Kim, Chanik Park, James Wonki Hong, Smart Mobile Platform for Supporting Context-aware Services, Journal of KIISE Vol.28 No.5, May, 2010.

[15] 신수원, 최윤석, 정기원, "모바일 C 기반의 GVM 응용프로그램을 Java 기반으로 변환하는 process", 한국정보과학회 학회지, 제29권, 제2호, 2002.

[16] 손윤식, 오세만, 이양선, "소스 레벨 콘텐츠 변환기를 이용한 GNEX C-to-Android Java 변환기의 설계 및 구현", 멀티미디어학회논문지, 제13권, 제7호, 2010년 7월.

[17] AIDL(Android Interface Definition Language) http://developer.android.com/guide/developing/tools/aidl.html

[18] Jerome Dimarzio, Android A Programmer's Guide, MC Graw Hill

[19] Android Serializable, Android Developers: http://developer.android.com/reference/java/io/Serializable.html

[20] Ken Arnold, James Gosling, David Holmes, The Java Programming Language, Fourth Edition

[21] Bryan Carpenter, Geoffrey Fox, Sung Hoon Ko, Sang Lim, "Object serialization for marshalling data in a Java interface to MPI", JAVA'99, 1999.

[22] 박성환, 장주연, 서영주, 박원주, 박상원, "플래시 변환 계층에 대한 TPC-C 벤치마크를 통한 성능 분석", 정보과학회 논문지 : 컴퓨팅의 실제, 제13권, 제4호, 2008년 4월.

최 화 영

e-mail : hychoe@dislab.hufs.ac.kr

2008년 한국외국어대학교 정보통신공학과 (학사)

2011년 한국외국어대학교 컴퓨터및정보통신공학과(석사)

2011년~현 재 현대엠엔소프트

관심분야 : 데이터베이스, 모바일 컴퓨팅, 플래시 메모리

박 상 원

e-mail : swpark@hufs.ac.kr

1994년 서울대학교 컴퓨터공학과(학사)

1997년 서울대학교 컴퓨터공학과(석사)

2002년 서울대학교 컴퓨터공학과(박사)

2002년~2003년 세종사이버대학교 디지털 콘텐츠학과 전임강사

2003년~현 재 한국외국어대학교 정보통신공학과 부교수

관심분야 : 모바일 컴퓨팅, 데이터베이스, 플래시 메모리