한국정보전자통신기술학회논문지(jkiiect)'15-04, Vol.8 No.2
논문 15-08-02-086 http://dx.doi.org/10.17661/jkiiect.2015.8.2.086

Java SDK를 이용한 파일 클라우드 시스템의 설계 및 구현

이상곤*

Design and Implementation of File Cloud Server by Using JAVA SDK

Samuel Sangkon Lee*

요 약 클라우드 컴퓨팅은 IT 관련된 기능들이 서비스 형태로 제공되는 컴퓨팅 스타일이다. 사용자들은 전문 지식이 없거나 제어할 줄 몰라도 인터넷으로부터 서비스를 이용할 수 있다. 정보가 인터넷 상의 서버에 영구적으로 저장되고 데스크탑 컴퓨터, 노트북, 휴대용 기기 등과 같은 클라이언트에는 일시적으로 보관된다. 이러한 클라우드 컴퓨팅에서 소프트웨어와 데이터는 서버에 저장된다. 클라우드 컴퓨팅은 웹 2.0, SaaS(software as a service)와 같이 최근 잘 알려진 기술 경향들과 연관성을 가지는 일반화된 개념이다. 이들 개념들의 공통점은 사용자들의 컴퓨팅 요구를 만족시키기 위해 인터넷을 이용한다는 사실이다. 이는 단순히 서버 등의 자원을 제공해 주면서 사용자가 디바이스에 제약없이 데이터에 접근할 수 있도록 해준다. 개인적인 용도의 파일을 저장하고 이를 여러 디바이스에서 공유하는 클라우드 서비스가 주목을 받고 있다. 본 논문에서는 Dropbox와 OAuth, PACloud를 통해 이와 같은 서비스를 구현할 수 있다. 또한 스레드 폴링을 이용하여 서버에 들어오는 여러 태스크들을 적절하게 처리할 수 있는 구현 기술을 제시하였다. 아울러 구현 기술을 설명하기 위해 소프트웨어 공학적인 여러 다이어그램을 제시하였다.

Abstract Cloud computing is a computing term that evolved in the late 2000s, based on utility and consumption of computer resources. Google say that “Cloud computing involves deploying groups of remote servers and software networks that allow different kinds of data sources be uploaded for real time processing to generate computing results without the need to store processed data on the cloud. Cloud computing relies on sharing of resources to achieve coherence and economies of scale, similar to a utility (like the electricity grid) over a network. At the foundation of cloud computing is the broader concept of converged infrastructure and shared services. Cloud computing, or in simpler shorthand just "the cloud", also focuses on maximizing the effectiveness of the shared resources.” The cloud service is a smart and/or intelligent service to save private files in any device, anytime, anywhere. Dropbox, OAuth, PAClous are required that the accumulated user’s data are archives with cloud service. Currently we suggest an implementation technique to process many tasks to the cloud server with a thread pooling. Thread pooling is one of efficient implementating technique for client and service environment. In this paper, to present the implementation technique we suggest three diagrams in the consideration of software engineering.

Key Words : Cloud Service, Thread Pooling, Multi Threading, Agile, Java, Wait/Notify, Dropbox API, PACloud,

1. 서론

과거 저장 공간에 있는 데이터를 다른 공간으로 옮기기 위해서는 데이터의 이동 매체인 디스크, CD-ROM, 외장 하드 디스크 등에 옮겨야 한다. 그렇기 때문에 불편함은 물론 비용까지 소모된다. 그러나 현대 사회는 유비쿼터스(Ubiquitous)화가 가속되며 많은 변화를 겪고 있다. 그 가속을 부추긴 것은 바로 IPV4(Internet Protocol Version 4)이며, 이 프로토콜은 많은

* Corresponding Author : Department of Computer Science and Engineering Professor of Jeonju University(samuel@jj.ac.kr)
Received march 08, 2015 Revised march 20, 2015 Accepted april 6, 2015

기기를 하나로 모아 연동할 수 있는 기술이다. 현대에는 스마트 폰 어플을 이용하여 대학 도서관의 좌석을 확인하고 열람실 자리에 대해 대여 신청도 할 수 있다. 이와 같이 이제는 원격으로 모든 것을 제어할 수 있는 세상이며, 사물 인터넷의 저장 공간 또한 유비쿼터스화가 진행 중이다. 이제 저장 공간은 과거의 물리적인 형태에서 가상의 형태로 바뀌고 있으며, 웹 포털 사이트에서도 개인별 웹 하드를 충분히 제공하고 있다.

미래에는 클라우드 시스템을 도입해 자기 PC의 저장 공간 일부를 사이버 상에서 공유하고, 다른 디바이스에서도 곧바로 접근할 수 있는 환경을 만들어 다른 서비스들과 통합하여야 한다. 이와 함께 스마트 폰 이용자 수가 늘어남에 따라 현대인의 생활 패턴도 많이 바뀌고 있다. 기존에는 하나의 장소에서 업무를 보았다면 지금은 이동 중에도, 심지어 길을 걸으면서 장소에 구속받지 않고 업무를 볼 수 있는 환경이 되어가고 있다. 또한 USB 메모리에 저장하던 개인 자료나 업무 파일들을 인터넷 상의 이메일로 저장하는 이용자도 많아졌다. 하지만 인터넷의 경우 개인 용량의 한계가 있고, 개인적인 문서들이 인터넷 서버에 그대로 남아 있음으로써 자료의 보안성 및 안전성이 떨어진다. 이를 해결하는 방안으로 클라우드 서비스(Cloud Service)가 주목받고 있다. 이 서비스는 앞에서 언급한 용량의 문제를 해결해 주며, 자료의 보안성을 보장하기 위해 개인의 가상 디스크를 제공하는 등 최신 트렌드로 주목받고 있다. 앞으로의 발전 가능성 및 차세대 컴퓨팅 환경에도 이러한 가상 디스크는 많이 사용될 것이며, 능률 및 생산성 향상 측면에서 그 활용성은 매우 높다.

그림 1. 여러 디바이스의 결합 관계
Fig. 1. The Combination of Multiple Devices

개발자가 개발한 소프트웨어에서 특정 저장 공간의 접근을 제공하고 클라우드 시스템(Cloud System)으로 다른 PC와의 동기화를 통해 [Fig. 1]과 같이 하나의 외장 하드로 여러 디바이스에서 어디서나 일관성 있는 작업을 할 수 있다. 또한 타 업체에서 제공하는 클라우드 서비스인 구글 드라이브(Google Drive)와 드롭 박스(Dropbox)를 연동하면 저장 장치의 공유도 쉽게 할 수 있다.

본 논문의 개발 방법을 이해하기 위해 우선 클라우드 서비스에 대한 사전 지식이 필요하다. 본 서비스를 이용하기 위해 기본 지식과 사전 조사가 이루어진 다음 개발이 이루어져야 할 것이며, 서버 설계를 보다 체계적으로 해야 할 것이다. 또한 PC와 PC간의 통신이 이루어져야 하기 때문에 네트워크와 보안 관련 연구도 동시에 이루어져야 한다.

2. 서버의 효율 향상을 위한 스레드 폴링

2.1 스레드 폴링의 정의 및 분류

멀티 스레딩(Multi Threading) 기법은 자바(Java)를 강력한 프로그래밍 언어[1, 2, 3]로 만든 특징 중의 하나이다. 이 기법은 프로그래밍 언어 차원에서 지원되기 때문에 간단한 몇 가지 규칙만으로 쉽게 스레드 프로그램을 만들 수 있다.

풀링(Pooling) 기법 중의 하나인 스레드 폴링(thread pooling)은 일반 객체가 아닌 스레드를 풀링하는 기법으로, 기본 원리는 스레드의 재활용성이다. 할당된 일을 마친 스레드를 소멸시키지 않고, 스레드 풀(thread pool)에 저장해 두었다가 필요할 때 다시 꺼내 쓰는 개념이다. 즉, 이 개념은 스레드의 생성과 소멸에 필요한 비용을 지불하지 않겠다는 것이다.

스레드 풀은 처리해야 할 일이 등록되기 전에 생성되는데, 풀이 생성됨과 동시에 스레드들도

생성되어 풀에 대기하게 된다. 지능적인 풀은 처리해야 할 일의 증가 및 감소에 따라서 풀 안의 스레드 개수를 늘리기도 하고, 줄이기도 한다. 만약에 풀에 존재하는 스레드의 수보다 처리해야 할 일의 수가 많다면, 일이 순서대로 처리되도록 디자인 할 수도 있고, 빠른 일 처리를 위해 추가적인 스레드가 생성되도록 풀을 할 수도 있다.

2.2 스레드 폴링의 사용 이유

```
public void run() {
    try {
        while (!_bClosed) {
            // 보통 getTaskFromQue()를 하게되면
            // 태스크가 있는 경우에는 해당
            // Task의 run() 메서드가 실행되고,
            // 없는 경우에는 wait()상태에 빠지게 된다.
            ((Runnable) (_Que.getTaskFromQue())).run();
        }
    } catch (InterruptedException ignore) {
    } catch (ThreadQueClosedException ignore) {
    }
}
```

그림 2. 쓰레드 반복 생성의 한계
Fig. 2. Thread Limit of Iterations

스레드 풀링은 객체 풀링과 마찬가지로 자원의 소모를 줄이는 효과가 있다. 무엇보다 스레드 생성 시에 많은 자원이 필요하고, 제거 역시 만만한 일이 아니다. 그렇기 때문에 자바와 같이 프로그래밍 언어 차원에서 스레드가 지원된다면 스레드가 필요할 때마다 생성하여 처리하고, 필요 없어지면 버리는 일을 반복한다면 효율적인 서버 프로그래밍이 될 수 없다. 특히 스레드라는 것이 보통 외부의 요청을 처리하는 독립적인 역할을 하는 경우가 많기 때문에, 외부 요청이 잦은 경우에 이를 모두 생성하는 방식으로 처리하면 효율이 낮아질 수밖에 없다.

이를 해결하기 위해 스레드 간에 CPU를 점유하기 위한 스레드 스케줄링(thread scheduling)이 필요하다. 모든 스레드는 자신의 기본적인 실행 시간 외에 스레드 스케줄링에 일부 시간을 할애해야 한다. 만일 외부 요청에 의해서 스레드가 한꺼번에 생겼다 사라지면 실제 실행 시간보다는 스레드 스케줄링에 많은 컴퓨팅 자원이 들어갈 것은 당연한 일이다. 따라서 네트워크 서버와 같이 여러 요청이 동시에 들어오는 응용 프로그램에서는 스레드의 관리가 컴퓨터의 실제 성능을 좌우하게 된다.

2.3 스레드 폴링의 원리

첫째로 스레드는 run() 메소드가 끝나지 않으면 종료되지 않는다는 점을 이용한다. 즉 [Fig. 2]와 같이 스레드가 run() 메소드 안에 있으면 해당 스레드는 종료되지 않는다. 따라서 한번 만들어진 스레드를 run() 메소드 안에 가두어 놓으면 된다. 둘째, 자바에서 스레드를 특정 상황에서 정지할 수 있다는 점을 이용한다. 스레드의 정지 없이 무한정 실행하게 하면 부하가 심하게 걸리므로 실행 중인 스레드를 태스크에게 할당하기 전에 실행을 정지하면 된다. 이 두 가지 조건이 잘 만족되면, 스레드 폴링을 하는 중 특정 태스크(task)가 생기면 이를 적절히 실행시키면 된다.

2.4 스레드의 기본 메소드

스레드를 반복 실행하기 위해서는 위의 [Fig. 2]와 같이 스레드를 무한 루프에 가두면 된다. 단, 외부에서 스레드의 완전 정지를 위해 특정 변수의 값을 조정할 수 있도록 해 주어야 프로그램을 종료할 수 있다. 위의 코드에서 볼 수 있는 것처럼 _bClosed 의 값이 참(true)이 되는 순간까지 스레드는 계속 수행된다.

스레드의 정지 및 실행은 다음과 같다. 스레드를 정지/실행하는데 주요하게 사용하는 방식은 Wait/Notify 방식이다. 이 방식은 스레드 간에 협력 작업이 필요할 때 유용하게 사용된다. 먼저 태스크에 스레드가 없는 경우에는 wait() 메서드로 자신을 정지시킨다. 후에 새로운 태스크가 발생하면 동기화 메서드 내에서 notify() 메서드를 호출함으로써 정지한 스레드들을 깨워 태스크를 실행하면 된다.

2.5 태스크 클래스의 관리

보통 스레드에 전달되는 태스크는 스레드가 실행할 수 있는 코드를 가지고 있어야 한다. 보통 이 코드의 제공을 위해 Runnable 인터페이스를 이용해 구현하는 경우가 일반적이다. Runnable 인터페이스로 구현한 태스크 클래스는 스레드가 실행하고자 하는 코드를 run() 메서드에 넣고 그 제어를 스레드 풀링으로 넘겨준다. 그러면 대기하고 있는 스레드가 이 태스크 객체를 실행한다. 만약 태스크가 없다면 스레드들은 wait() 상태로 만들어야 한다. 다음의 [Fig. 3]은 스래드 풀의 처리 과정을 나타내었다.

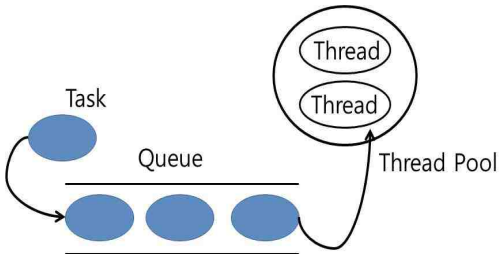


그림 3. 쓰레드 풀의 태스크 처리
Fig. 3. Task Processing Thread Pool

```
private ArrayList _TaskList = new ArrayList();
private boolean _bClosed = false;
private int _WaitNum = 0;

// 객체를 Que에 집어넣고, 관심있는 쓰레드를 깨운다.
public synchronized final void putTaskIntoQue(Object newTask)
        throws ThreadQueClosedException
{
    if (_bClosed)
    {
        throw new ThreadQueClosedException();
    }
    _TaskList.add(newTask);
    notify(); // wait하고 있는 쓰레드중 하나만 깨운다.
}
```

그림 4. 쓰레드 큐의 Notify() 함수
Fig. 4. Notify() of Thread Queue

2.6 스래드 큐의 작성 방법

스레드 큐(thread queue)는 스레드 풀의 핵심적인 모듈이다. 이 큐가 하는 역할은 대기하고 있는 스레드 들에게 태스크 클래스를 넘겨주는 역할을 한다. 만일 적당한 태스크가 없는 경우에는 대기하고 있는 스레드들을 wait() 메서드로 모두 재운다. 만약 새로운 태스크가 들어온 경우에는 [Fig. 4]와 같이 notify() 메서드를 호출하여, 대기하고 있는 스레드들 중 한 개를 깨워 태스크를 넘겨준다.

```
public synchronized final Object getTaskFromQue()
        throws InterruptedException, ThreadQueClosedExceptic
    try {
        while (_TaskList.size() <= 0) {
            _WaitNum++;
            wait();
            _WaitNum--;

            if (_bClosed) {
                throw new ThreadQueClosedException();
            }
        }
        return _TaskList.remove(_TaskList.size() - 1);
    } catch (NoSuchElementException e) {
        throw new Error("ThreadQue에 문제생김");
    }
}

public synchronized final int getWaitNum() {
    return _WaitNum;
}

public synchronized final boolean isEmpty() {
    return _TaskList.size() > 0;
}

public synchronized void close() {
    _bClosed = true;
    notifyAll();
```

그림 5. 쓰레드 큐
Fig. 5. Thread Queue

태스크 객체를 관리할 ArrayList 객체를 생성한다. [Fig. 5]와 같이 스래드 풀이 완료되면 스레드들을 모두 정지시키기 위해서 _bClosed 라는 변수를 선언하고, 새로운 태스크를 ArrayList에 넣는다. 큐에서 태스크를 할당받는 데 처리해야 할 태스크가 없다면 스레드는 wait() 한다. _TaskList의 사이즈가 0 이라는 것은 아무런 스레드도 없다는 것이다. Close() 메서드는 스레드 풀을 종료할 때 사용하며 대기하고 있는 모든 스레드들을 깨운다. 그 전에 _bClosed 값을 참으로 만들어 대기하고 있는 모든 스레드들에게 ThreadQueClosedException이 발생한다.

2.7 스래드 풀링 매니저의 작성

스래드 풀링(thread pooling) 매니저는 외부에서 태스크 객체를 execute() 메서드로 할당 받아 [Fig. 6]과 같이 스레드 큐에 넣어 주는 역할을

```
public synchronized void execute(Runnable action)
        throws ThreadQueClosedException {
    if (_bClosed) {
        throw new ThreadQueClosedException();
    }
    synchronized (_Que) {

        if (_Que.getWaitNum() == 0) {
            new WorkerThread(++_Size).start();
        }

        _Que.putTaskIntoQue(action);
    }
}
```

그림 6. 쓰레드 매니저 1

Fig. 6. Thread Manager 1

한다. 그 외에 [Fig. 7]과 같이 내부 클래스로 WorkerThread 클래스를 유지하고 있는데, 이 내부 클래스는 스레드 풀링 매니저의 생성자에 의해 생성되어 스레드 큐에 태스크를 할당 받는 작업을 한다.

```
private class WorkerThread extends Thread {
    private int _ThreadID;

    public WorkerThread(int threadID) {
        _ThreadID = threadID;
    }

    public void run() {
        try {
            while (!_bClosed) {
                ((Runnable) (_Que.getTaskFromQue())).run();
            }
        } catch (InterruptedException ignore) {
        } catch (ThreadQueClosedException ignore) {
        }
    }
}
```

그림 7. 쓰레드 매니저 2

Fig. 7. Thread Manager 2

태스크 큐(Task Queue)에서 getTaskFromQue() 메서드로 얻은 객체의 run() 메서드를 실행하는데 만일, 태스크의 객체가 없다면 wait() 상태에 빠져들게 된다.

3. 클라우드 서버에 적용

2장에 기술한 스래드 풀링의 기법을 그대로 적용하여 클라우드 서버(cloud server)를 구축하였다[11]. [Fig. 8]은 Runnable 인터페이스로 만든 스래드로 2장에서 소개한 태스크 관리를 위한 것이다. 그리고 ThreadPoolManager의 객체인 매니

```
public class mainSocket implements Runnable {
    public static final int serverPort = 5001;
    public static ThreadPoolManager manager;
    DBControl dbconnect;
    public static String PCStateName = null;


    /**
     * @brief DB접속, 소켓 통신 위한 스레드 실행 메서드
     */
    @Override
    public void run() {

        dbconnect = new DBControl();
        try{
            System.out.println("Wait Server On..");
            ServerSocket server = new ServerSocket(serverPort);
            manager = new ThreadPoolManager(10);

            while(true)
            {
                Socket socket = server.accept();
                System.out.println("Read File.....");

                try{
                    System.out.println("from : " + socket.getInetAddress());
                    SocketThread sender = new SocketThread(socket, dbconnect);
                    manager.execute(sender);
                }catch(Exception e){
                    e.printStackTrace();
                }
            }
        }catch(Exception e){
            e.printStackTrace();
        }
    }
```

그림 8. 런어블 인터페이스에 의한 쓰레드 생성

Fig. 8. Creating a Thread Using Runnable Interface

저를 생성하여 클라이언트의 접속 연결인 태스크를 받아들이기 전에 10개의 유휴 스레드를 생성하여 대기한다. while() 의 무한 루프를 시작하면, [Fig. 8]과 같이 서버는 server.accept()로 클라이언트(Client)의 소켓을 통한 접속을 대기하며, 클라이언트가 접속한 이후 통신하며 계속해서 일을 하게 될 태스크 클래스를 생성하고, 이를 Thraed PoolManager인 매니저에게 매개 변수로 전달하여 유휴 스레드에 일을 하도록 execute() 시켜 원활하게 하였다. 위에서 설명한 방법을 이용하여 본 논문에서 제안하는 클라우드 서버를 설계한 시스템 아키텍처는 [Fig. 9], 소프트웨어 아키텍처는 다음의 [Fig. 10]과 같다.

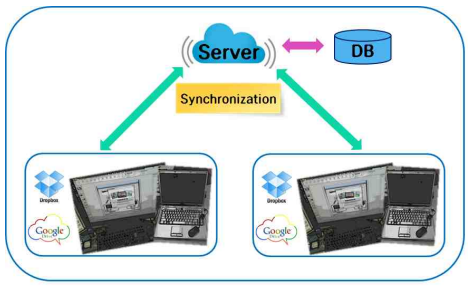


그림 9. 시스템 아키텍처

Fig. 9. System Architecture

4. OAuth 인증의 연구

클라우드 컴퓨팅은 인터넷 기반으로 하는 컴퓨팅 기술로 인터넷상의 유틸리티 데이터를 서버쪽 프로그램에 두고 이 데이터를 다양한 장치로 불러오는 웹 기반 소프트웨어이다. 클라우드 컴퓨팅에는 다양한 서비스가 있는데, 외국 서비스로는 Google Drive, Dropbox, 애플의 iCloud, 아마존의 AWS가 있고, 국내에는 네이버 N 드라이브, 다음 클라우드, KT ucloud 등이 있다. 클라우드 컴퓨팅은 초기 Web 2.0과 SaaS(Software as a Service)와 같이 잘 알려진 기술들과 연관성을 갖는 일반화된 개념으로 IaaS(Infrastructure as a Service), PaaS(Platform as a Service) 등이 있다. 이런 다양한 클라우드 서비스는 여러 기기에서 사용할 수 있게 애플리케이션을 제공하여 언제 어디서나 사용자가 파일을 관리할 수 있다. 따라서 서비스 개발사에서는 누구나 손쉽게 자사 서비스를 활용할 수 있는 API(Application Programming Interface)를 제공한다. 본 논문에서는 서비스 개발사들의 다양한 API를 활용하여 통합된 애플리케이션을 제공하고 사용자가 하나의 프로그램으로 여러 디바이스에서 공유할 파일을 관리하는 프로그램을 개발하였다. 그 중 Dropbox API의 연동 및 인증 위한 Dropbox API[10, 12]에 대해 논의하고자 한다.

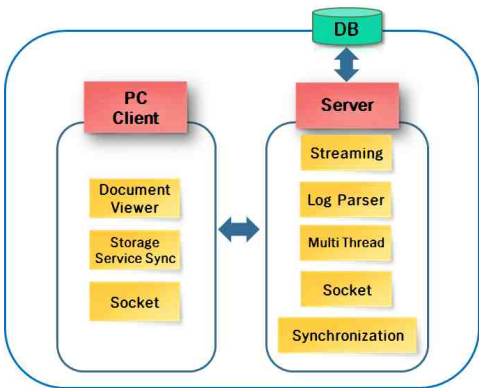


그림 10. 소프트웨어 아키텍쳐
Fig. 10. Software Architecture

본 설명의 범위는 다음과 같이 세 가지의 범위를 설정한다[8, 9].

- OAuth 인증과 Dropbox API에 대한 관련 연구
- OAuth 인증 방식과 용어
- OAuth 인증을 기반으로 한 Dropbox API의 사용 방법

OAuth가 사용되기 전에는 인증 방식의 표준이 없었기 때문에 기존의 기본 인증인 아이디와 비밀 번호를 사용하였는데, 이는 보안상 취약하다. 기본 인증이 아닐 경우는 애플리케이션들은 각 개발 회사의 방법에 따라 적법한 사용자를 확인하였다. OAuth는 각 개발사 제각각의 인증 방식을 하나로 표준화 한 인증 방식이다. OAuth 프로토콜은 웹 사이트나 프로그램(컨슈머)에서 API를 통해서 컨슈머에 대한 사용자의 확인 없이 웹 서비스(서비스 프로바이더)의 보호된 리소스에 접근할 수 있다. 또한 OAuth는 자유로운 구현과 일반적인 API 인증 방법을 만들 수 있게 해준다. OAuth의 목표는 커뮤니티 중심 프로토콜과 웹 서비스의 통합 인증을 구현하는 것이다. OAuth는 존재하는 프로토콜과 독립적으로 구현된 여러 웹 사이트 중에서 한 가지 모범 사례를 바탕으로 구현되었고, 개방형 표준과 크고 작은 업체들에게 지원하는 개발자와 사용자에게 일관되게 신뢰할 수 있는 경험을 장려한다.

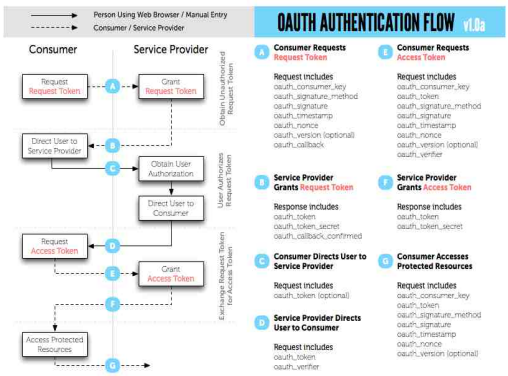


그림 11. OAuth 1.0의 인증 절차
Fig. 11. OAuth 1.0 Authentication Process

논문의 독자를 위해 몇 가지 용어를 정의하면 다음과 같다.

- 서비스 프로바이더(Service Provier) : OAuth를 통해서 접근을 허용하는 웹 애플리케이션.
- 사용자(User) : 서비스 프로바이더에 계정을 가지고 있는 개인.
- 컨슈머(Consumer) : OAuth를 사용해서 사용자를 대신해 서비스 프로바이더에 접근하는 웹사이트나 애플리케이션.
- 보호된 리소스(Protected Resource) : 컨슈머가 인증을 통해 접근할 수 있는 서비스 프로바이더의 데이터.
- 컨슈머 개발자(Consumer Devloper) : 컨슈머를 규현하는 개인이나 단체.
- 컨슈머 키(Consumer Key) : 컨슈머가 서비스 프로바이더에 자신을 나타내는 값. 컨슈머 개발자가 서비스 프로바이더에서 발급한다.
- 컨슈머 시크릿(Consumer Secret) : 컨슈머가 컨슈머 키의 소유권을 입증하는데 사용하는 시크릿. 컨슈머 개발자가 서비스 프로바이더에서 발급한다.
- Request 토큰 : 컨슈머가 사용자에게 인증을 획득하기 위해 사용하는 값. 접근 토큰으로 교체됨. 컨슈머가 서비스 프로바이더에 요청한다.
- Access 토큰 : 컨슈머가 서비스 프로바이더의 사용자 인증을 사용한 사용자 대신 보호된 리소스에 접근 권한을 얻는데 사용되는 값. 컨슈머가 서비스 프로바이더에 요청한다.
- 토큰 시크릿(Token Secret) : 컨슈머가 토큰의 소유권을 입증하는데 사용하는 시크릿. 컨슈머가 서비스 프로바이더에 요청한다.
- OAuth 프로토콜 파라미터(OAuth Protocol Parameter) : 이름이 "oauth_"로 시작하는 파라미터이다.
- OAuth 인증은 소비자와 서비스 공급자 사이에서 일어나는데, 이 인증 과정은 위의 [Fig. 11]과 같다.
- 소비자가 서비스 제공자에게 요청 토큰을 요청한다.
- 서비스 제공자가 소비자에게 요청 토큰을 발급해 준다.
- 소비자가 사용자를 서비스 제공자로 이동시킨다. 여기서 사용자 인증이 수행된다.
- 서비스 제공자가 사용자를 소비자로 이동시킨다.
- 소비자가 접근 토큰을 요청한다.
- 서비스 제공자가 접근 토큰을 발급한다.
- 발급된 접근 토큰을 이용하여 소비자에서 사용자 정보에 접근한다.

Dropbox API는 https://www.dropbox.com/developers 에서 제공한다. Sync API의 종류는 세 가지를 제공하는데, Sync API는 모바일 응용 프로그램을 저장할 수 있는 강력한 보관 및 파일 동기화를 제공하고, Datastore API는 앱의 구조화 된 데이터를 동기화하여 유지하는데 도움이 된다. 마지막으로 서버 기반 애플리케이션을 위한 Core API를 제공한다. Dropbox 사는 사용자의 요구에 따라서 누구나 손쉽게 개발을 할 수 있는 API를 제공한다. 다만 단순한 개발을 위해서는 애플리케이션의 이름만 입력하고 개발할 수 있지만, 최대 접속 인원은 5명으로 제한이 되어 있어 이를 정식 제품화 하여 출시할 때는 기존의 iOS, Android 플랫폼[6, 7]의 애플리케이션과 유사하게 간단한 심사를 받아야 한다. 본 연구에서는 Core API를 사용한다.

사용자는 우선 Drop Box의 회원이 되어야 하며, 간단한 회원 가입 절차를 밟아야 한다. 그리고 위의 인터넷 사이트에서 개발자 등록을 한다. 개발자 등록은 단순히 개발자임을 증명하는 단계를 거치면 되고, 사용자가 개발할 Application의 이름을 작성을 하면 바로 개발에 참여할 수 있는 App Key와 App Secret을 제공하게 된다. 이곳에서 사용자는 App folder와 Full Dropbox를 선택하는데, 이는 개발자가 개발하려는 특성에 맞게 선택하며, Full Dropbox의 경우에는 추후 정식 릴리즈를 할 때 액세스의 보안 수준을 작성해야

승인이 되는 것을 숙지하고 있어야 한다.

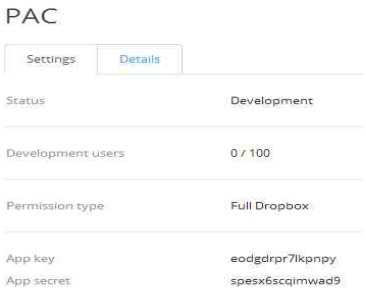


그림 12. 개발자 등록
Fig. 12. Developer Registration

위의 [Fig. 12]에서 App이 등록이 되면 두 가지의 Key를 받게 되는데, 이는 개발을 할 때 꼭 필요한 사항으로 따로 작성하는 것이 좋다. 그리고 개발자는 Additional User를 활성화 하여 초기 개발을 할 수 있다. 이 과정을 무시한다면 Session을 얻지 못하므로 반드시 활성화 해야 하고, 출시 이전의 App일 경우에는 위에서 언급한 바와 같이 최대 5명의 User만 Session을 받을 수 있다.

위의 개발자 등록 과정을 마치면 사용자는 Drop Box Lib를 다운받고 개발을 원하는 Project 메뉴에 첨부해 주면 개발 준비는 완료된다. 하지만 개발에 앞서 개발자 등록을 하여 받은 App Key와 App Secret은 어디에 등록하는지는 나와 있지 않다. 이것을 개발자가 생성시키는 Class에 변수로 설정해야 하며, 이를 통해 Session을 얻어 올 수 있게 된다. 개발자 App 등록 후 출력되는 App name, App key, App secret과 동일하게 작성하고 이를 통해서만 Session을 얻어 올 수 있다. 이와 같이 key를 작성 후에 초기 세션을 얻어 와야 한다. 여기서 Session은 사용자의 로그인 정보가 아닌 애플리케이션에서 사용하기 전 허가를 받는 것을 의미한다. 우선 개발자는 위와 같이 Session을 얻을 수 있는 함수를 생성하고, 이를 통해서 Session을 생성하는 준비를 할 수 있다.

그리고 Application Class의 생성 부분에 [Fig. 13]의 코드를 작성하면 해당 애플리케이션의 Session을 얻어 올 수 있다. 하지만 이 부분에서 정상적인 키 값인지 확인하는 함수가 있어야 한다. 이 작업은 위에서 작성한 개발자의 Key와 비교하고 정상적인 Key 값인지 확인한다. 위의 모든 인증 절차가 끝나면 이제 해당 애플리케이션에서 Dropbox의 사용이 가능하고, 사용자 정보를 받아 사용하면 된다.

Login Session을 얻어 오는 방법은 다음과 같다. 사용자가 원하는 응용 프로그램을 선택하면 사용자의 정보를 얻어 사용자에 해당하는 정보를 Dropbox에 보여준다. 위의 과정은 단순히 응용 프로그램의 사용을 허가하는 인증 절차이다. 사용자에 맞춰서 해당 File List를 보여 주는데, 이 과정은 비교적 간단하다. 위에서 얻은 Session을 가지고 사용자는 자신의 로그인 값을 넣으면 되는데, 이 부분은 개발자가 하지 않고 인증된 App에 맞춰 인터넷에 접속한다.

```
private static OAuthToken GetAccessToken()
{
    var oauth = new OAuth();

    var requestToken = oauth.GetRequestToken(new Uri(DropboxRestApi.BaseUri), ConsumerKey, ConsumerSecret);

    var authorizeUri = oauth.GetAuthorizeUri(new Uri(DropboxRestApi.AuthorizeBaseUri), requestToken);
    Process.Start(authorizeUri.AbsoluteUri);
    Thread.Sleep(700); // Leave some time for the authorization step to complete

    return oauth.GetAccessToken(new Uri(DropboxRestApi.BaseUri), ConsumerKey, ConsumerSecret, requestToken);
```

그림 13. 엑세스 토큰 취득 방법
Fig. 13. Access Token Acquisition Method

```
public bool Login_d(string key,string secret){
    try
    {
        ConsumerKey = key;
        ConsumerSecret = secret;
        var accessToken = GetAccessToken( );
        return true;
    }
```

그림 14. 로그인 정보

Fig. 14. Enter Login Information

위의 [Fig. 14]와 같이 사용자는 처음에 얻은 Session 값을 통해 사용자의 정보를 받아 올 수 있으며, 여기서 사용자는 로그인을 통해서 Dropbox를 사용하면 된다.

```
public Dropboxq.Apiq.FileSystemInfo GetFolder(string root, string path)
{
    var uri = new Uri(new Uri(DropboxRestApi.BaseUri), String.Format("metadata/{0}/{1}", root, path));
    var json = GetResponse(uri);
    return ParseJson<Dropboxq.Apiq.FileSystemInfo>(json);
```

그림 15. 디렉토리 정보의 취득 방법

Fig. 15. Method of Obtaining Directory Information

이제 Drop Box에 접근이 가능하므로 각종 기능에 어떻게 접근을 하는지 설명하고자 한다. 기본적으로 Cloud Service는 File을 Upload, Download, Metadata View로 나눌 수가 있다. Metadata View는 흔히 Application의 File List를 보는 것이 사용자의 파일 정보를 가져온다. 사용 방법은 간단하며, 세부 Directory에 접근하려면 다른 방법을 사용해야 하지만, 기본적인 접근 방법은 다음의 [Fig. 15]과 같다. 위의 [Fig. 15]과 같이 일반적인 List에 저장이 되고, 이를 활용하여 개발자의 특성에 맞게 사용하면 된다. 또한 Upload 및 Download 기능은 [Fig. 16]과 같다. 위의 [Fig. 16]에서 주의할 점은 업로드하는 경로를 작성해 주어야 한다.

```
public void _UpdateFile(string FileName,string FilePath)
{
    var accessToken = new OAuthToken(ConsumerKey, ConsumerSecret);
    var api = new DropboxApi(ConsumerKey, ConsumerSecret, accessToken);
    var file1 = api.UploadFile("dropbox", @"CloudSystem₩" + FileName, FilePath);
}
public void DownLoadFile(String selectFile,String path,downLoad dl)
{
    path += (@"₩" + selectFile);
    var accessToken = new OAuthToken(ConsumerKey, ConsumerSecret);
    var api = new DropboxApi(ConsumerKey, ConsumerSecret, accessToken);
    var file = api.DownloadFile(dl,"dropbox", "CloudSystem/" + selectFile,path);
}
```

그림 16. 업/다운 로드 방법

Fig. 16. Upload and download Methods

5. 구현

본 연구의 구현을 위해 먼저 사용한 개발 모형은 에자일(Agile) 방법론[4, 5]이다. 이 방법론은 프로세스, 도구보다 의사소통을 강조함으로써 요구 사항 변경에 적극적으로 대처할 수 있도록 하는 방법론이다. 문서화 된 계획보다 단위 모듈 개발 결과와 고객의 신속한 요구사항 반영, 그리고 테스트에 중점을 두고 있다. 유스케이스 식별은 다음의 <Table 1>에서 제시하는 세 가지 기능을 구현하였으며, [Fig. 17]은 유스케이스 다이어그램을 제시하였다. 식별 작업은 요구 분석 -> 설계 -> 개발 -> 테스트의 순으로 작성한다. [부록 A]에 이 세 가지 기능의 클래스 다이어그램을, [부록 B]에는 시퀀스 다이어그램을, 그리고 [부록 C]에는 유스케이스 명세서를 제공하였다.

표 1. 유스케이스 다이어그램의 사양

Table 1. Use-case Specification

Function Category	Function(Usecase)
Synchronization	File Synchronization
View of Modification	Display of Modified Files after Synchronization
Sign Up	Find of User's Cloud Storage

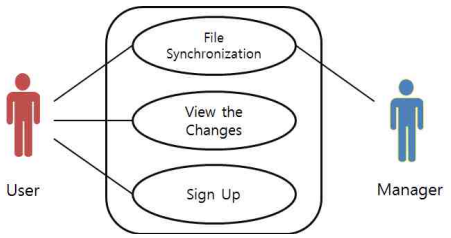


그림 17. 유스케이스 다이어그램

Fig. 17. Usecase Diagram

다음으로 [Fig. 18]에서 [Fig. 32]은 추가적으로 PACloud에 대한 단계별 과정을 그림으로 도식화 한 것이다.

그림 18. 시작 화면
Fig. 18. Starting Screen

그림 19. 시작 화면
Fig. 19. Starting Screen

그림 20.회원 가입 창
Fig. 20. Implementation of Sign Up

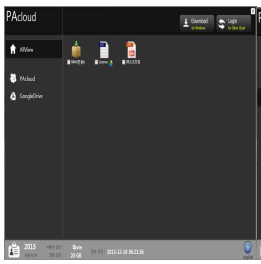


그림 21. 파일 뷰
Fig. 21. File View

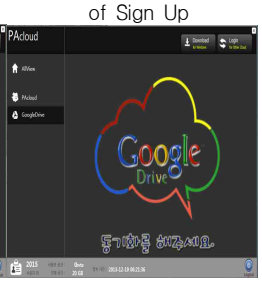


그림 22. 구글 드라이브 (로그인 전)
Fig. 22. Google Drive (Login before)

그림 23. 구글 드라이브 (로그인 후)
Fig. 23. Google Drive (Login)

그림 24. 구글 드라이브 (로그인 완료)
Fig. 24. Google Drive (Log in Complete)

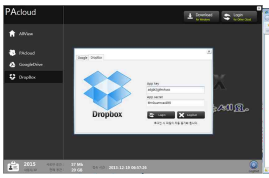


그림 27. 드랍박스의 로그인 창
Fig. 27. Login Windows of Dropbox

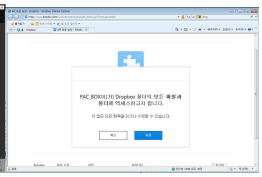


그림 28. 드랍박스 (인증 1)
Fig. 28. Dropbox (Authenfication of OAuth 1)

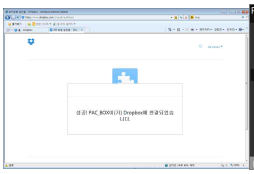


그림 29. 드랍박스 (인증 2)
Fig. 29. Dropbox (Authenfication of OAuth 2)

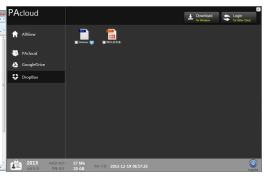


그림 30. 드랍박스의 파일 보기
Fig. 30. File View of Dropbox

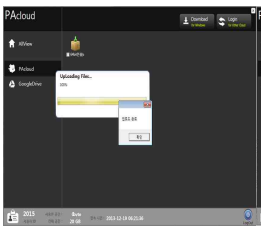


그림 31. 파일 업로드
Fig. 31. File Upload

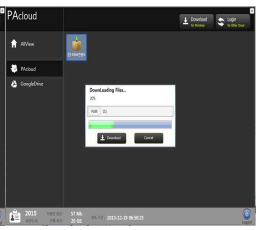


그림 32. 파일 다운로드
Fig. 32. File Download

6. 사용자 인터페이스

본 장에서는 앞에서 언급한 Dropbox의 사용 방법을 소개한다. 사용 방법은 순서대로의 그림이다. 본 연구의 개발 환경은 다음의 [Table 2]와 같다.

표 2. 개발 환경
Table 2. Environment of Development

Item	Specification
CPU	Intel(R) Core(TM) i5-2300 @ 2.80 GHz
MEMORY	4.00 GB
OS	Microsoft Windows 7(32 bit)
Development Tool	Eclipse Juno, Visual Studio 2012
Language	Java Development Kit(JDK) 6, C#

추가 개발을 위해 향후에는 다양한 디바이스에서 사용 가능한 애플리케이션 개발하여야 한다. 서비스 향상을 위한 서버의 멀티 스레드 지원하고, 다른 클라우드 서비스를 통합 지원이 필요하다. 또한 드롭 박스 로그인 과정을 개선하여야 한다.

그림 33. 드랍 박스의 동기화 전

Fig. 33. Dropbox Synchronized before

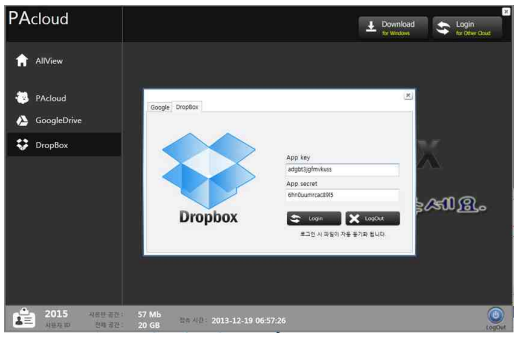


그림 34. 드랍박스의 로그인 창

Fig. 34. Login Window of Dropbox

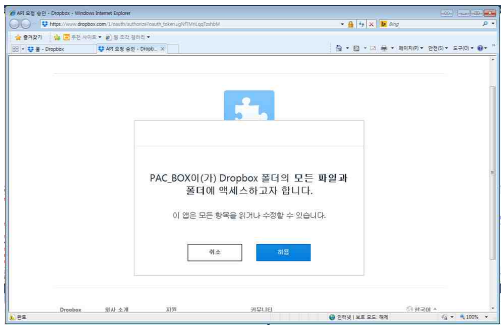


그림 35. 접근을 허용하는 창

Fig. 35. Check whether the Access is Allowed

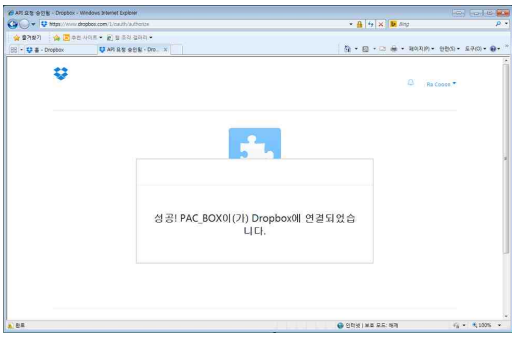


그림 36. 드랍박스의 연결 성공

Fig. 36. Connection Success of Dropbox

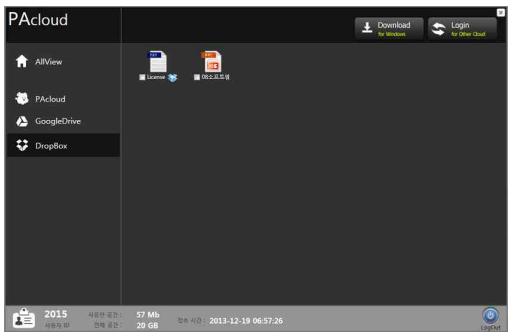


그림 37. 드랍 박스의 동기화 된 파일 목록

Fig. 37. Dropbox synchronized files list

7. 결 론

본 논문에서는 여러 디바이스에서 하나의 저장 공간을 공유하는 클라우드 파일 시스템을 설계하여 제작하였다. 이를 이용하면 Android 와 Windows 에서 동일한 저장 공간을 공유할 수 있다. 또한 다양한 디바이스에서 공간의 제약성을 극복하고 데이터의 손실을 방지할 수 있다. 이 시스템은 실시간 데이터 동기화가 가능하고, 타 클라우드 서비스 계정과 동기화가 가능하며, 하나의 뷰(view)에서 여러 클라우드 시스템을 관리할 수 있는 장점이 있다[13].

본 논문에서는 OAuth 인증과 Dropbox API에 대해서도 조사하였다. OAuth가 사용되기 전에는 인증 방식의 표준이 없어 개발사별 서로 다른 인

증 방식을 사용하였다. 또한 보안상의 취약성이 존재하였지만, OAuth와 같은 표준화 된 인증 방식을 사용하면 보안 문제가 해결되고 인증 과정도 공유할 수 있다. 이러한 장점으로 인해 대부분의 회사에서 OAuth를 지원한다. Dropbox사에서는 역시 OAuth를 기반으로 하는 다양한 API를 지원하여 개발자가 특정 API를 선택할 수 있다. 따라서 본 논문에서도 이를 이용하여 사용자가 폴더를 지정하여 백그라운드 상에서 자동으로 파일을 업로드 하는 기능을 설계하여 구현하였다. 또한, App key를 이용하여 로그인 하지만 차후 Dropbox 아이디로 직접 로그인을 할 수 있도록 지원하고, OAuth의 최신 버전인 2.0으로 프로그램을 작성하고 이를 소프트웨어 공학적인 방법론을 이용하여 테스트하였다.

직접 UI를 개발하지 않고도 기존의 API를 이용하거나, Open Source를 이용해 자신이 필요한 부분을 바로 사용하고, 이를 수정함으로써 더 좋은 프로그램을 제작할 수 있다고 생각한다. 최신의 모바일 플랫폼의 경우에도 이러한 API만을 사용함으로써 클라우드 서버를 손쉽게 개발할 수 있다. 특히 Dropbox를 자신이 개발하는 애플리케이션에 추가함으로써 데이터 보관의 안정성을 높이고, 사용자에게는 편리함을 제공할 수 있다.

또한 스레드 풀링은 해당 어플리케이션이 이용할 CPU의 자원을 미리 점유함으로써 효율적이고 빠른 성능을 보장하는 좋은 기술이다. 미리 할당 받은 스레드들은 태스크를 더욱 빠르게 처리할 수 있다. 그렇기 때문에 한 순간에 갑자기 여러 태스크가 발생하여도 스레드 풀링을 이용하여 미리 얻은 자원으로 단 시간 내에 일을 처리 할 수 있는 것이다. 이를 적용하여 개발한 5장의 PACloud는 서버에 접속하는 클라이언트들의 태스크를 바로 처리할 스레드를 미리 점유하여 갑작스런 많은 사용자들의 접속 폭주를 완만하게 처리해 낼 수 있게 된다. 주의할 점으로는 미리 점유할 자원에 대해서 서버가 스스로 판단할 수 없기 때문에 프로그래머 자신이 서버 특성을 이해하고, 여러 번의 테스트 과정을 거쳐 점유할 자원을 미리 예측하고 연구하는 과정도 필요하다. 서버가 감당할 수 있는 클라이언트의 동시 접속자 수를 가늠하여 확보하여야 하는데, 그 방법으로 동시 접속자 수의 하루 통계치를 내어 시간대마다 유동적으로 유휴 스레드를 조절하여 CPU의 효율을 극대화 하고, 전력 소모를 현저하게 낮출 수 있을 것이다.

추가 개발을 위해 향후에는 다양한 디바이스에서 사용 가능한 애플리케이션으로 개발하여야 한다. 서비스 향상을 위한 서버의 멀티 스레드 지원하고, 다른 클라우드 서비스를 통합 지원이 필요하다. 또한 드롭 박스 로그인 과정을 개선하여야 한다. 그리고 향후 파일을 파편화 하여 동시에 전송하는 멀티 스레드 풀링 기술을 이용한다면 파일을 보다 빠르게, 인터넷 자원을 효율적으로 전송할 수 있는 시스템은 굉장한 파급력을 보일 것으로 전망한다.

REFERENCES

[1] Sanghyun Kim, .Net Programming - C#, Win Form, ADO, Ga-me Publishing, 2008. (Korean)

[2] Yunmyung Kim, JAVA Programming for Brain, Hanbit Media Publishing, 2006. (Korean)

[3] Seungbak Kim, Java I/O & NIO Network Programming, Hanbit Media Publishing, 2004. (Korean)

[4] Eunman Choi, Software Engineering, 5th Ed., Jeongiksa Publishing. 2011. (Korean)

[5] Dongho Han, Android Programming with Example - Step by Step, J-Perm, 2011. (Korean)

[6] Jaekon Jeong, Do it! Android App Programming, Easysper Publishing, 2013. (Korean)

[7] Hyun-hee Jang, Programming WPF, Hanbit Media Publishing, 2008. (Korean)

[8] Daum DNA Development Network, “OAuth Look Around”, http://dna.daum.net/apis/oauth.
[9] Wikipedia, “OAuth”, http://ko.wikipedia.org/wiki/OAuth.
[10] Wikipedia, “Dropbox”, http://ko.wikipedia.org/wiki/Dropbox
[11] Wikipedia, “Cloud Computing”, http://ko.wikipedia.org/wiki/Cloud_Computing.
[12] Dropbox, “Dropbox - Developers”, [Internet], https://www.dropbox.com/developers.
[13] Hakgeon Lee, Changho Yun, Jongwon Park, Yongwoo Lee, "An Analysis of Big Video Data with Cloud Computing in Ubiquitous City," Vol. 15, No. 3, pp. 45-52, 2014. (Korean)

저자약력

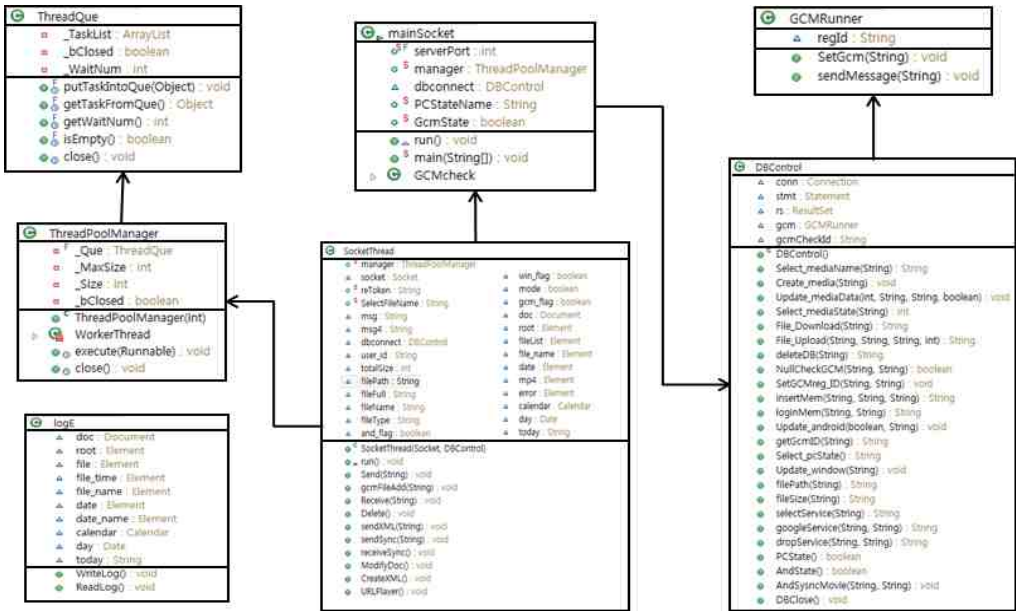
이 상 곤(Samuel Sangkon Lee) [종신회원]

1994년 02월: 전주대학교 영어영문학과 (이학사)
1996년 02월: 전북대학교 컴퓨터과학과 (이학사)
1998년 08월: 전북대학교 전산통계학과 (이학 석사)
2001년 09월: 日本 도쿠시마대학교 지능 정보공학과 (공학 박사)
2002년~현재: 전주대학교 컴퓨터공학과 교수

<관심분야>
클라우드시스템, 자연언어처리, 한글 및 한국어 정보처리

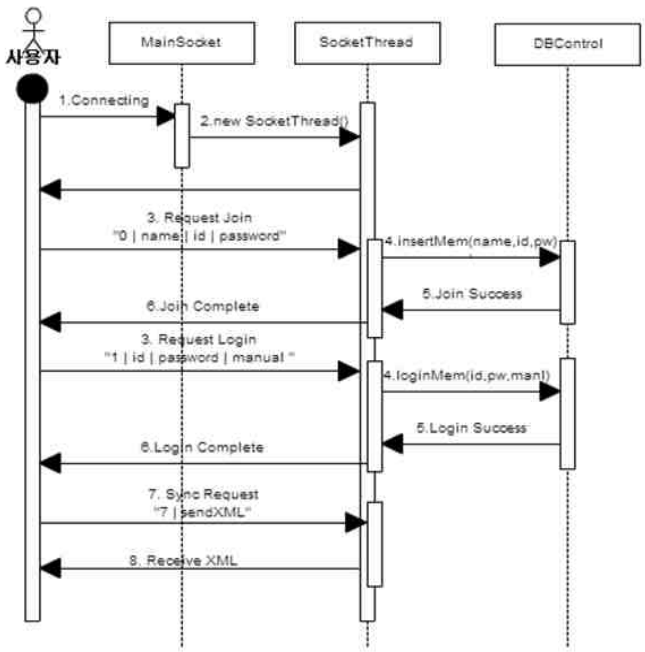
[부록 A] 클래스 다이어그램

[Appendix A] Class Diagram



[부록 B] 시퀀스 다이어그램

[Appendix B] Sequence Diagram



[부록 C] 유스케이스 명세서

[Appendix C] Use-case Diagram

파일 동기화에 대한 유스케이스 명세서

유스케이스 명		파일 동기화
액터명		사용자
설명	개요	파일 동기화 기능은 사용자가 프로그램에 접속하면 자동으로 폴더 및 파일을 동기화 하여, 각 디바이스 간에 동기화된 내용을 가지고 추가, 수정, 삭제 사항을 변경 및 알린다.
	시작 조건	사용자가 클라우드 동기화 할 폴더를 설정하여 파일을 추가, 수정, 삭제하였을 경우
	사건의 흐름	클라우드가 설치된 디바이스에 자동으로 추가, 수정, 삭제 사항을 알린다.
	종료 조건	변경된 파일의 다운로드 혹은 삭제를 하여 동기화를 한다.

변경 사항 보기의 유스케이스 명세서

유스케이스 명		변경 사항 보기
액터명		사용자
설명	개요	변경사항 보기 기능은 동기화된 파일을 최신 내역을 사용자에게 알려준다.
	시작 조건	파일 동기화를 한다.
	사건의 흐름	최신 내역을 조회해 사용자에게 알린다.
	종료 조건	변경 사항을 화면에 표시한다.

회원 가입의 유스케이스 명세서

유스케이스 명		회원 가입
액터명		사용자
설명	개요	회원 가입 기능은 사용자에게 자신만의 클라우드 공간을 사용할 수 있도록 하기 위한 기능이다.
	시작 조건	사용자가 회원 가입을 요청한다.
	사건의 흐름	회원 가입에 대한 명세서를 작성한다. 아이디 중복 등을 확인한다.
	종료 조건	회원 가입 명세서를 서버로 제출한다.