SyncML 서버 데이터 동기엔진의 설계 및 구현

*조진현[0] *최 훈 **김경용
*충남대학교 컴퓨터공학과, **㈜ 네오스텝스
{jhcho, hchoi}@ce.cnu.ac.kr knkim@neosteps.com

Design and Implementation of SyncML Synchronization Engine

*JinHyun Cho[0] *Hoon Choi **KyungNyong Kim
*Dept. of Computer Engineering, Chungnam National University,
**Neosteps, Inc.

요 약

유무선 단말간의 데이터 동기화를 위한 표준인 SyncML을 이용하여 SyncML 서버를 구현하였다. SyncML의 개략적인 설명과 함께 구현한 SyncML 서버 구조를 소개하고, 그 중 Sync Engine에 대하여 기술하였다. Sync Engine은 SyncML 프로토콜에 영향을 주지않고 제공되어지는 서비스 및 정책에 종속적인 역할을 담당하는 프레임이다. Sync Engine을 구현함으로써 사용자에게는 편리하게 다양한 서비스를 사용할 수 있고, 개발자에게는 쉽고 빠르게 서비스의 추가 및 정책 반영을 할 수 있다.

1. 서 론

무선 이동 통신이 보편화되면서 휴대 전화나 PDA(Personal Data Assistant), 또는 휴대용 컴퓨터 등 이동 단말기 사용 인구가 급속도로 증가하고 이러한 이동 단말기를 이용한 개인 정보관리(PIMS : Personal Information Management Service) 프로그램 등 정보와 관련된 서비스 및 데이터의 사용량이 급속도로 증가되고 있다. 또한, 한 사람이 여러 대의 단말기를 사용함으로써 같은 데이터를 여러 단말기에 각각 저장하게 되는데, 여러 단말기들 사이의 데이터를 동기화 하는 작업이 필요하게 되었다. 현재 주요 단말 회사마다 동기화를 수행하는 솔루션이 있으나, 이들이 제공하는 동기화 솔루션은 디바이스, 데이터 타입 및 시스템에 의존적인 서로 다른 프로토콜이기 때문에 데이터의 액세스 및 공유에 한계가 있어서, 동기화 프로토콜의 표준화의 필요성이 제기되었다[1,2]. 이러한 데이터 동기화 표준을 위해 2000년 2월 에릭슨, IBM, 로터스, 모토롤라, 노키아, 팜, 사이언, 스타피시 소프트웨어 등 8개 회사가 SyncML(Synchronization Markup Language) 표준 협의체[1]를 구성하여 현재 동기화 프로토콜 표준을 위한 스펙 1.0.1[3,4,5]을 발표하였다. 이 표준을 이용하여 동기화 솔루션을 구현함으로써 서로 다른 회사 단말들 사이에서도 상호 운용성을 보장할 수 있다.

저자 조진현은 BK21 정보인력양성사업의 지원으로 연구를 수행하였음.

본 연구를 통해 SyncML[1]을 이용한 서버를 설계, 구현하였다. 그 중 Sync Engine[6]은 SyncML 프로토콜에 영향을 주지않고 제공되어지는 서비스 및 정책에 종속적인 역할을 담당하는 프레임이다. 본 논문에서는 SyncML의 간략한 소개와 SyncML의 프레임 구조, 구현한 SyncML 서버의 아키텍처와 그 중 Sync Engine이라는 세부 모듈 구현에 대해 살펴본다.

2. SyncML (Synchronization Markup Language)

데이터 동기화(Data Synchronization)란 여러 이동 단말과 서버에 존재하는 데이터의 내용을 일치시키는 것을 말한다.

SyncML이란 서로 다른 디바이스 및 어플리케이션 간에 데이터 동기화를 제공해 주기 위해 제정된 표준 언어이다[3]. SyncML은 유, 무선간에 데이터 동기화를 제공하며, 데이터 전송을 위해 HTTP, WSP, OBEX 등 다양한 전송 프로토콜을 이용한다. 그리고 일정관리, 주소록등 여러 PIMS 이외의 다양한 서비스를 이용할 수 있도록 확장 가능하다. 또한 명령어를 포함한 데이터 동기화를 위한 모든 정보를 XML DTD 타입으로 정의하고, 일반 텍스트 형태인 XML 뿐만 아니라 바이너리 형태인 WBXML 형식의 메시지를 제공함으로써 쉽고 빠르게 디바이스, 서버간에 데이

터 동기화를 제공한다[2,7,8].

3. 구현된 SyncML 서버 프레임 구조

그림 1은 구현한 SyncML 서버에서 제공하는 기본적인 프레임 구조를 블럭화 한 것이다. 각 구성 요소를 살펴보면, 서버쪽에서 데이터 변경을 수행하는 주체인 Server Application이 있고, 클라이언트 디바이스로부터 전송되어진 메시지를 받아서 툴킷으로 전달하기 위한 SyncML Adapter가 있다. SyncML Toolkit은 전달되어진 메시지를 디코딩하고, 보낼 메시지를 인코딩하는 기능을 담당한다. Sync Agent는 SyncML 프로토콜에 맞도록 메시지를 처리하는 기능을 담당하고 Sync Engine은 그 중 서비스 정책에 종속적이면서 데이터간 충돌이 발생 할 경우 이를 해결해 주는 역할을 담당한다. Session Manager는 동기화를 위한 세션을 유지하고 관리하는 기능을 담당하며, Open DB Interface는 데이터를 저장하기 위한 DB와의 Interface 역할을 한다. 각 프레임은 DLL로 구현하였고, 디바이스와의 통신은 HTTP를 이용하였으며, DLL과의 연결을 위해 JNI를 이용하였다.[5,6]

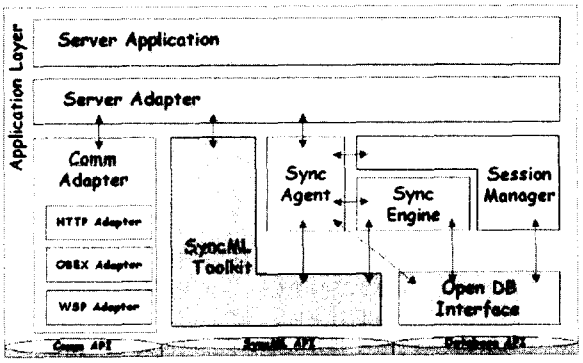

그림 1. SyncML 서버 프레임 구조

4. Sync Engine

Sync Engine은 SyncML 프로토콜에 영향을 받지 않으면서 서비스에 종속적이고, 데이터간의 충돌이 발생하였을 경우를 처리하고, 서버의 정책에 따라 변경될 수 있는 부분을 구현한 프레임이다[6].

4.1 Sync Engine의 설계 및 구현

Sync Engine은 크게 Service Dependent Handler, Conflict Resolution, Authentication Handler, Utility 등 4개의 모듈로 설계하였다. 그림 2는 Sync Engine 프레임의 세부 구성이다.

① Service Dependent Handler

SyncML은 일정관리, 주소록 이외에도 Email, 메시지, 텍스트 등 여러 가지 서비스를 적용할 수 있다. 이미 구현

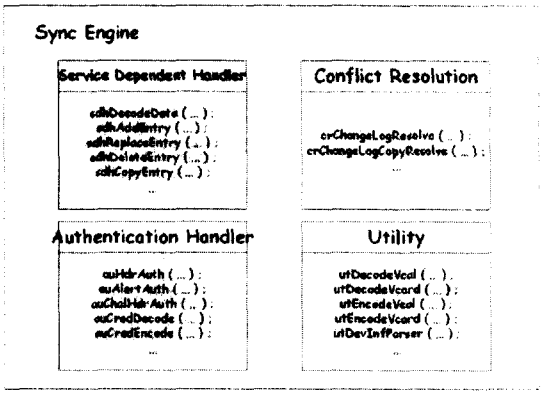

그림 2. Sync Engine 세부 블럭도

된 SyncML 서버에 이러한 서비스를 추가하려 할 때, SyncML 프로토콜에 관련된 부분까지 수정을 한다면 매우 많은 부분을 새로 구현하거나 변경해야 한다. 이렇게 서비스에 종속적인 부분을 따로 모듈로 구현함으로써 서비스의 확장성을 유지 할 수 있고, 적은 비용으로 빠르게 서비스를 적용할 수 있도록 하였다.

② Conflict Resolution

Sync Engine에는 디바이스로부터 온 데이터가 서버에 있는 데이터와 충돌이 발생할 경우, 이를 해결하기 위한 모듈이 포함되어 있다. 충돌에 대한 정책은 서비스의 성격, 사용하는 용도에 따라 많이 좌우가 되는데 이렇게 정책에 따라 결정될 수 있는 부분을 쉽게 변경할 수 있도록 모듈로 구현하였다.

충돌은 디바이스간 데이터의 시간적 충돌과 명령 충돌로 나뉠 수가 있다. 디바이스간 데이터의 시간적 충돌은 한 유저에 해당하는 복수개의 디바이스가 시간적 차이를 두어 서버의 같은 데이터를 변경할 경우 발생하며, 가장 최근에 서버에 요구한 데이터를 서버는 채택한다. 데이터의 명령 충돌은 한 유저에 해당하는 여러 디바이스들이 하나의 데이터에 다른 명령을 요구하였을 때 발생하게 되는데, 예를 들어 한 유저에 속한 두개의 디바이스 1, 2가 있을 경우, 디바이스1 이 데이터 A를 삭제하라는 명령을 서버에 요구했다면, 서버에서는 A가 삭제되고, 디바이스 2에게 A가 삭제되었다는 명령을 보낼 준비를 한다. 디바이스 2는 서버에 동기화를 요구하기 전까지 데이터 A를 가지고 있으며 또 서버가 데이터 A를 가지고 있는 것으로 알고 있다. 이 때, 디바이스 2가 데이터 A를 데이터 B로 변경하라는 명령을 서버에 요구하였을 경우, 서버는 데이터 A를 삭제하라는 명령을 디바이스 2에게 보내는 대신, 서버에 데이터 B를 추가하고, 디바이스 1에게 데이터 B를 추가하라는 명령을 보낼 준비를 함으로써

데이터 명령의 충돌을 방지한다.

③ Authentication Handler

SyncML은 정식 사용자인지 확인하기 위해 인증 절차가 필요하다. SyncML이 지원하고 있는 인증 방식은 Basic과 MD5 인데, 인증 방식의 정책 또한 서버의 정책사항이 된다. 이를 위해 인증에 관련된 부분을 모듈로 구현하였다.

④ Utility

디바이스들과 서버와의 동기화를 위해서는 메시지 안의 실제 데이터 부분의 디코딩과 인코딩 작업이 필요하다. 서비스가 확장되면서 이러한 데이터의 디코딩, 인코딩 과정도 서비스에 맞게 확장되어야 한다. 또, 그 밖에 여러 가지 서비스에 종속적인 부분이 있을 수 있는데 이를 Utility 모듈에 구현하였다.

그림 3은 우리가 구현한 SyncML 서버를 이용하여 주고 받은 메시지 중의 일부분이다. 디바이스로부터 요청된 일정관리의 수정사항 처리와 서버쪽에서 다른 디바이스 또는 서버에 의해 변경된 내용을 클라이언트로 요청하는 메시지를 생성한 결과이다 [3,4].

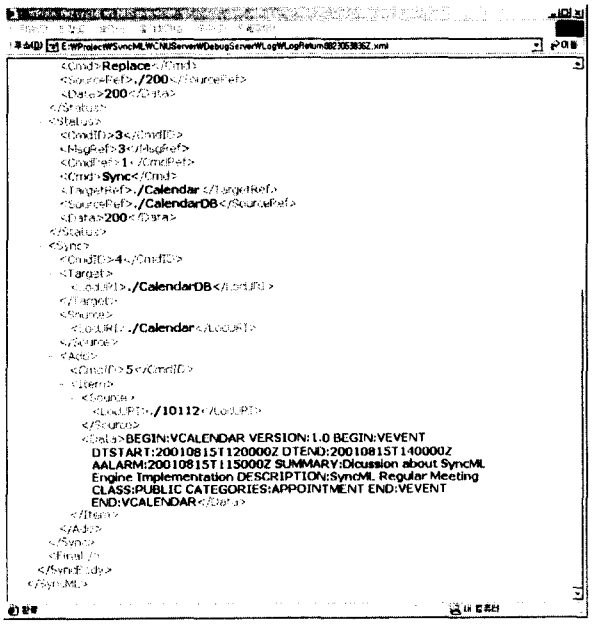

그림 3. 서버로부터 디바이스로 전송된 메시지 (일부)

5. 개발 환경 및 구성

SyncML 서버 개발 환경은, 운영체제는 윈도우즈 2000, 데이터베이스는 오라클 8i를 사용하여 많은 데이터를 안정성 있게 관리할 수 있도록 하였고 , 개발 언어는 Visual C++ 6.0을 사용하였으며, 각 프레임들은 6개의 DLL로 구성하였다. 디바이스로부터 HTTP를 이용하여 메시지를 주고 받기 위해 아파치 웹 서버 1.3.12를 사용하였고, 아파치 웹 서버가 전달 받은 메시지를 Tomcat 3.0을 이용하여 Java Class파일이 받고, 이를 Sync Agent로 전달하기 위하여 JNI를 이용하였다.[9]

6. 결론

다양한 회사들의 디바이스간 데이터 동기화의 문제점을 보완하기 위해 SyncML이라는 표준이 만들어지게 되었고, 이를 이용한 SyncML을 이용한 SyncML 서버를 구현하였다. 본 논문에서는 SyncML에 대한 개략적인 소개와, 구현 서버의 프레임, 구현한 서버 중 Sync Engine에 대하여 기술하였다. 본 논문에서 제시한 동기화 엔진은 데이터 동기화의 표준이라는 SyncML 자체의 장점 뿐만 아니라 서비스에 종속적인 부분을 Sync Engine이라는 프레임으로 분리를 함으로써 서비스의 확장성에 빠르게 대처할 수 있으며, 데이터간 충돌 발생 등에 관한 정책 결정사항에 대하여 쉽게 수정을 할 수 있다. 본 논문이 제시한 서버 및 Sync Engine 프레임은 사용자에게는 어느 디바이스, 어느 서버든 쉽게 사용할 수 있고, 개발자에게는 쉽고 빠른 서비스의 확장 및 정책에 대한 반영을 할 수 있다는 장점을 가지고 있다.

향후 연구 과제로는 서비스의 추가등에 대해 코드를 직접 수정하지 않고 자동으로 추가할 수 있는 엔진 라이브러리를 만들어 빠르고 쉽게 엔진을 확장하게 하는 방법을 고려할 수 있다.

8. 참고 문헌

[1] SyncML Initiative, http://www.syncml.org

[2] SyncML Initiative, Building an Industry-Wide Mobile Data Synchronization Protocol, SyncML White Paper, Mar. 30, 2000.

[3] SyncML Initiative, SyncML Sync Protocol, version 1.0.1, Jun. 15, 2001.

[4] SyncML Initiative, SyncML Representation Protocol, version 1.0.1, Jun. 15, 2001.

[5] SyncML Initiative, SyncML HTTP Binding, version 1.0.1, Jun. 15, 2001.

[6] SyncML Initiative, SyncML Architecture, version 0.2, May 10, 2000.

[7] 하인숙, 조재혁, 양지현, "데이터 동기화의 표준 SyncML 기초 다지기", 마이크로 소프트웨어 2001년 5월, pp.330-340, May 2001.

[8] 하인숙, 조재혁, 양지현, "SyncML 레퍼런스 툴킷 그 내부를 보자", 마이크로 소프트웨어 2001년 7월, pp.324-336, Jul. 2001.

[9] SyncML Initiative, SDA2 Specification, version 0.2. Aug. 21, 2000.