2001 년 한국정보처리학회 추계 학술발표논문집 제 8 권 제 2 호

SyncML 프로토콜을 이용한 데이터 동기화 서버 Agent 설계 및 구현

*류수희, *최훈, **류시원
*충남대학교 컴퓨터공학과
**㈜ 네오스텝스
e-mail : {shryoo,hchoi}@ce.cnu.ac.kr, wonrs@neosteps.com

Design and Implementation of the SyncML Server Agent

*SooHee Ryoo, *Hoon Choi, **ShiWon Ryu
*Dept. of Computer Engineering, Chungnam National University,
**Neostpes, Inc.

요 약

무선 이동 통신의 발달로 인해, 각 이동 단말 장치에 따른 데이터 동기화 방법이 다양해짐에 따라 공통된 동기화 프로토콜 규격의 필요성이 대두되었다. SyncML 은 이기종 시스템 간의 상호 운용성을 보장하는 데이터 동기화 표준으로서 사용자와 벤더들에게 많은 장점을 제공하는 프로토콜로 인식되고 있다. 본 논문에서는 SyncML 프로토콜에 대한 간략한 설명과 함께 SyncML 을 이용하여 개발된 서버의 프레임웍을 제시하고, 데이터 동기화를 처리해 주는 Sync Agent 에 대하여 기술하였다.

1. 서론

무선 인터넷이 발전함에 따라 이동 데이터 단말기를 통해 인터넷(Internet)을 검색하거나, 정보 제공자(ISP, Internet Service Provider) 서버에 액세스하여 데이터 서비스를 이용하는 것이 보편화되어 가고 있다. 이로 인해, 개인 정보 관리자(PIMS : Personal Information Management Service) 소프트웨어를 이용하여 자신의 업무 일정을 관리하거나 전자 메일을 교환하는 등 이동 데이터 단말기 이용이 개인적인 용도 뿐만 아니라 비즈니스의 새로운 수단으로 발전하고 있다. 또한 무선 이동 통신 환경이 다양해지면서 각 벤더마다 제공하고 있는 이동 단말 장치의 종류가 다양해지고, 개별 유저당 보유하고 있는 단말의 수도 늘어나고 있다. 이때, 여러 단말 장치에 분산되어 있는 동일한 데이터에 대해서 서로 동기화(Synchronization)가 이루어져야 하는데, 이를 위해 사용되는 동기화 프로토콜은 현재 단말기 벤더마다 각기 독자적인 방법을 개발하여 사용하고 있다. 따라서 이기종 시스템 간, 서비스 간 데이터 호환이 불가능하다.

호환성 없는 동기화 프로토콜이 점점 늘어나면서 데이터 동기화 표준의 필요성이 대두되었고, 이에 2000 년 2 월 IBM, Lotus, Motorola, Nokia, Palm, Psion, Starfish Software 등의 모바일 관련 업체를 중심으로 SyncML(Synchronization Markup Language) 컨소시엄이 구성되었다. SyncML 그룹은 서로 다른 컴퓨터 플랫폼, 네트웍, 응용서비스에 이용될 수 있는 데이터 동기 방식의 개방형 표준 개발을 목적으로 한다. 2000 년 12 월 SyncML 스펙 1.0 이어 현재 스펙 1.0.1 까지 발표하였다.

본 연구를 통해서 SyncML 프로토콜을 적용한 데이터 동기화 서버를 설계하고 구현하였다. 논문 구성은 2 장에서 SyncML 에 대한 간략한 소개를, 3 장에서 SyncML 을 적용하여 구현한 서버 프레임웍을 제시하였다. 제시한 동기화 서버 프레임 구조에서 SyncML 프로토콜에 따라 데이터 동기화 작업을 처리하는 Sync Agent 에 대한 세부 구현 사항을 4 장에서 기술하고 5 장에서 본 논문에 대한 결론을 마무리 지었다.

2. SyncML 에 대한 개요

데이터 동기화의 목적은 분산되어 있는 여러 유무선 단말 내의 공통된 데이터들을 서로 일치(Synchronization) 시키도록 하는 데에 있다. SyncML 은 임의의 네트웍 환경 하에서, 서로 다른 단말기와 서비스의 종류에 구애받지 않고 데이터 동기화를 수행하는 표준 언어이다.

SyncML 은 표준 데이터 동기화를 위해 크게 메시지의 데이터 포맷을 정의한 표현(Representation) 프로토콜, 동기화 규칙을 정의한 동기(Synchronization) 프로토콜, 전송 프로토콜의 바인딩을 정의하고 있다.

SyncML 이 메시지를 전송하기 위해 정의하고 있는 전송 프로토콜로는 HTTP, WSP(Wireless Session Protocol), OBEX(OBject EXchange Protocol) 등인데, 데이터 표현 프로토콜과 동기 프로토콜이 전송 프로토콜에 독립적이므로 향후 다른 전송 프로토콜과의 바인딩이 가능하다[7].

SyncML 데이터 표현 프로토콜은 데이터, 메타 데이터, 동기화 명령어 등과 같이 동기화를 수행하기 위해서 필요한 모든 표현 양식을 표시할 수 있는 XML(Extensible Markup Language)의 DTD(Document Type Definition)를 정의하고 있는 SyncML 메시지에 대한 구조체 규약이다. 데이터 표현 프로토콜로 작성된 메시지는 일반 텍스트 형태의 XML 양식이나 WAP 과 같은 프로토콜을 이용하는 초소형 장치를 위해 재정된 바이너리 형태의 WBXML(Wireless Binary XML) 양식을 따른다. 또한 SyncML 메시지는 인터넷의 다목적 메시지 전송을 위해 정의된 MIME 타입으로 정의되어 있어 MIME(Multi-purpose Internet Mail Extensions) 타입을 지원하는 응용 프로그램이나 트랜스포트 층에서 쉽게 전송될 수 있다[5,8].

SyncML 동기화 프로토콜은 SyncML 서버와 클라이언트 간에 데이터를 추가, 삭제, 변경하기 위해서 SyncML 메시지가 교환되는 방법과 실제 동기화를 위해서 동작하는 방법 및 동기화 타입을 정의한다. 동기화는 클라이언트가 먼저 SyncML 메시지를 서버에게 전송하고, 서버는 서버에 저장되어 있는 데이터와 클라이언트가 전송한 SyncML 메시지 내의 데이터 동기화를 수행하고 응답 메시지를 보내는 일련의 과정을 통해서 이루어진다. 그림 1 은 클라이언트와 서버 간의 기본 역할을 도시하였다[1,2].

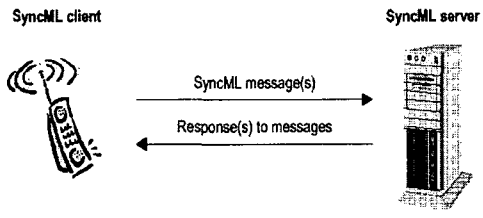


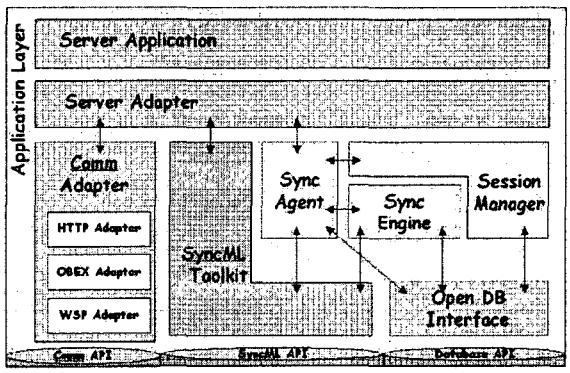
그림 1. SyncML 동작 절차

3. 구현한 SyncML 서버 구조

3.1 서버 환경

SyncML 서버를 위해서 설정되어 있어야 하는 시스템 환경으로 아파치 웹 서버 1.3.12, 서블릿 구동을 위한 톰캣, 데이터 저장을 위한 데이터베이스로 오라클 8i 를 사용하였다. 구현 환경 하에서의 클라이언트 서버 동작은 다음과 같은 순서로 이루어지고 있다.

SyncML 클라이언트가 서버에게 XML 포맷의 Request 메시지를 HTTP 를 통하여 전송하고, 이를 아파치 서버가 받아서 서블릿 엔진에게 포워드 하면 서블릿이 실제 서버의 동기화를 구현한 DLL 모듈을 호출함으로써 동기화를 수행한다. 이 때, 자바 서블릿과 원시코드 모듈(DLLs)을 호출하게 하는 방법으로써 JNI(Java Native Interface)가 사용되었다[3,4]. 실제 동기화를 수행하는 SyncAgent 와 SyncEngine 등의 원시 코드 모듈은 C++로 작성한 6 개의 DLL 들로 이루어져 있다. 3.2 절에서 서버 프레임웍을 정의하고 각 모듈 간의 관계와 기능을 살펴보겠다.



3.2 서버 프레임웍

그림 2. SyncML 서버 프레임웍

위의 그림 2 는 구현한 SyncML 서버의 전체 프레임웍을 정의한 것이다. 가장 상위 레벨에 정의되어 있는 Server Application 은 서버쪽에서 데이터를 변경하거나 서버 내의 정보를 검색하는 등의 작업을 용이하게 하기 위한 GUI 로서 실제 클라이언트와의 데이터 동기화를 수행하는 Sync Agent 및 Engine 등의 모듈과는 독립적이다. 서블릿이 클라이언트로부터 받은 Request 메시지를 JNI 를 통해 서버 Agent 내에 넘겨주게 되는데, 이 때 DLL 과 서블릿 간의 인터페이스 역할을 해주는 모듈이 Server Adapter 이다. Server Adapter 와 직접적으로 인터페이싱하는 모듈로는 SyncML Toolkit 과 Sync Agent 가 있다. SyncML Toolkit 에서는 Sync Adapter 로부터 전달받은 XML 포맷의 메시지를 SycML 에서 정의한 DTD 에 맞추어 디코딩하고, 서버에서 보내는 응답 메시지를 인코딩 해 주는 기능을 담당한다. 즉, SyncML 메시지의 각 엘리먼트를 추출하고 생성할 수 있는 SyncML 내부적으로 정의한 파서 기능이 구현되어 있는 것이다. Sync Agent 는 SyncML 프로토콜에 따라서 메시지를 처리하고

Sync Engine 과 데이터베이스를 컨트롤하는 모듈로서, 제공되는 서비스의 종류에 영향 받지 않고 수행되는 모듈로 설계되었다. 그리고 나머지 구조로는 서비스에 종속적이고 동기화 내용을 데이터베이스에 반영하고 Conflict 발생시 중재하는 등의 역할을 담당하는 Sync Engine, 서버 내에 데이터를 저장하기 위해 데이터베이스와 인터페이스 하는 Open DB Interface 모듈, 현재 동기화를 진행 중인 세션에 대한 정보를 관리하고 있는 Session Manager 가 있다.

4. Sync Agent 설계 및 구현

Sync Agent 는 새로운 응용 서비스가 추가되거나 삭제된다고 할지라도 Sync Agent 내의 구현 시나리오가 변경될 필요가 없도록 서비스에 독립적인 프로토콜만을 구현한 모듈이다. Sync Agent 내의 설계 내용 및 세부 구현 시나리오는 4.1 과 4.2 절을 통해서 기술하였다.

4.1 Sync Agent 구현 시나리오

SyncML 메시지는 SyncHdr 태그로 기술된 헤더와 SyncBody 태그로 기술된 바디로 구성되어 있는데, SyncML 헤더에는 SyncML 메시지가 전송될 타겟(Target) 장치를 찾을 수 있는 라우팅 정보 및 인증관련 정보를 담고 있으며, SyncML 바디는 동기화를 하고자 하는 데이터에 대한 정보를 담고 있는 하나 또는 그 이상의 SyncML 커맨드(Command)들로 이루어진다[9].

Sync Agent 는 동기화를 위해 *Toolkit* 이 제공하는 내부 파서를 통하여 디코딩된 메시지를 생성하는 기능을 담당하는데, 데이터 처리단위는 추가(Add), 변경(Replace), 삭제(Delete) 등을 지시하는 프로토콜 커맨드 단위로 수행되고 있다. 현재 SyncML 동기화 프로토콜에서 정의하고 있는 커맨드 중에서 Atomic, Search 를 제외한 Alert, Sequence, Sync, Add, Replace, Delete 등 SyncML 그룹에서 필수 항목으로 처리되어야 하는 모든 프로토콜 커맨드를 지원하고 있다. 들어온 메시지를 처리하는 방식으로 크게 두 가지로 분류할 수 있는데, 하나는 메시지가 완전히 파싱된 후, 동기화 실행 프로세서에 넘겨 처리하는 컴파일러 방식이 있고, 또 하나는 커맨드 단위로 바로 처리해 버리는 인터프리터 방식이 있다. 본 연구를 통해 구현한 Sync Agent 는 클라이언트로부터 받은 요청 메시지에 대해서 툴킷이 커맨드를 파싱하면 다음 커맨드를 파싱하기 전에 파싱된 커맨드를 실행 프로세서에 넘기는 인터프리터 방식으로 동작한다.

4.2 Sync Agent 세부 구성도

Sync Agent 는 크게 기능별로 Command Handler, Command Builder, Message Handler 등 3 개의 클래스 단위의 모듈로 구성하였다. Command Handler 는 들어온 메시지를 처리하는 모듈이고, Command Builder 에서는 보낼 메시지를 생성한다. Message Handler 는 들어온 메시지를 처리하기 위해 자주 접근되는 정보들을 관리하는 모듈이다. 그림 3 에서는 Agent 의 구성도를 도시하고 있고, 그림 4 에서는 Sync Agent 와 다른 모듈들과의 동작의 일부를 보여주고 있다.

1) Command Handler

Command Handler 는 Toolkit 이 파싱한 메시지를 커맨드 단위로 처리하기 위해서 SyncEngine 이나 Open DB Interfac, Session Manager 의 DLL 모듈간의 관계를 설정해주고 동기화 수행을 위해 해당 모듈을 호출하는 컨트롤 역할을 담당한다. 실제적으로 동기화를 수행하는데 있어서의 동작 흐름을 제어하고 있는 곳이 Command Handler 내에서 이루어지고 있다. 즉, 들어온 메시지를 처리하고 보낼 메시지를 위한 준비를 하는 기능을 수행함으로써, 나머지 다른 모듈 간의 인터페이스 역할을 담당하고 있다. 메시지 처리가 커맨드 단위로 이루어지고 있기 때문에, Command Handler 를 각 프로토콜 커맨드별로 핸들링할 수 있도록 세분화하여 구현하였다. 들어온 메시지를 처리하고 보낼 메시지를 위한 준비를 하는 기능을 수행함으로써, 나머지 다른 모듈 간의 인터페이스 역할을 담당하고 있다.

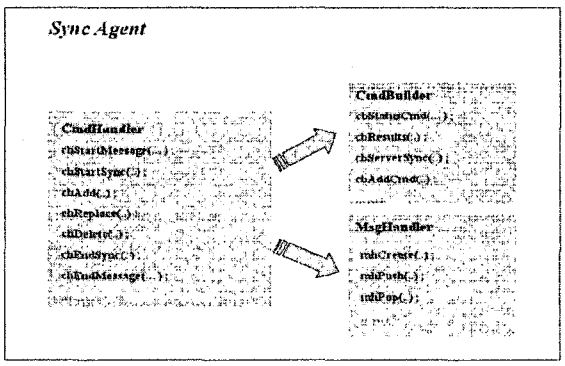


그림 3. Sync Agent 내부 구성도

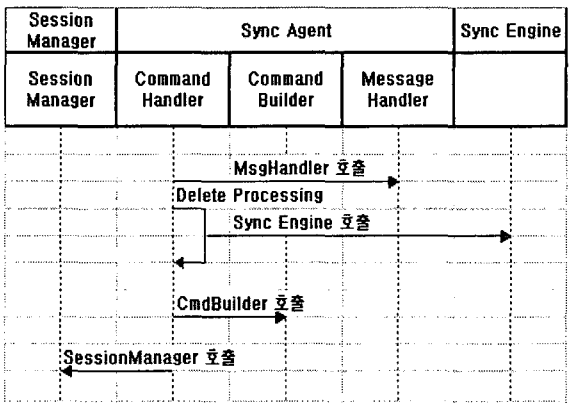


그림 4. Sync Agent 흐름도

2) Command Builder

Command Builder 에서는 클라이언트 메시지에 대한 응답 메시지를 생성하거나 서버 내에서의 변경 사항이 있을 경우, 클라이언트에게 변경 내용을 반영시키

기 위한 커맨드를 생성하도록 Toolkit 의 커맨드 생성 모듈과 인터페이스하는 역할을 담당하고 있다. Command Handler 에 의해서 제어되고 있지만, 내부적으로는 메시지를 생성하기 위해서 Toolkit 의 커맨드를 생성하는 모듈을 핸들링 하는 것 외에도, 서버쪽 변경 사항 등의 정보를 커맨드 내용으로 설정하기 위해서 Open DB Interface 와도 인터페이싱하고 있다.

3) Message Handler

Message Handler 를 통해서 클라이언트로부터 들어온 메시지를 처리하는 데 있어 빈번하게 사용되는 Global 한 정보들을 전역 구조체로 정의하여 메모리에 두고 리스트로 관리하고 있다. 이 때 Global 하게 정의해 두고 사용하는 정보로 현재 세션 내에 유효한 세션 관련 정보와 Add, Replace, Delete 등의 개별 커맨드를 담고 있는 Sync, Atomic, Sequence 등의 Parent Command 들을 스택으로 관리하고 있는 구조를 포함하고 있다. Message Handler 에서 관리하는 정보들은 동기화를 수행하는데 있어 세션 전체에 유효한 정보를 유지하는 것이 아니라 현재 Request 메시지를 처리하는 동안만 유효한 정보이다. 메시지가 들어오면 Command Handler 의 SyncML 메시지의 헤더를 처리하는 루틴 내에서 Message Handler 에 대한 구조체를 초기화 하고, SyncBody 의 내용을 커맨드 단위로 메시지를 파싱하면서 생기는 추가적인 정보가 생기면 리스트로 연결하여 관리, 사용한다.

5. 결 론

무선 이동 통신이 발전함에 따라 다양한 이동 단말 장치에게 공통적으로 적용될 수 있는 데이터 동기화 기술의 표준에 대한 중요성이 부각되었고, 이로 인해서 SyncML 이 탄생되었다. 모든 서로 다른 장치 혹은 서로 다른 시스템 환경에 구애받지 않고 동기화를 수행할 수 있도록 제안된 SyncML 프로토콜은 현재 600 개 이상의 스폰서들의 지원 하에 있고, 국내에서도 SyncML 프로토콜을 이용하여 데이터 동기화 솔루션을 개발하고 있는 업체가 20 군데 이상이 되고 있는 등 연구가 활성화되고 있다.

본 논문에서는 데이터 동기화를 위해서 표준 데이터 동기화 기술의 필요성과 함께, 표준 동기화 기술인 SyncML 에 대해 소개 하였다. 또한 개발한 SyncML 서버의 전체 프레임 웍을 제시하고 SyncML 서버 프레임 내에서 Sync Agent 에서 적용한 구현 상황에 대해서 살펴보았다. 본 논문에서 제시한 SyncML 서버 프레임 구조는 각기 기능별로 모듈화하여 라이브러리화 하여 사용함으로써, 확장성과 이식성이 높아 불필요한 코드의 낭비를 줄일 수 있다. 특히 Sync Agent 는 서비스에 종속적인 Sync Engine 과는 달리 응용 서비스에 종속적이지 않으면서 데이터 동기화를 수행할 수 있도록 설계된 모듈이므로, 개발자가 새로운 서비스를 추가한다거나 업그레이드가 한다고 하더라도 전체 모듈을 수정하는 오버헤드를 없앴다.

향후 과제로는 SyncML 에서 정의하고 있는 동기화 프로토콜 커맨드에 대한 기능 및 전송 프로토콜 바인딩에 대해서 확장 지원하고, 지원되는 MIME 타입 서비스를 확장해 나가는 것이 필요하겠다. 그리고 현재 SyncML 서버에서 제시하는 보안 메커니즘에 더 나은 안정성을 제공하기 위해 해당 메시지를 해독할 수 있는 키를 이용하는 등의 방식을 추가적으로 제안하고 아울러 사용자 인증 뿐 아니라 메시지 자체에 대한 보안을 보장할 수 있도록 보안 알고리즘을 강화하는 방안을 고려할 수 있겠다. 더 나아가 현재 윈도우 기반의 C++로 개발되어있는 현재 서버를 향후 플랫폼에 독립적이고 안정적인 자바 기반의 개발 환경으로의 개선을 생각해 볼 수 있겠다.

참고문헌

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

[2] SyncML Initiative, Building an Industry-Wide MobileData Synchronization Protocol, SyncML White Paper, Mar. 20, 2000

[3] SyncML Initiative, SyncML Architecture Version 0.2, May 10, 2000

[4] SyncML Initiative, SDA2 Specification Version 0.2, Aug. 21,2000

[5] SyncML Initiative, SyncML Representation Protocol, version 1.0.1, June 15,2001

[6] SyncML Initiative, SyncML Sync Protocol, version 1.0.1, June 15,2001

[7] SyncML Initiative, SyncML HTTP Binding, version 1.0.1, June 15,2001

[8] Extensible Markup Language (XML) 1.0 Second Edition), http://www.w3.org/TR/REC-xml

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