SyncML 소개 및 응용

서동환* · 조병호** · 이길흥***

1. 서 론

두개 이상으로 분산된 데이터가 있다고 가정해 보자. 우리는 이 두 개 이상의 데이터가 항상 일치하길 원한다. 이 경우 먼저 고려할 수 있는 해결책은 분산된 데이터를 하나로 통합하는 것이다. 그러나 물리적으로 데이터를 통합하는 경우 도메인에 따라 몇 가지 문제점이 발생할 수 있다.

통합 데이터가 데이터 접근자로부터 원거리에 위치할 경우 접근자의 도메인 성능에 따라 원거리 데이터로 접근하지 못할 수 있다. 또한 모든 데이터 접근자가 통합된 하나의 데이터로 몰리는 과부하를 피할 수 없게 된다. 이런 문제가 있는 도메인의 경우 물리적으로 데이터를 통합하는 것 이외의 다른 해결책을 고민해야 한다. 특히 모바일 장치의 경우 연결을 계속해서 유지하는 상태로 데이터에 접근하게 하는 것은 좋은 방법이 아니다. 물리적인 데이터 통합이 어려운 도메인의 경우 하나의 데이터를 여러 곳에 분산시켜 관리하는 방법을 사용해야 한다. 분산 데이터를 통합 데이터와 같은 방식으로 운영하기 위해서는 분산 데이터 간 데이터 변경을 동기화할 수 있는 프로그램을 이용해야 한다. 이것이 논리적 의미의 데이터 통합이다.

* (주)지어소프트 시스템 3팀 과장
** 서울산업대학교 공과대학 컴퓨터공학과 교수
*** 서울산업대학교 컴퓨터공학과 조교수

분산 데이터를 통합하기 위해 사용할 프로그램을 작성하려면 공통으로 고려해야 할 것이 있다. 동기화 프로그램의 기초가 되는 메시지 프로토콜의 구조와 데이터 타입에 대한 설계가 있어야 한다. 또한 해당 메시지 실행과 오류를 정밀하게 구현할 수 있어야 한다. 마지막으로 여러 응용 프로그램에서 사용할 수 있는 동기화 프로그램으로 만들기 위해 특정 응용 프로그램의 종속성을 제거한 공통적인 동기화 프로세서를 설계해야 한다. 즉 응용 프로그램과 동기화 프로그램을 분리하고 둘 간의 접합점 역할을 할 수 있는 인터페이스가 필요한 것이다. 개발자 각자가 설계한 데이터 동기화 프로그램의 설계서나 로직 프로토타입을 보면 동일한 설계와 로직은 없을 것이다. 이렇게 만들어진 프로그램은 다른 사람의 동기화 프로그램과 같이 동작하거나 연동되지 않는다. 메시지와 처리 규칙을 서로 이해하지 못하기 때문이다. 이 경우, 결국은 호환성이 부족한 프로그램이 양산될 것이다.

동기화 프로그램은 해당 도메인 문제에 밀접하게 연관돼 있을 것이다. 도메인 문제를 해결하는 다수의 응용 프로그램이 있을 경우 동기화 프로그램을 응용 프로그램 수만큼 작성해야 한다. 다시 말하면 무수히 많은 동기화 프로그램을 만들어내야 하는 것이다(n개의 응용프로그램이 있다면 최

대 (n×n-1))/2개의 동기화 프로그램이 필요하다). 문제는 여기에서 끝나지 않는다. 우리는 문제 도메인의 사용디바이스, 운영체제, 개발언어, 네트워크 프로토콜의 종류에 따라 도메인 환경에 종속적인 설계와 코딩을 하고 있을 것이다.

도메인 환경의 작은 변화에도 우리가 작성한 동기화 프로그램은 그 변화를 견뎌내지 못할지도 모른다. 이러한 문제점을 극복할 수 있는 가장 좋은 해결방법은 다양한 문제 도메인에 대한 경험이 있는 사람이 모여 각자가 동기화 문제를 해결하면서 얻은 지식과 경험을 나눠, 모두에게 통용될 수 있는 메시지 표준과 처리 규칙을 표준화하는 것이다.

2. SyncML의 탄생

SyncML(Synchronization Markup Language)은 현재의 데이터 동기화 프로그램의 문제인 호환성 부재, 환경에 종속적인 운영 문제를 해결하기 위해 만들어졌다.

SyncML 컨소시엄의 스폰서들을 살펴보면 대부분 모바일 계열의 유명한 업체가 많다는 것을 알 수 있다. 실제로 데이터 동기화가 이슈로 떠오르는 것은 모바일 장치 사용자의 수가 일정 수준을 돌파하면서부터라고 할 수 있다. 또한 기업 환경에서 기동성이 뛰어난 모바일 장치가 시스템의 한 축으로 사용되면서 PC와 모바일 장치간의 데이터 동기화가 핵심 기능으로 떠오른 것이다.

각 모바일 장치의 운영체제는 SyncML 이전부터 이미 운영체제 레벨에 탑재한 자사의 동기화 스펙이 있었다. 팜(Palm) 계열의 핫싱크(Hot Sync)나 윈도우 CE 계열의 액티브싱크(Active Sync)가 바로 이런 벤더스펙(Vender Spec)을 구현한 제품이다. 또한 각 통신 서비스 사를 중심으로 개인정보관리(PIMS)에 동기화 프로그램이 활발히 사용되고 있다. 모바일 사용자가 증가하면서 호환성이 없는 벤더 스펙의 동기화 프로그램이 점점 늘어나자 모바일 관련 업체를 중심으로 통일적인 동기화 표준을 작성하기 위해 SyncML 컨소시엄이 구성됐다.

자사의 폐쇄적인 스펙을 사용하던 개발사들은 상호호환성 보장을 위한 표준을 도출했고, 그 결과 2000년 12월 SyncML 1.0 스펙이 탄생했다. 모바일 환경에 대한 고려가 스펙에서 눈에 자주 띄는 것은 모바일 관련 업체가 SyncML 컨소시엄의 구성과 스펙 작성을 주도한 것과 무관하지 않다. 하지만 SyncML은 단순히 모바일 기기와 PC간의 데이터 동기화에 사용이 국한된 것은 물론 아니다. PC와 PC, PC와 메인프레임간의 데이터 동기화를 포함한 모든 분산 데이터간의 동기화에 SyncML을 이용할 수 있다[1].

3. SyncML 스펙

데이터 동기화 표준을 위해서는 데이터 동기화에 필요한 정보를 담는 메시지 구조체, 메시지의 처리에 필요한 프로토콜, 구성된 메시지의 네트워크 송수신을 위한 트랜스포트 바인딩에 대한 표준이 있어야 한다. SyncML은 이를 위해 전체적으로 네 가지 카테고리로 구성되어 있다.

3.1 XML (eXtensible Markup Language) 표현

XML 표현 카테고리는 SyncML 메시지 구조체를 정의한 스펙이다. 각각의 필드가 어떠한 정보를 담고 있으며 해당 정보가 어떤 의미를 내포한 것인지에 대한 약속들을 정의하고 있다. SyncML은 메시지 구조체의 정의를 위해 세 가지 스펙을 제시하고 있다. SyncML 표현 프로토콜(Representation Protocol) 스펙은 SyncML의

XML 기본 표현을 기술한 핵심 스펙이다. 디바이스 정보 표현을 위한 SyncML Device Information DTD 스펙과 추가적인 동기화 정보 및 해석을 표현하기 위한 SyncML Meta Information DTD 스펙은 SyncML 표현 프로토콜에 캡슐화되어 사용된다. 그림 1은 크게 헤더(Header)와 몸체Body)로 구성된 SyncML 메시지를 보여 주고 있다[2].

3.2 싱크 룰 (Sync Rule)

싱크 룰 카테고리는 XML 표현 표준을 이용하여 동기화를 수행하는 일련의 프로세스 규칙들을 제시하고 있다. SyncML 싱크 프로토콜 스펙은 표 1과 같이 데이터 동기화를 수행하기 위해 필요한 각종 정보의 개념과 동기화 프로세스 등을 7가지 동기화 타입으로 나누고 각각에 대한 메시지 교환 규칙을 정의하고 있다[3].

3.3 트랜스포트 바인딩 (Transport Bindings)

SyncML은 네트워크 상에 존재하는 분산 데이터간의 데이터 동기화에 관련된 스펙이다. 트랜스포트 바인딩 카테고리는 여러 네트워크 프로토콜을 이용해서 메시지를 전송할 때 필요한 요구사항을 표준화 하고 있다. 현재 SyncML은 HTTP, WSP, OBEX에 대한 스펙을 가지고 있다. 향후 이메일, 메시지큐에 대한 바인딩이 정의될 예정이며, 이외에도 여러 종류의 트랜스포트 프로토콜이 바인딩 될 것이다. 그림 2는 크게 3가지 HTTP, WSP, OBEX 구성된 트랜스포트 바인딩 구조를 보여 주고 있다[4].

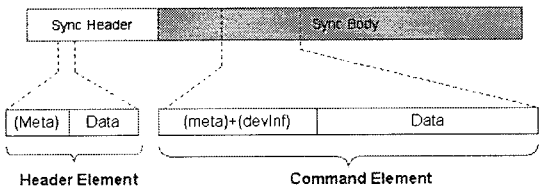


그림 1. SyncML 메시지 구성

표 1. SyncML 명령어 종류

종류	역할	명령어
Request 명령어	데이터 동기화	Add, Copy, Delete, Exec, Map, Replace, Search
	초기화	Alert
	디바이스 정보	Get, Put
	컨테이너	Atomic, Sequence, Sync
Response 명령어		Status, Results

3.4 상호운용성 스펙 (Interoperability Specification)

SyncML은 기존의 데이터 동기화 프로그램의 한계를 극복하고자 제시된 스펙이다. 따라서 상호운용성은 SyncML의 존재 이유가 되고 있다. 앞서 세 개의 카테고리가 실제 SyncML 엔진을 구현하기 위해 필요한 스펙이라면 본 카테고리에 속하는 스펙들은 자신이 구현한 SyncML 엔진이 다른 벤더의 엔진과 호환되어 동작하는지를 테스트할 수 있는 절차와 주의사항을 명시하고 있다.

3.5 표현 프로토콜 스펙

SyncML 메시지에 대한 구조체 규약(동기화 포맷)을 정의한 것이 SyncML 표현 프로토콜 스펙이다. 표현 프로토콜 스펙은 내부의 기술 언어로 XML을 사용하고 있어 최종적으로 XML 문서

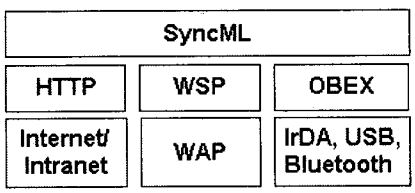


그림 2. 트랜스포트 바인딩 구조도

가 되어 네트워크로 전송된다. 메시지는 인터넷 상의 다목적 메시지 전송을 위해 정의된 MIME (Multipurpose Interface Mail Extensions) 타입으로 정의되어 있다. 따라서 SyncML 메시지는 MIME을 지원하는 어플리케이션이나 트랜스포트 층에서 쉽게 전송될 수 있다. 이와 같이 SyncML은 ebXML, VoiceXML, MathML등과 같이, 전문분야(전자상거래, 음성, 수학 등)에서 통일적으로 인식할 수 있는 포맷을 정의하기 위해 XML을 사용한 규약의 한 종류라고 할 수 있다. SyncML 스펙을 이해하거나 동기화 엔진을 구현하기 위해서는 전문적인 XML 지식을 반드시 갖추고 있어야 한다. 그림 3은 기본적인 SyncML 메시지 구조를 보여주고 있다[5].

3.6 SyncML 패키지와 메시지

SyncML 프로토콜 스펙에서 동기화의 기본이 되는 단위는 패키지이지만, 패키지의 크기가 해당 디바이스의 한계메모리를 넘어서는 경우가 있다. 대용량의 데이터를 동기화 할 경우 상대편 디바이스가 소량의 메모리로 동작하는 모바일 환경이라면 메모리 오버플로우가 발생할 수 있다. 따라서 패키지는 상대편 디바이스의 자유메모리 보다 작은 메시지로 분할되어 전송될 수 있어야 한다. 다중 메시지를 지원하기 위해 Final 엘리먼트를 사용하는데, 이를 이용하여 마지막 메시지인지를 판단한다. 여기서 잠깐 살펴볼 것은 세션 (session)이라는 개념이다. 세션은 하나의 완전한 데이터 동기화를 위해 클라이언트, 서버 간에 주고받은 모든 패키지를 묶어주는 단위의 개념으로 사용된다. 세션은 여러 패키지를 포함하고 하나의 패키지는 다시 다수의 메시지를 포함한다.

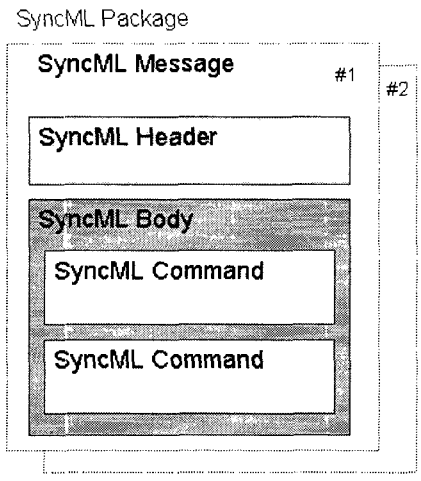


그림 3. SyncML 메시지 구조

3.7 SyncML 명령어

SyncML 메시지는 하나하나 개별적인 잘 정의된 (well-formed) XML 문서가 된다. 메시지는 SyncHdr 태그로 기술된 헤더와 SyncBody 태그로 기술된 몸체로 구성된다. SyncML 헤더는 SyncML 메시지가 전송될 타겟 디바이스를 찾을 수 있는 라우팅 정보를 가지고 있다. 라우팅 정보 이외에도 SyncML 버전에 대한 정보, 인증 절차에 필요한 정보를 담고 있다. SyncML 몸체는 하나 혹은 그 이상의 SyncML 명령어들을 담는 컨테이너 역할을 한다. 명령어는 표 2와 같이 역할별로 구분할 수 있다. 헤더가 동기화 대상이 위치하고 있는 디바이스에 대한 접근 정보를 담고 있다면 SyncML 몸체의 명령어는 동기화 대상이 되는 데이터를 식별하고 실제 동기화에 사용될 정보를 표현하고 있다. Atomic은 Atomic 안에 존재하는 명령어들은 어느 하나가 실제 동기화에 실패할 경우 기존에 동기화된 모든 것이 다시 복귀 (rollback)하는 트랜잭션의 단위 영역을 나타내 준다. Sequence는 Sequence 안에 포함된 명령어들을 순차적으로 처리하도록 요구한다. SyncML의 경우 각각의 명령어를 성공, 실패의 최소단위로 처리

표 2. SyncML 명령어 설명

Alert	동기화를 진행할 상대방과 최초로 교환하는 명령어. 양방향, 단방향 같은 동기화 방식 교환. Anchor 정보 교환. Anchor 정보는 이전에 상대방과 데이터 동기화의 일치성 여부를 확인하는 키워드가 된다. 만약 받은 Anchor 정보가 자신이 기억하는 Anchor 정보와 다르다면 어느 시점에서 동기화가 어긋난 것으로 판단 전체 데이터를 재 동기화 하는 작업을 진행한다.
Atomic	트랜잭션 처리 영역을 설정하는 명령어. Atomic은 Atomic에 포함된 명령어들 중 하나의 명령어라도 실패할 경우 이미 실행된 모든 명령어를 롤백 (rollback) 시킨다. SyncML의 경우 다수의 명령어가 SyncML 몸체에 포함될 수 있는데 몸체에 포함된 각각의 명령어가 하나의 성공/실패 단위가 된다. 따라서 Atomic을 이용해 성공/실패 수행 단위를 조절할 수 있다.
Sequence	Atomic이 트랜잭션 처리를 위한 명령어라면Sequence는 순차 처리를 위한 명령어이다. Atomic 안에 포함된 명령어들은 실행 순서와는 상관없이 롤백과 커밋(commit)에 신경을 쓰는 반면 Sequence는 포함된 명령어의 순차적인 처리에 초점을 두고 있다.
Sync	Add, Delete, Replace는 동기화 대상 자체에 대한 정보(테이블)가 아닌 동기화 대상 안에서 구분되는 일부단락(레코드)에 대한 타겟 정보만을 가지고 있다. 따라서 이들 명령어는 홀로 쓰이는 것이 아니라 반드시 Sync를 같이 사용해 동기화 대상에 대한 정보를 획득할 수 있어야 한다. Sync는 Atomic이나 Sequence 같은 특수한 역할은 없으며 단순히 동기화 대상을 지칭하는 역할을 수행한다.
Add	데이터의 추가를 요구하는 명령어.
Copy	데이터의 복사를 요구하는 명령어.
Delete	데이터의 삭제를 요구하는 명령어.
Replace	상대편 디바이스에 데이터가 존재하는 경우 수정 처리가 되며 데이터가 없는 경우 추가 처리를 요구하는 명령어.
Get	동기화 하려는 상대편 디바이스 정보를 자신에게 보내달라고 요청하는 명령어.
Put	자신의 디바이스 정보를 담고 있는 명령어. Put 명령어에 들어갈 정보는 SyncML Device Information DTD 스펙에 명시되어 있다. Put과 Get 명령어를 통해 동기화를 진행할 상대편 디바이스와 자신의 디바이스 정보를 교환하고 이런 정보를 내부에 저장해 메시지 분할이나 정보 포맷 형태의 변경과 같은 지능적인 동기화 처리 프로세스를 진행하게 할 수 있다.
Map	서버와 클라이언트가 서로 다른 데이터식별 ID (DB에서는 P-Key에 해당)를 사용하는 경우 서버 쪽은 맵테이블 (map table)을 통해 서로의 ID를 일치시켜야 한다. 그러나 서버가 클라이언트에 Add를 요청할 경우 서버는 아직 클라이언트가 데이터를 추가한 상태가 아니기 때문에 맵테이블에 정보를 넣을 수가 없다. 따라서 클라이언트는 서버가 요청한 Add를 실행한 후 반드시 서버 측에 자신의 ID와 서버에서 보내준 ID를 매칭시킨 Map 명령어를 보내줘야 한다. 서버와 클라이언트가 동일한 ID를 사용한 다면 Map 명령어는 사용하지 않는다.
Results	Search나 Get 명령어의 결과값을 반환하는데 사용하는 명령어.
Search	상대편 데이터베이스를 검색하는데 필요한 정보를 담고 있는 명령어.
Exec	상대편 디바이스의 특정 프로세스를 실행 시키는 명령어. 예를 들어 데이터를 변경시켜주는 로직이 담겨있는 실행 파일이 있을 경우 상대편은 동기화 데이터를 보내주는 것이 아니라 실행파일을 실행시켜 로컬 상에서 직접 데이터를 일치시키는 작업을 진행할 수 있다.
Status	요청한 명령어를 처리하고 그 결과를 반환할 때 사용하는 명령어. Status는 성공과 실패를 반환하는데 성공과 실패는 그 구분은 상당히 모호하다. 예를 들어 Delete 명령어를 실행하려는데 이미 해당 데이터가 삭제되어 있다면 Status 명령어에 성공과 실패의 의미를 동시 담고 있는 상태코드(status-number)를 담아 반환한다. 따라서 Status는 성공과 실패로 처리를 구분하지 말고 해당 명령어가 가질 수 있는 최대한의 반환 값을 예상하여 로직을 작성토록 해야 한다. 상태(Status)를 반환 할 수 있는 엘리먼트는 SyncML 헤더와 단위 명령어이다. Sync, Sequence 경우 소속된 각각의 명령어도 상태를 반환할 수 있다. 단 Atomic의 경우 소속된 명령어는 상태를 반환하지 않고 Atomic 자체가 상태를 반환한다. Status를 반환하는 것은 네트워크의 상태가 좋지 않거나 처리 능력이 떨어지는 디바이스의 경우 큰 부담이 될 수 있다. 따라서 헤더나 각 명령어에 NoResp를 사용하여 결과에 대한 반환여부를 지정할 수 있다. (물론 Status 명령어는 처리 결과를 반환하지 않는다.)

표 3. SyncML 엘리먼트

Common Use Elements	데이터 동기화에 필요한 각각의 정보를 제공하는 필드 구조체
	Archive, Chal, Cmd, Cred, Final, Lang, LocName, LocURI, MsgID, NoResp, NoResults, RespURI, SessionID, Source, Target 등
Message Container Elements	메시지를 이루는 구조체
	SyncML, SyncHdr, SyncBody
Data Description Elements	동기화할 데이터 자체를 담고 있는 구조체
	Data, Item, Meta
Protocol Management Elements	실행을 요청한 명령어의 성공/실패를 반환
	Status
Protocol 명령어 Element	SyncML 몸체에 들어가는 명령어들
	Add, Alert, Atomic, Copy, Delete, Exec, Get, Map, MapItem, Put, Replace, Results, Search, Sequence, Sync

한다. 때문에 사용자는 컨테이너 명령어들을 이용해 성공, 실패의 최소단위를 조절할 수 있다[3].

3.8 SyncML 엘리먼트(Element)

SyncML 헤더와 SyncML 몸체의 각 명령어들은 엘리먼트의 집합체이다. SyncML은 아래 표3과 같은 엘리먼트를 조합하여 데이터 동기화에 필요한 여러 정보를 전송한다[3].

4. XML 기반 메시지

SyncML에서 메시지의 기술언어로 XML을 사용한다. SyncML은 동기화가 이루어지는 클라이언트, 서버의 도메인 환경을 이기종 환경으로 가정하고 있다. 예를 들어 'A' 디바이스는 팜 운영체제 위에서 C++로 구현된 SyncML 엔진을 사용하고 'B' 디바이스는 윈도우나 리눅스 운영체제 위에서 Java로 구현된 SyncML 엔진을 사용한다고 가정해 보자. 메시지는 이러한 이기종 환경을 극복하고 'A'와 'B' 디바이스 모두에서 올바르게 인식될 수 있어야 한다. SyncML에서는 이기종 환경에서 메시지가 동일하게 인식되기 위한 특별한 규약이 없다. 때문에 SyncML은 메시지의 상호 운용성을 위해 XML을 사용하고 있다. 현재 산업계 근간에서 XML이 광범위하게 여러 스펙에 사용되고 있는 것은 XML이 이기종 환경에서 서로 간 상호운용성을 보장해 줄 수 있기 때문이다. XML은 모든 정보가 문자(char)로 표현되기 때문에 양쪽 다비이스가 서로 인식할 수 있는 문자코드를 사용하여 메시지를 캡슐화 한다면 양쪽 디바이스는 언제든 메시지를 동일하게 인식할 수 있다.

이기종 분산객체 환경을 엮어주는 코바 (CORBA : Common Object Request Broker Architecture)와 XML의 상호운용성을 비교하면 표 4와 같다.

표 4. 코바와 XML의 상호운용성 비교

분 류	XML	코 바
객체 호출	DOM, SAX	POA, BOA, ORB등 코바 인터페이스
메타데이터 규정	DTD, XML Schemas, XML Data	IDL
메시지 마샬/언마샬	코드셋을 이용한 XML 문서의 스트림 변환	CDR을 적용을 통한 스트림 변환

5. 파서의 구현

SyncML 엔진을 구현한 경우 내부적으로 파서가 들어가야 한다. 그러나 현재 대부분 상용 파서의 크기는 상당히 크다. 작은 용량의 메모리를 사용하는 모바일 디바이스에 탑재하기 위해서는 결과적으로 작은 크기의 파서를 직접 만들어야 한다. Clear-text 파서의 경우 일반적으로 컴파일러를 구현할 때 사용하는 스트링 토큰 방식으로 무리 없이 구현할 수 있다. WBXML (Wap Binary eXtensible Markup Language)의 경우는 Clear-text 보다 더 많은 주의점이 요구된다. 태그식별, 코드페이지, 코드페이지 변경을 알려주는 토글 값, 종료태그 모두 1바이트 값으로 표현되고 있기 때문에 이들을 인식하기 위해서는 보다 세분화된 로직이 필요하다. 일반 PC의 경우 사용메모리 제한이 모바일 디바이스보다 여유가 있으므로 파서 자체의 크기를 줄이기보다는 성능에 주안점을 두는 것이 좋다.

파서를 통해 파싱된 정보는 데이터 동기화 수행 프로세스에 넘겨져 어플리케이션과의 인터페이스를 통해 데이터 동기화를 수행한다. 앞서 말했듯이 SyncML 몸체는 다수의 명령어가 올 수 있다. 하나의 메시지가 완전히 파싱된 후, 실행 프로세스에 '명령어들'을 처리하도록 넘기는 컴파일러 방식은 미리 파싱을 끝낸 이후에 명령어들을 실행하는 방식으로 실행을 하기 전에 파싱에서 발생하는 오류를 미리 체크할 수 있는 장점이 있다. 그러나 단일 쓰레드로 파싱 작업과 실제 명령어를 실행하게 되므로 전체 실행시간을 어느 정도 희생하게 된다. 반면 명령어 하나를 파싱한 후 다음 명령어를 파싱하기 전에 파싱해 놓은 명령어를 실행 프로세스에 넘기는 인터프리터 방식은 파싱과 처리를 다중 쓰레드로 처리할 수 있기 때문에 전체처리 속도는 컴파일러 방식보다 빠르다. 하지만 이후 파싱 과정에서 발생하는 오류를 제어하기 힘들어 지는 단점을 가진다.

6. 패키지 교환 (동기화 프로토콜)

실제 하나의 데이터가 동기화 되려면 단순히 동기화 데이터의 교환으로 작업이 완료될 수 없다. 앞서 설명한 SyncML의 여러 명령어들은 보다 정확하고 정밀한 동기화 작업을 위해 사용될 것이다. SyncML 프로토콜 스펙에서는 모두 7가지 동기화 타입을 정의하고 있다. 여기서 가장 기본이 되는 동기화 타입은 단방향 싱크로서 나머지 동기화 타입은 양방향 싱크의 서브 셋 또는 용도에 따라 조금씩 변형을 가한 타입이라고 할 수 있다. 특이한 점은 양방향 싱크를 포함한 6개의 타입은 모두 클라이언트가 서버에게 동기화를 요청하게 되어 있으나 서버 Alerted 타입일 경우에는 서버가 동기화를 시작하는 주체가 된다. 서버가 동기화를 시작하기 위해서는 푸시(push) 모델을 사용한 전송이 반드시 필요하다[4].

6.1 패키지 교환 예

클라이언트와 서버에 분산 데이터를 동일하게 유지하고자 한다. 그림 4와 같이 클라이언트에는 CDB에 'C_24'를 키 값으로 "문상철"이라는 데이터가 저장되어 있다. 서버에는 SDB에 'S_24'라는 키로 "최상훈"이 저장되어 있고 'S_25'라는 키로 "박자영"이 저장되어 있다. 서버와 클라이언트의 데이터베이스는 키 값의 데이터 형이 다르다. 서로 다른 데이터식별ID인 키를 매칭시키기 위해 서버에서 맵 테이블을 관리하고 있다. 그림 4에서 맵 테이블을 살펴보면 클라이언트의 'C_24' 레코드와 서버의 'S_24' 레코드가 동일한 정보를 나타내고 있음을 알 수 있다. 만약 'C_24' 레코드의

정보가 변경이 되면 데이터 동기화 프로그램은 서버의 'S_24' 레코드에 클라이언트 변동 사항을 수정해 줘야 한다. 아직 클라이언트의 'C_24' 레코드와 서버의 'S_24' 레코드는 동기화 되지 않았다. 그리고 서버에 'S_25' 레코드는 새로 추가된 레코드로 이 서버에 정보 변경이 클라이언트로 동기화 되지 않았다.

그림 4의 초기 상태에 두 디바이스 간 동기화가 일어나면 클라이언트에 'C_24' 레코드의 정보가 서버의 'S_24' 레코드에 변경정보가 적용되어야 하고 서버에 'S_25' 레코드가 클라이언트의 데이터베이스에 추가되어야 한다. 동기화가 완료된 후의 서버와 클라이언트의 정보는 그림 5와 같이 변경된다.

클라이언트는 'C_24' 레코드의 "문상철"이 서버와 동기화 되면서 서버의 'S_24' 레코드의 "최상훈"이 "문상철"로 수정되었다. 서버의 'S_25' 레코드를 클라이언트에 추가하면 클라이언트 데이

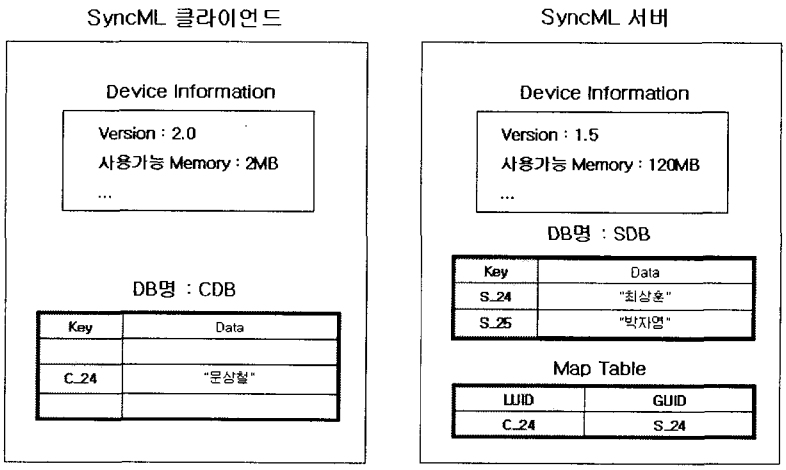


그림 4. 동기화 이전 서버, 클라이언트

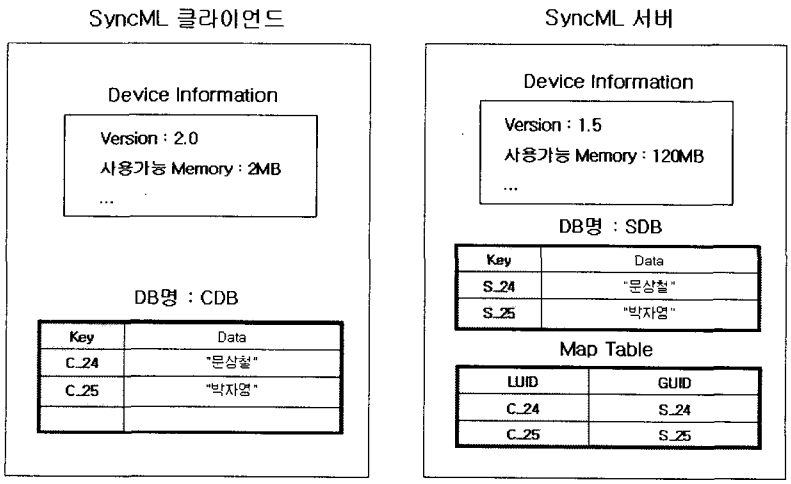


그림 5. 동기화 이후 서버, 클라이언트

터베이스에 'C_25' 레코드로 "박자영"이 생겨난다. 마지막으로 클라이언트 'C_25'와 서버 'S_25'가 동일한 정보를 표현하고 있다는 것을 매칭 시켜주는 정보를 맵 테이블에 추가하였다.

6.2 패키지 교환 흐름도

두 데이터베이스 간 동기화를 위해 그림 6과 같이 6단계로 패키지를 교환한다.

(1) 패키지 1, 2

클라이언트와 서버는 패키지 1, 2를 통해 서로 간의 동기화를 요청, 응답하게 된다. 이 패키지 1, 2를 통해 동기화 타입, 동기화 할 데이터베이스, 상호간의 인증 및 디바이스 정보를 교환할 수 있다.

(2) 패키지 3, 4

패키지1, 2가 끝나면 실제 데이터가 동기화 하는데 필요한 여러 기본 정보와 동기화 방식에 대한 정보교환이 완료된다. 클라이언트는 패키지3을 통해 클라이언트의 데이터 변동 정보를 서버 측으로 전송하여 동기화를 요구한다. 서버는 패키지 4를 통해 요구 받은 동기화를 처리하고 처리결과를 Status 명령어로 만드는데, 이는 양방향 싱크이기 때문에 서버측도 데이터 변동 정보를 클라이언트로 전송한다. 서버는 내부적으로 클라이언트와 서버간의 데이터식별 ID를 일치시키기 위해 맵 테이블을 가지고 있는데, 서버 측 데이터 변동 정보가 추가(Add)일 경우 클라이언트는 아직 데이터를 추가한 것이 아니기 때문에 패키지 5, 6을 통해 맵 테이블 정보를 획득한다.

(3) 패키지 5, 6

패키지 4를 통해 서버 측 데이터 동기화 요청을 받은 클라이언트는 처리결과를 패키지5를 통해 서버 측에 전송한다. 만약 패키지 4를 통해 추가 명령어를 받았다면 서버 측 맵 테이블을 유지하기 위한 정보를 맵 명령어에 담아 서버 측에 전송한다. 한 가지 기억해야 할 것은 서버 측 추가를 받은 클라이언트는 반드시 서버 측으로 맵 명령어를 보내야 하는 것은 아니다. 서버와 클라이언트가

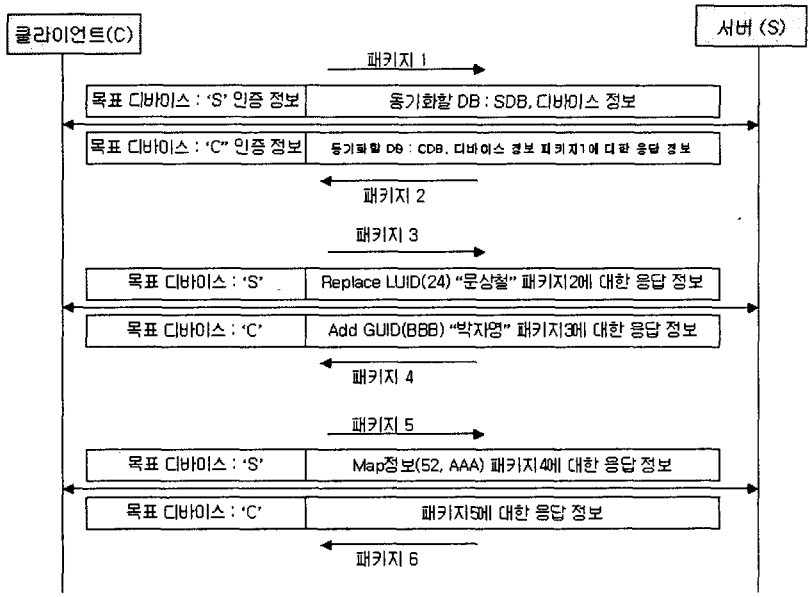


그림 6. 양방향 메시지 교환

서로 다른 데이터식별 ID를 사용하는 경우에만 맵 명령어가 필요한 것이다. 데이터식별ID가 동일한지에 대한 판단은 어플리케이션이 판단을 내려줘야 한다. 맵 테이블에 대한 정책(policy)을 어플리케이션으로부터 획득하여 맵 테이블 문제를 해결하고 서버측은 맵 명령어에 대한 처리를 하고 패키지 6을 통해 클라이언트로부터 맵 명령어에 대한 처리결과를 반환한다.

7. SyncML 응용

7.1 시스템 구성 예

SyncML 구현 예로 데스크탑 PC 에 RDBMS 설치하여 서버를 구성하고 임베디드 DB를 설치한 PDA 클라이언트와 RMS을 가지고 있는 핸드폰 클라이언트로 그림 7과 같이 시스템을 구성한다. 데이터 동기화는 서버와 PDA는 Two-way 싱크로 HTTP 프로토콜을 사용하고 핸드폰은 Two-way 싱크로 TCP/IP 프로토콜로 동기화 한다.

7.2 소프트웨어 구성 예

소프트웨어 구성은 그림 8과 같이 SK-VM 기반으로 J2ME을 활용하여 핸드폰 App 연동모듈을 구현한 핸드폰 클라이언트와 WinCE 기반에

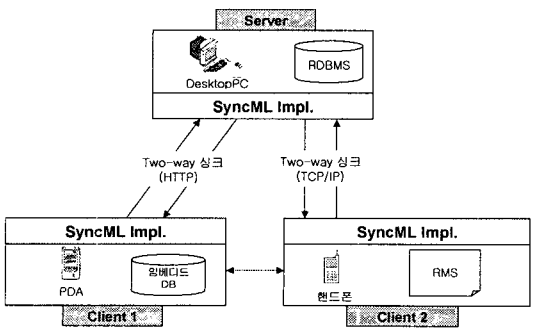


그림 7. 시스템 구성도

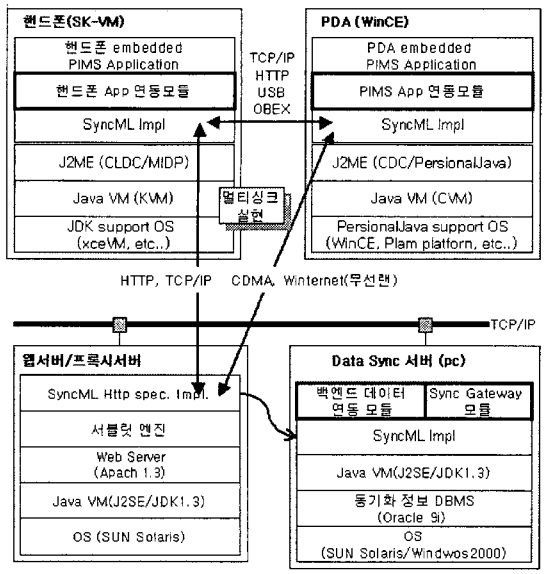


그림 8. 소프트웨어 구성도

J2ME을 활용하여 PIMS App 연동모듈을 구현한 PDA 클라이언트로 구성하고 핸드폰과 PDA 간에는 TCP/IP, HTTP, USB, OBEX 등을 이용하여 동기화 한다. 서버는 SyncML HTTP Spec을 구현한 웹서버와 백엔드 데이터 연동 모듈과 Sync Gateway 모듈을 구현한 DataSync 서버를 구성하여 서로 간에 HTTP, TCP/IP 통신을 통하여 동기화 한다.

8. 결 론

본문에서는 XML에 기반한 동기화 표준인 SyncML의 표준화 동향과 SyncML 의 탄생 배경을 설명하였고 SyncML스펙인 표현 프로토콜, 동기화 프로토콜, 트랜스포트 바인딩에 대해 설명하였다.

동기화 표준이 제정됨에 따라 독립적으로 실행되던 응용 프로그램들이 상호 자료를 교환할 수 있게 되고 어느 네트워크에 연결된 데이터라도 동기화 할 수 있는 기반이 닦임에 따라 자료의 활용도를 높일 수 있게 되었고 사용자에게 편리하게 정보를

제공할 수 있게 되었다. 국내에서는 몇몇 업체를 중심으로 SyncML 개발이 활발히 이루어지고 있으며 SyncFest를 통과한 업체들도 등장하고 있다.

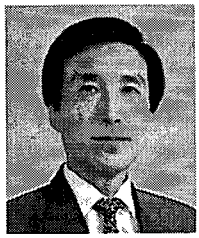
SyncML 이 널리 사용되고 사용자들에게 더 많은 혜택을 주기 위해서는 SyncML 을 이용한 응용 서비스와 응용 프로그램 개발이 필요하다. 특히 SyncML은 무선 인터넷상의 플랫폼에서 무선 인터넷이 갖는 제약 사항을 어느 정도 보완할 수 있기 때문에 무선 인터넷 서비스 분야에서 적극적으로 활용할 수 있다.

참 고 문 헌

[1] www.syncml.org
[2] SyncML Device Information DTD, SyncML.
[3] SyncML Meta Information DTD, SyncML.
[4] SyncML Synchronization Protocol, SyncML.
[5] WAP Binary XML Content Format Specification, WAP Forum.
[6] eXxtensible Markup Language (XML) 1.0, W3C.

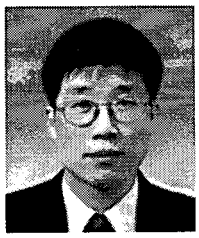
서 동 환

- 2002년 한성대학교 산업공학과 졸업(공학사)
- 2006년 서울산업대학교 산업대학원 컴퓨터 공학과 졸업(공학석사)
- 2006년1월~현재 (주)지어소프트 시스템 3팀 과장
- 관심분야 : 모바일 컨버전스, 모바일 메시지(SMS, MMS, 인스턴스메세지, 모바일 메일)



조 병 호

- 1978년 고려대학교 전자공학과 졸업(공학사)
- 1983년 서울대학교 대학원 컴퓨터공학과 졸업(공학석사)
- 1992년 서울대학교 대학원 컴퓨터공학과 졸업(공학박사)
- 1978년~1981년 금성통신㈜ 연구소 소프트웨어연구실 연구원
- 1984년~현재 서울산업대학교 공과대학 컴퓨터공학과 교수
- 관심분야 : 시스템 소프트웨어, 결함허용 시스템, 분산처리 시스템



이 길 흥

1989년 연세대학교 전자공학과 졸업(공학사)
1991년 연세대학교 대학원 전자공학과 졸업(공학석사)
1999년 연세대학교 대학원 전기컴퓨터공학과(공학박사)
1991년~1995년 LG 정보통신 안양연구소 네트워크 그룹
2000년5월~현재 서울산업대학교 컴퓨터공학과 조교수
관심분야 : 에이전트 응용, 초고속망 관리