SyncML 어플리케이션에서의 데이터 충돌 해결 방안

김연수°, 최 훈
충남대학교 컴퓨터공학과
{kimys°, hchoi}@ce.cnu.ac.kr

Conflict Resolution Policies and Rules for SyncML Applications

Youn Soo Kim°, Hoon Choi
Dept. of Computer Engineering, Chungnam National University

요 약

PDA(Personal Data Assistant)나 Handheld PC와 같은 모바일 디바이스(Mobile Device)를 이용하여 주소록이나 일정관리와 같은 개인 정보를 관리하는 것이 일반적인 추세이다. 모바일 디바이스는 그 특성상 중요한 정보를 신뢰성 있는 서버와 같은 컴퓨터에 중복 저장하는 것이 필요한데, 동일한 데이터를 중복 저장하면 데이터 동기화(Data Synchronization)를 통하여 일치성을 유지하도록 해야 한다. 그러나 동기화 과정에서 데이터 충돌이 발생할 수 있다. 본 논문에서는 이러한 데이터 충돌에 대한 해결 정책을 제시하였고, 그 중 서버 우선 정책과 클라이언트 우선 정책에 필요한 해결 규칙을 설명하였다. 서버 우선 정책은 서버에 이미 존재하고 있는 값을 유지하는 것을 기본으로 하는 정책이며, 클라이언트 우선 정책은 클라이언트가 요청한 값이 우선하는 정책이다.

1. 서론

PDA(Personal Data Assistant)나 Handheld PC와 같은 모바일 디바이스의 보급이 확산되고 무선 인터넷이 발전함에 따라 주소록이나 일정관리와 같은 개인 정보를 모바일 디바이스를 이용하여 관리하는 것이 일반적인 추세가 되어가고 있으며, 나아가 비즈니스의 새로운 수단으로 발전하고 있다. 그러나 이러한 모바일 디바이스는 휴대가 용이해야 하는 특성상 물리적인 크기가 작아야 하기 때문에 소규모의 저장공간만을 가지며, 성능상의 제약이 있다. 따라서 디바이스의 중요한 정보를 신뢰성 있는 서버와 같은 컴퓨터에 중복 저장하는 것이 필요하다. 그러나 동일한 데이터에 대한 중복 저장은 데이터 일치성을 유지하기 어렵게 만든다. 따라서 데이터 동기화(Data Synchronization)가 필요하다. 데이터 동기화는 분산된 데이터 저장소에 중복되어 존재하는 동일한 데이터의 값이 일치성을 유지할 수 있도록 하는 중재 동작이다. 데이터 동기화가 이루어지면 서버와 모바일 디바이스 사이의 데이터는 같은 값을 유지할 수 있게 된다 .

한명의 사용자가 여러 개의 디바이스를 소유하게 되는 경우도 있을 수 있다. 그리고 그 사용자는 여러 개의 모바일 디바이스를 가지고 자신의 정보를 관리할 수 있으며, 하나의 개인정보를 각각의 디바이스에 중복 관리할 수도 있다. 이러한 경우 각 디바이스들에 존재하는 중복된 정보에 대해서는 역시 동기화를 통하여 일치성을 유지해야 하는데, 중복된 정보를 서로 다른 두가지 이상의 값으로 각각 변경한 후에 동기화를 시도한다면 동기화 과정에서 어떤 값을 선택해야 할지 결정할 수 없기 때문에 데이터 충돌이 발생할 수 있다. 데이터 충돌이 발생하게 되면 충돌을 해결하기 위한 규칙이 꼭 필요하다. 그러나 아직까지 명확한 규정이나 표준적인 충돌 해결 규칙이 정의 되어 있지 않은 상태이며, SyncML(Synchronization Markup Language)에서도 데이터 충돌 해결에 대한 부분은 아직 미정이다. 다만 단말기 제조업체들이 나름대로의 충돌 해결 규칙을 정의해서 사용하고 있을 뿐이다. 따라서 본 연구에서는 데이터 충돌이 발생했을 때 이를 해결할 수 있는 일반적인 충돌 해결 규칙을 제안 한다.

본 논문의 2장에서 SyncML의 간단한 소개와 SyncML 에서 사용되는 Change Log 정보에 대해 설명하고 3장에서는 데이터 충돌 해결 정책을 제시하고, 4장에서는 데이터 충돌 해결 정책에 따른 해결 규칙을 기술한다.

2. SyncML

2.1 SyncML 프로토콜

표준 데이터 동기화 프로토콜의 필요성에 따라 IBM, Lotus, Motorola, Nokia, Palm, Psion, Starfish Software 등 모바일 관련 업체를 중심으로 SyncML그룹을 구성하여 [1,2] 서로 다른 컴퓨터 플랫폼, 네트웍, 응용 서비스에 이용될 수 있는 데이터 동기 방식의 개방형 표준인 SyncML을 제정하였으며, 현재 SyncML 규격 1.0.1 까지 발표되었다. SyncML은 XML(Extensible Markup Language) 형태로 표현된 데이터를 포함한 메시지를 서버와 디바이스간에 주고 받음으로써 서버와 디바이스간에 데이터 동기화를 수행한다 [3].

2.2 Change Log 정보

[표 1] 은 SyncML 서버에서 관리하는 Change Log 정보를 나타낸 것이다.

[표 1] Change Log 정보 테이블

UserID	SeqNum	DeviceID	dbID	actionFlag
Kim	10001	1	A	A
Kim	10002	2	C	R
Lee	10034	1	A	R
Lee	10023	2	C	D
...				

° 이 연구는 BK21충남대학교 정보통신인력양성사업단의 지원을 받았음

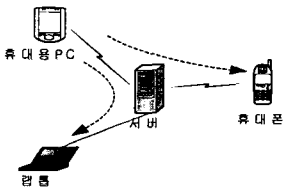
Change Log는 서버와 클라이언트 디바이스 사이에 동기화가 일어날 때, 변경된 정보 즉, 동기화가 이루어야 하는 데이터의 기록이다 [4]. 서버측 데이터베이스에 존재하는 데이터에 대해서 변경 여부를 각 디바이스 별로 기록해 둔 것으로, 이 Change Log를 이용하여 동기화 시에 클라이언트 측 데이터를 어떻게 변경할 것인지를 결정한다. 따라서 어떤 디바이스에 대해서 Change Log에 'replace' 로 기록되어 있다면 서버는 클라이언트에게 해당 데이터 아이템의 값이 변경 되었음을 알리고, 변경하도록 한다.

[표 1]의 각 항목에 대한 설명은 다음과 같다.
- UserID
  클라이언트 디바이스를 소유하고 있는 사용자의 ID
- GuidNum
  클라이언트 어플리케이션 데이터에 대한 Global ID
- DeviceID
  한명의 사용자에 속한 클라이언트 디바이스 ID
- dbID
  SyncML 응용 서비스 종류
  ("A" :addressbook "C" :calendar)
- actionFlag
  클라이언트 디바이스에게 보낼 동기화 커맨드의 종류
  ("A" :add "R" :replace "D" :delete)

예를 들어 [표 1]의 첫번째 항목은 "Kim" 이라는 사용자의 1번 디바이스에 addressbook의 GUID 가 10001인 데이터를 추가하라는 의미이다.

3. 데이터 충돌 해결 정책

모바일 디바이스와 신뢰성 있는 서버와의 동기화는 간헐적으로 이루어진다. 모바일 디바이스는 항상 네트웍에 연결되어 있지 않고, 필요한 때에만 네트웍에 연결하여 동기화를 수행하는 방식을 취한다. 따라서 일반적인 분산 데이터베이스나 파일시스템과는 달리 변경된 내용이 변경 후 바로 반영되는 것이 아니라 동기화를 성공적으로 마친 후에 반영이 되며, 동일한 정보를 관리하는 다른 디바이스 역시 서버와 동기화를 해야만 변경된 정보를 알 수 있다 <그림 1>.



<그림 1> 모바일 환경에서의 데이터 동기화 모델

따라서 클라이언트 디바이스가 응용서비스의 어떤 데이터 항목에 대해서 변경을 가한 후 동기화를 시도하고자 할 때 서버측의 동일한 항목이 이미 다른 디바이스에 의해서 변경이 되었고 현재 동기화를 시도하려고 하는 클라이언트 디바이스가 이러한 변경사실을 모른채 자신의 데이터를 또다시 변경하여 동기화를 시도하는 것이라면 의미적 충돌(semantic conflict)이 발생한다.

의미적 충돌을 해결(conflict resolution)하는 과정은 크게 2단계로 구분해 볼 수 있으며, 각 단계에서 취하는 행동은 충돌 해결 정책에 따라서 달라진다 [표 2].

첫째, 클라이언트가 동기화를 요구하는 데이터 항목에 대해서 서버측 데이터베이스의 동일 항목의 값을 서버의 값으로 유지하거나 클라이언트의 값으로 변경

둘째, 현재 동기화를 시도하고 있는 디바이스를 제외한 나머지 디바이스들의 서버 Change Log 기록을 변경

[표 2] 데이터 충돌 해결 정책

충돌 해결 정책	의 미
송신자 우선 (Originator Win)	메시지를 요청한 측의 데이터에 우선순위를 둔다
수신자 우선 (Recipient Win)	메시지를 수신한 측의 데이터에 우선순위를 둔다
클라이언트 우선 (Client Win)	디바이스의 데이터에 우선순위를 둔다
서버 우선 (Server Win)	서버의 데이터에 우선순위를 둔다
최근 데이터 우선 (Recent Data Win)	최근에 생성/변경된 데이터에 우선순위를 둔다
중복화(Duplication)	데이터 충돌이 발생할 경우 기존의 데이터는 그대로 유지하고 새로운 데이터로 인식하여 데이터를 따로 생성한다

본 논문에서는 기본적인 여섯가지의 충돌 해결 정책 중 서버 우선 정책과 클라이언트 우선 정책에서의 충돌 해결을 위해 각각 4가지의 Rule을 제안한다.

4. 데이터 충돌 해결 규칙

4.1 서버 우선 정책

서버 우선 정책은 데이터 충돌 발생시 서버에 이미 존재하고 있는 값을 유지하는 것을 기본으로 하는 정책이다. 즉 서버측 데이터를 선택하는 방법이며, 클라이언트의 요청은 무시된다.

[표 3] 서버 우선 정책의 충돌 해결 규칙

$op^t(o_i)$	$op^{t-1}(o_i)$	Rule	Action
Add	add	error	T.1
	delete		
	replace		
Delete	add	delete/error	T.2
	delete	delete	T.3
	replace	replace	
Replace	add	replace/error	T.2
	delete	delete	T.4
	replace	replace	

$op^{t-1}(o_i)$은 데이터 객체 o_i 에 대해 현재 동기화를 시도하고 있는 클라이언트가 적용해야 할 change-log의 기록이며, $op^t(o_i)$ 은 클라이언트가 현재 요청한 커맨드이다. 즉 $op^{t-1}(o_i)$ 은 클라이언트가 서버로부터 받게되는 커맨드이고, $op^t(o_i)$은 클라이언트가 서버에게 요청하는 커맨드이다. 각 충돌의 경우에 대한 해결 규칙을 살펴보면 다음과 같다.

Rule T.1 : 서버측에 이미 'add/delete/replace' 로 처리가 된 데이터에 대해서 클라이언트가 다시 'add' 로 요청하는 것은 불가능하다. 클라이언트의 'add' 요청에 대해 서버는 새로운 GUID 를 할당한 후 추가하기 때문에 $op^{t-1}(o_i)$에 의해 'add' 된 o_i와 $op^t(o_i)$에 의해 'add' 되는 o_i는 서로 다른 GUID를 가지게 되며, 전혀 새로운 아이템으로 인식하게 된다. 따라서 위의 표에서와 같은 경우는 있을 수 없으며 서버는 오류로 처리하고 클라이언트에게 오류가 발생되었음을 알린다.

Rule T.2 : 이 경우 두가지로 나누어 생각해 볼 수 있다.

T.2.1 서버의 change-log 기록이 'add' 로 남아있는 경우는 해당 데이터 아이템에 대해서 클라이언트가 전혀 알지 못하기 때문에 이 데이터 아이템에 대해서 'delete' 또는 'replace' 요청을 한다는 것은 정상적인 동기화 과정에서는 생길 수 없는 충돌유형이다. 따라서 이러한 경우도 T.1과 마찬가지로 오류로 처리한다.

T.2.2 만약 현재 동기화를 요구하는 클라이언트가 직전까지 모든 데이터 항목에 대해서 동기화를 마친 상태에서 데이터 항목을 변경한 후 동기화를 시도하는 것이라면, 즉 '$op^{t-1}(o_i)$=NULL' 이라면 현재 동기화를 요구하고 있는 클라이언트 디바이스는 충돌이 발생하지 않는다. 따라서 충돌이 발생하지 않은 동기화 과정과 동일하게 처리한다. 그러나 이때 아직 동기화를 마치지 못한 다른 클라이언트들은 아직 'add' 되지도 않은 데이터 아이템을 'delete' 하거나 'replace' 해야 하므로 충돌이 발생한다. 따라서 다른 클라이언트에 대한 change-log를 변경해 주어야 하는데, 'delete' 일 경우에 클라이언트에 데이터 아이템이 아직 반영되지 않았으므로 change-log 값을 삭제하면 된다.

'replace' 의 경우 클라이언트는 자신이 그 데이터 아이템을 가지고 있지 않으면 'add' 와 동일하게 처리하므로 그대로 둔다.

Rule T.3

서버의 change-log 에 'delete/replace' 로 남아있는 데이터 아이템에 대해서 클라이언트가 'delete' 를 요청하는 경우이다. 이 경우 서버에 남아있는 값이 우선하기 때문에 클라이언트의 요청을 무시한다. 그리고 현재 동기화를 요청한 클라이언트와 다른 클라이언트의 change-log 값을 변경해야 하는데 서버의 change-log 값이 'delete' 라면 'delete' 로 유지하고 'replace' 라면 'replace' 로 유지한다. 이때 'replace' 로 유지하게 되면 클라이언트는 'add' 로 처리하게 된다.

Rule T.4

서버의 change-log 에 'delete/replace' 로 남아있는 데이터 아이템에 대해서 클라이언트가 'replace' 를 요청하는 경우이다. Rule T.3 와 마찬가지로 클라이언트의 요청을 무시하고, 현재 동기화를 요청한 클라이언트와 다른 클라이언트의 change-log 값을 변경한다. 서버의 change-log 값이 'delete' 라면 'delete' 로 유지한다. 이때 'delete' 로 유지하면 클라이언트는 해당 데이터 아이템을 지울 것이다. 'replace' 라면 'replace' 로 유지한다. 클라이언트는 데이터 아이템을 변경(replace)할 것이다.

4.2 클라이언트 우선 정책

[표 4] 클라이언트 우선 정책의 충돌 해결 규칙

$op^t(o_i)$	$op^{t-1}(o_i)$	Rule	Action
Add	add	Error	T.1
	delete		
	replace		
Delete	add	delete/error	T.2
	delete	Delete	T.3
	replace		
Replace	add	replace/error	T.2
	delete	Replace	T.4
	replace		

클라이언트 우선 정책은 데이터 충돌 발생시 클라이언트가 요청한 값이 우선하는 정책이다. 즉 서버측 데이터를 클라이언트가 요청한 값으로 변경한다.

클라이언트 우선 정책에서 Rule T.1 과 Rule T.2는 서버 우선 정책의 Rule T.1, Rule T.2와 동일한 경우이며 같은 규칙을 적용한다. 따라서 설명을 생략한다.

Rule T.3

서버의 change-log 에 'delete/replace' 로 남아있는 데이터 아이템에 대해서 클라이언트가 'delete' 를 요청하는 경우이다. 클라이언트의 요청값을 우선하기 때문에 서버 데이터를 클라이언트가 요청한 값으로 변경한다. 그리고 다른 클라이언트의 change-log 값을 변경해야 하는데 클라이언트의 요청 커맨드가 'delete' 이므로 모두 'delete' 로 변경한다.

Rule T.4

서버의 change-log 에 'delete/replace' 로 남아있는 데이터 아이템에 대해서 클라이언트가 'replace' 를 요청하는 경우이다. 클라이언트의 요청값을 우선하기 때문에 서버 데이터를 클라이언트가 요청한 값으로 변경한다. 그리고 다른 클라이언트의 change-log 값을 변경해야 하는데 클라이언트의 요청 커맨드가 'replace' 이므로 모두 'replace' 로 변경한다.

4. 결론

한 사람이 여러 개의 모바일 디바이스를 소유하고 이 디바이스들에 동일한 정보를 중복하여 저장하고 관리한다면, 데이터 동기화 과정에서 데이터의 충돌이 발생할 수 있다. 즉 같은 값을 가져야 하는 데이터가 서로 다른 두개 이상의 값이 되어 일치성을 유지할 수 없는 경우가 발생하게 된다. 이러한 충돌 상황을 해결하기 위해 나름대로의 해결 규칙을 정의해서 사용할 수는 있지만, 표준적인 정의가 없다.

본 연구에서는 모바일 컴퓨팅 환경에서 응용 가능한 6가지의 데이터 충돌 해결 정책을 제시하고, 서버 우선 정책과 클라이언트 우선 정책에 대한 데이터 충돌 해결 규칙을 제시하였다. 본 논문에서 제시한 데이터 충돌 해결 정책과 해결 규칙이 모바일 컴퓨팅 분야는 물론 분산 데이터베이스, 파일 시스템 등 좀더 다양한 데이터 동기화 분야에서 활용 가능할 것으로 생각한다.

5. 참고 문헌

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

[2] SyncML Initiative, Building an Industry-Wide Mobile Data Synchronization Protocol, SyncML White paper, Mar. 20, 2000

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

[4] SyncML Initiative, SyncML Synchronization Protocol version 1.0.1, June 15, 2001

[5] JiYeon Lee, ChangHoe Kim, Jong-Pil Yi, Hoon Choi, "Implementation of the Session Manager for a Stateful Server" ,2002 IEEE TENCON, Beijing, pp.387-390, Oct. 31,2002

[6] D. B. Terry at al., "Managing update conflicts in Bayou, a weakly connected replicated storage system," Proc. of the 15th Symposium on Operating Systems Principle, 1995