서비스 기반 모바일 어플리케이션의 MVC 아키텍쳐

이호중 ○ 라현정 김수동
숭실대학교 컴퓨터학과
{hjlee, hjla}@otlab.ssu.ac.kr, sdkim777@gmail.com

A Design of MVC Architecture for Service-based Mobile Applications

Ho Joong Lee ○ Hyun Jung La, Soo Dong Kim
Department of Computer Science
SoongSil University

요 약

모바일 디바이스는 유연한 이동성을 제공하는 대신에 제한된 자원을 가지고 있는 휴대용 장치로서, 어플리케이션 구동을 가능하게 하는 새로운 클라이언트 단말기로 빠른 속도로 보급되고 있다. 대표적인 모바일 디바이스로는 iPhone, 안드로이드 폰이 있다. 현재 이러한 모바일 디바이스에는 엔터테인먼트 관련 어플리케이션이 널리 사용되고 있지만, 앞으로 모바일 디바이스의 성장 및 네트워크 기술에 따라 다소 복잡한 어플리케이션을 필요로 하게 된다. 그러나, 모바일 디바이스에서 실행될 수 있는 어플리케이션의 복잡도에 제약이 발생하여 높은 복잡도의 어플리케이션은 실행이 힘들다. GUI 어플리케이션에서 데이터와 그것을 표현하는 부분을 나누어 설계하는 MVC(Model-View-Control) 아키텍쳐는 어플리케이션을 설계하는데 일반적으로 많이 사용되고 있지만, 이는 모바일 어플리케이션의 특징을 반영하지는 못한다. 그러므로, 본 논문에서는 자원 사용의 최적화를 고려한 서비스 기반 모바일 어플리케이션을 설계하기 위하여 기존의 MVC 아키텍처를 보완/확장한다. 본 논문에서 제안하는 MVC 아키텍쳐는 어플리케이션 특성에 맞춰 모바일 어플리케이션의 특성들을 극복하여 모바일 디바이스의 한계점을 극복하는 동시에 고성능을 보장하는 모바일 어플리케이션을 개발하는데 적용할 수 있다.

1. 서 론

기술의 발전으로 대부분의 사람들이 휴대폰과 멀티미디어 디바이스를 사용하고 있다. 하지만 휴대폰과 멀티미디어 디바이스를 모두 휴대하는 불편함이 점점 커졌으며 사용자들은 점점 소형화되기를 요구하였고 멀티미디어 디바이스들은 휴대폰으로 기능이 흡수되었다. 따라서 휴대폰만 있으면 많은 기능들을 사용할 수 있었고 휴대성도 증대되었다.

점점 기능이 늘어나는 휴대폰은 컴퓨터의 기능까지 담고 있는 스마트폰으로 발전하였다. 사용자들은 컴퓨터로만 하던 작업들은 휴대폰을 이용해서 할 수 있게 되었으며 기존의 휴대폰과 달리 어플리케이션을 사용자가 설치할 수 있어 필요한 기능을 휴대폰에 쉽게 추가할 수 있게 되었다.

많은 기능을 담고 있는 휴대폰의 발전 흐름에 발맞추어 네트워크의 기술도 발전하였다. 네트워크 기술은 휴대폰을 항상 인터넷에 연결하여 다양한 정보를 접할 수 있고 많은 서비스를 사용할 수 있는 것을 뜻하게 되었다.

휴대기술과 네트워크 기술이 합쳐서 다양한 모바일 디바이스가 출시되었고 이제 컴퓨터의 기술까지 흡수하고 있다. 현재 컴퓨터의 어플리케이션은 단순히 컴퓨터 차체의 기능만을 이용하는 것이 아니라 서비스 기반으로 변화되어 항상 연결되어 있는 인터넷을 활용하고 있다 [1]. 이러한 기술들은 컴퓨터의 기능들을 받아들이는 모바일 디바이스에도 적용될 수 있다. 모바일 디바이스의 특성에 맞게 적용된 어플리케이션이 모바일 어플리케이션이다.

모바일 디바이스의 특징에는 제한된 리소스, 물리적 위험, 풍부한 네트워크 연결성, 한정된 배터리 용량이 있다. 여기서 어플리케이션을 개발함에 있어서 가장 많이 고려되는 것이 제한된 리소스이다. 모바일 디바이스에서 실행될 수 있는 어플리케이션의 복잡도에 제약이 발생하여 높은 복잡도의 어플리케이션은 실행이 힘들기 때문이다.

하지만 이러한 특성을 모바일 디바이스의 다른 특성인 풍부한 네트워크 연결성을 이용하여 보완하는 방법이 있다. 네트워크를 동해 특정 서비스를 호출 할 수 있기 때문이다. 복잡한 어플리케이션은 서비스를 이용함으로써 모바일 디바이스에서 실행 불가능한 복잡도 높은 어플리케이션도 실행 가능하게 할 수 있다.

GUI 어플리케이션에서 데이터와 그것을 표현하는 부분을 나누어 설계하는 MVC 아키텍쳐는 어플리케이션을

설계하는데 일반적으로 많이 사용되고 있다[2]. 모바일 어플리케이션 또한 GUI를 가지고 있고 많은 기능들이 컴퓨터의 속성을 가지고 있기 때문에 MVC 아키텍쳐를 적용할 수 있다. 하지만 모바일 디바이스의 특성 때문에 발생되는 문제 때문에 서비스 기반의 모바일 어플리케이션으로 설계하는 경우가 생겨난다. 하지만 하나의 디바이스에서 실행되는 것을 기반으로 한 MVC 아키텍쳐는 서비스 기반의 어플리케이션에서는 서비스 이용에 대한 부분을 설명할 방법이 없다 따라서 MVC 아키텍처의 컨트롤러와 모델의 부분의 기능을 분할하고 각 부분들의 상호작용을 고려하여 어플리케이션을 설계해야 한다. 따라서 MVC 아키텍쳐도 모바일 환경에 맞게 특화되어야 한다.

모바일 어플리케이션은 모바일 디바이스의 휴대성 때문에 발생되는 문제점을 앉고 있으며 이것을 해결하기 위해 서비스 기반의 어플리케이션은 불가피한 선택이다.

본 논문은 숭실대학교 모바일 서비스 소프트웨어 공학센터(MSSEC)의 연구 프로젝트인 Mobile Mate Service(MMS) 수행 중 도출된 개발 기법의 일부를 기반으로 하여 기존 어플리케이션 개발에 자주 사용하는 MVC 아키텍쳐를 보완하여 서비스 기반 모바일 어플리케이션 개발에 활용할 수 있는 향상된 서비스 기반의 MVC 아키텍쳐를 제안한다. 3장에서는 서비스 기반 모바일 어플리케이션이 가지고 있는 특성과 서비스를 사용해 극복할 수 있는 사항들을 정의한다. 4장에서는 서비스 기반 어플리케이션에 적용할 수 있는 변화된 MVC 아키텍쳐를 제안한다. 5장에서는 제안된 아키텍쳐에서 각 레이어 간 상호작용에 대해 설명한다. 6장에서는 이것을 이용해 설계한 사례연구를 보여준다. 본 논문에서 제안하는 MVC 아키텍쳐는 어플리케이션 특성에 맞춰 모바일 어플리케이션의 특성들을 극복하여 모바일 디바이스의 한계점을 극복하는 동시에 고성능을 보장하는 모바일 어플리케이션을 개발하는데 적용할 수 있다.

2. 관련연구

Peter의 연구에서는 모바일 디바이스가 자바를 기반으로 된 어플리케이션의 실행이 가능하고 내장된 웹 브라우저가 있는 특징을 이용해 자신들이 개발하는 소프트웨어 아키텍쳐 설계하였다[3]. 설계한 아키텍쳐는 클라이언트와 서버로 나누어져 있으며 클라이언트에는 GUI, 비즈니스 로직으로 구성되어 있고 서버에는 비즈니스 로직과 데이터베이스로 구성되어 이 둘 사이는 네트워크로 연결되어 있다. 이러한 구조를 모델 기반으로 변경해 전체를 포괄하는 어플리케이션 모델이 존재하고 그 안에 GUI 모델로 화면을 담당하는 부분과 각각의 특징으로 특징 모델을 가지고 있는 아키텍쳐를 제안하고 있다. 하지만 클라이언트와 서버를 나누어 설계하였지만 그것에 대한 이점이 불분명하고 네트워크 비용을 줄이기 위한 방안이 다루어지지 않았다.

Chen의 연구에서는 MVC 패턴의 문제점을 두 가지 제시하 그것을 해결하기 위해 다른 디자인 패턴들을 이용하고 있다[4]. 첫번째 문제점은 모델과 뷰가 긴밀하게 연결되어 있는 있다는 것이다. 하나의 모델를 여러 뷰가 공유함으로써 데이터 의존성이 높아지기 때문이다. 따라서 필요한 뷰에 모델의 데이터 변화를 알려주는 옵저버 패턴을 이용해 이것을 해결하였다. 두번째 문제점은 모델과 비즈니스 로직 사이의 문제이다. 비즈니스 로직은 데이터 처리를 위해 데이터베이스에 접근을 많이 하게 되고 그것에 따른 오버해드가 있다는 것이다. 이것을 해결하기 위해 싱글톤 패턴과 팩토리 패턴, Value Object 패턴을 이용한다. 이 세가지 패턴을 이용해 비즈니스 로직이 데이터베이스에 접근하는 오버헤드를 줄였다. 여기서는 패턴을 이용하여 MVC 패턴의 문제점을 해결하고 있다.

Natchetoi의 연구에서는 모바일 어플리케이션의 문제점을 해결하기 위한 방안으로 SOA를 사용, 네트워크 연결에 대한 투명성, 서비스 결합에 대한 loose-coupling, 낮은 비용의 ownership을 제시하고 있다[5]. 이것을 실현하기 위해 이 논문에서는 모바일 디바이스를 위한 가벼운 SOA 기반 아키텍쳐를 사용하기 위해 5가지 기술을 보여주고 있다. 1) 전송데이터 최소화, 2) 고압축, 3) 효과적인 메시지, 4) 서버 데이터 보호, 5) 비연결 모드. 본 논문저자들은 이전에 XML를 효과적으로 압축하는 것을 연구하였고 이것을 이용하여 모바일 디바이스에서 데이터 전송 시 압축된 XML를 사용하는 것을 사용하고 있다. 그리고 Data Connection Manager를 사용해 낮은 네트워크 연결과 제한적인 메모리를 극복하는 방법을 제시한다. 보안에서는 Elliptic Curve Cryptography를 사용해 암호화한다. SOA의 네트워크 비용을 최소화하는 방법만을 제시할 뿐 어플리케이션이 어떠한 아키텍쳐를 가져야 하는지는 다루지 않았다.

3. 서비스 기반 모바일 어플리케이션의 개요

3.1. 모바일 어플리케이션의 특성

모바일 어플리케이션은 모바일 디바이스에서 특화되어 실행되는 어플리케이션이다[1]. 따라서 모바일 어플리케이션은 일반 어플리케이션과는 달리, 모바일 디바이스의 특징이 반영되어 개발되어야 한다. 다음은 모바일 어플리케이션 개발시 고려해야하는 모바일 디바이스의 특성이다.

- 부족한 컴퓨팅 자원

  모바일 디바이스는 편리한 이동성 때문에 크기가 작고 가볍다. 하지만 이러한 이점과 반대로 그만큼 성능 면에서는 컴퓨터보다는 많이 떨어진다. 메인 메모리, 저장공간이 작고 CPU의 성능이 낮다. 따라서 어플리케이션 설계 시 이러한 특징들을 고려해야 한다.

- 물리적인 위험

  모바일 디바이스를 휴대성이 강하기 때문에 그만큼 분실이나 파손의 위험이 많다. 따라서 개인적인 정보들이 많은 담고 있는 모바일 디바이스는 물리적인 위험에도 대비 해야 한다.

- 풍부한 네트워크 연결성

  모바일 디바이스는 무선 네트워크를 사용하며 휴대폰 망 또는 Wi-Fi를 사용한다. 하지만 모든 장소에서 이것을 이용할 수 없고 휴대폰 망의 경우 과금이 발생하며 Wi-Fi의 경우 한정적인 장소에서만 사용할 수 있기 때문에 네트워크의 연결은 가변적이다.

- 한정된 배터리 용량

  모바일 디바이스는 크기가 작기 때문에 배터리의 용량 또한 한정적일 수 밖에 없다. 배터리 기술의 많은 발전에 있지만 그 가격이 높아 쉽게 용량을 늘릴 수는 없는 상황이다.

3.2. 서비스 기반 모바일 어플리케이션

모바일 디바이스의 특성으로 모바일 어플리케이션은 높은 복잡성을 가진 어플리케이션을 수행할 수 없으며 휴대성으로인한 분실/파손의 위험성, 한정된 배터리 용량이라는 제약사항을 가진다. 히지만 이러한 부분들은 풍부한 네트워크 연결성이라는 특성을 이용해 일부 기능을 서비스 형태로 제공하므로써 제약사항을 극복할 수 있다.

서비스 기반 모바일 어플리케이션은 다비이스에서 실행해야 할 기능을 일부 또는 대부분을 서비스로 제공받는 어플리케이션을 말한다[5]. 서비스 기반 모바일 어플리케이션은 그림 1과 같은 구조를 가진다.

그림 1 서비스 기반 모바일 어플리케이션

서비스 기반의 모바일 어플리케이션은 표 1과 같이 모바일 디바이스 특성을 극복할 수 있다.

표 1. 서비스를 이용한 모바일 디바이스 특성 해결

제약사항	서비스 기능	해결
부족한 컴퓨팅 자원	풍부한 자원	복잡도 높은 연산 풍부한 저장공간
물리적인 위험	데이터 저장 및 보관	데이터 손실 위험 최소화
한정된 배터리 용량	복잡도 높은 연산 가능	배터리 소모 최소화
	데이터 저장 및 보관	데이터 손실 최소화

서비스 기반은 네트워크에 항상 연결되어 있다는 것을 가정하고 있고 이것을 이용하여 모바일 어플리케이션의 특징들을 많이 보완할 수 있다. 서비스는 성능 좋은 서버에서 제공되는 것으로 모바일 디바이스의 특징인 부족한 컴퓨팅 자원을 확장해서 사용할 수 있다. 또한 모바일 디바이스에 자료를 저장하는 것이라 서비스에 저장하여 이용함으로써 물리적인 위험에도 대비할 수 있다. 따라서 서비스 기반의 모바일 어플리케이션은 더 많은 기능을 구현하고 성능을 높이기 위해 사용할 수 있는 좋은 선택이다.

4. 서비스 기반 모바일 어플리케이션에 적용한 MVC 아키텍쳐

4.1. 기존 MVC 아키텍쳐

MVC 아키텍쳐는 소프트웨어 공학에서 널리 사용되는 패턴으로 GUI 어플리케이션에서 데이터와 그것을 표현하는 부분을 나누는 특성을 잘 나타내고 있다[2]. MVC 아키텍쳐는 그림 2와 같이 세 개의 계층인 모델, 뷰, 컨트롤러로 나누어지며, 각 계층은 서로 상호작용하여 기능을 수행하게 된다.

그림 2. 단일 어플리케이션의 MVC 아키텍쳐

모델은 어플리케이션의 데이터를 저장하는 객체이며 뷰는 모델이 가진 객체를 화면에 출력하는 것이고 컨트롤러는 사용자 입력으로부터 반응하는 것을 의미한다.

따라서 사용자 인터페이스가 존재하는 많은 어플리케이션에 적용되고 있으며 모바일 어플리케이션에도 자주

사용되는 아키텍쳐이다.

4.2. Thin-Client MVC 아키텍쳐

모바일 어플리케이션도 사용자 인터페이스를 가지는 어플리케이션 이기 때문에 기존 MVC 패턴을 동일하게 적용할 수 있다. 하지만 모바일 디바이스의 특성 때문에 어플리케이션이 적용될 때 문제점이 발생한다. 따라서 그림 3에서 나타난 것처럼 모바일 디바이스의 특성을 해결하기 위해 서비스가 필요하다. 하지만 기존의 MVC 아키텍쳐는 단일 어플리케이션에 대한 부분만을 고려하였기 때문에 서비스에 대한 부분이 고려되어 있지 않아 서비스 기반 어플리케이션에 적용하는 것은 적절하지 않다[6].

서비스 기반 모바일 어플리케이션에 MVC 패턴을 적용한 형태는 그림 3와 같은 Thin-Client MVC 아키텍쳐이다.

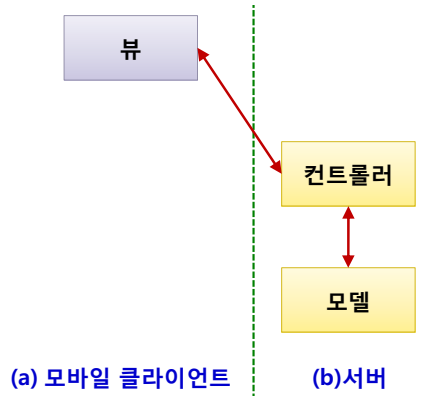

그림 3. Thin-Client MVC 아키텍쳐

Thin-Client를 위한 MVC 아키텍쳐에서 사용자에게 정보를 보여주고 입력을 받는 뷰는 모바일 클라이언트에 존재하고 컨트롤러와 모델은 서버에서 서비스로 제공된다.

Thin-Client MVC에서 컨트롤러가 모바일 클라이언트에서 분리됨으로써 복잡한 연산 시에 클라이언트의 낮은 연산 능력 때문에 발생하는 문제점을 극복할 수 있다. 그리고 모델의 분리는 부족한 데이터 저장공간을 확보하고 모바일 디바이스의 분실 및 파손으로 인한 데이터 손실을 막을 수 있다.

하지만 이 아키텍쳐의 문제점은 모든 데이터가 네트워크를 통해 이동하기 때문에 속도 및 연결 비지속성에 따른 문제점이 발생한다. 따라서 데이터 전송 시 데이터의 양을 줄일 수 있는 방법을 고려해야 한다.

4.3. Balanced MVC 아키텍쳐

Thin-Client MVC 아키텍쳐의 문제점으로 네트워크 비용이 언급되었다. 이것을 보완할 수 있는 방법 중 하나로 그림 4과 같은 Balanced MVC 아키텍쳐를 제안한다.

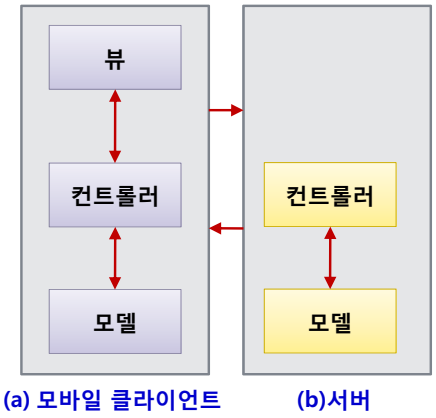

그림 4. Balanced MVC 아키텍쳐

컨트롤러와 모델이 서버에만 국한되어 존재하게 된다면 모든 정보와 결과들을 네트워크를 통해서만 가져오기 때문에 데이터을 빈번하게 이용하는 어플리케이션에서는 비효율적이다.

그러므로, 컨트롤러와 모델의 필수적인 부분들을 모바일 클라이언트에서 자주 사용되는 데이터의 사본을 가지고 있음으로써 네트워크 비용을 줄일 수 있다. 또한 모바일 클라이언트 디바이스 자체에 데이터가 저장되기 때문에 개인정보를 저장하기 적당하며 네트워크와 서버의 해킹에 따른 개인 정보 노출을 방지 할 수 있다. 하지만 클라이언트의 모델이 가지고 있는 데이터와 서버의 모델의 데이터와의 일관성은 문제가 될 수 있으며 클라이언트가 수정한 내용을 서버에 업데이트 시키고 다른 클라이언트가 서버의 모델의 정보를 수정하였다면 클라이언트로 그 내용을 알려줘 업데이트 하는 것이 필수적이다.

5. 모바일 클라이언트와 서버간의 상호작용

Balanced MVC 아키텍쳐는 컨트롤러와 모델이 모바일 클라이언트와 서버에 각각 존재하게 된다. 따라서 기존의 기능을 두 개로 분리되어 각각 2개의 컨트롤러와 모델이 생겨나며 각 구성요소의 상호작용이 존재한다. 그림 5와 같이 4개의 구성요소 사이에 5개의 관계가 있다.

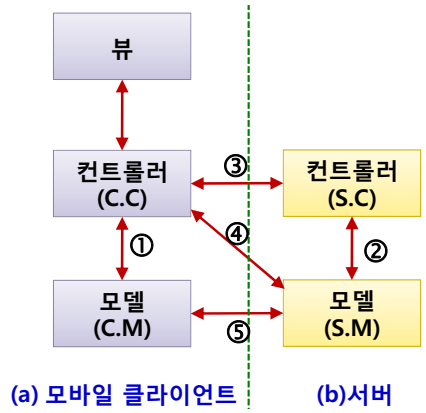

그림 5. 레이어 사이의 상호작용

5.1. 클라이언트 컨트롤러(C.C)-클라이언트 모델(C.M)

C.C와 C.M사이에서는 기존의 MVC 아키텍쳐의 기능들을 제공한다. C.C가 뷰에서 요청을 받으면 C.M의 정보를 가져와 그것을 연산 또는 변환하여 뷰에 반환한다. 이 상호작용은 기존의 단일 어플리케이션에서 사용된 컨트롤러와 모델의 역할과 같다. 하지만 C.C는 간단한 연산만을 하게 되어있고 C.M도 모든 데이터를 가진 것이 아니라 일부 데이터를 가지고 C.C에 제공한다. 이것은 캐시된 부분을 사용하거나 사용자의 개인적인 정보, 네트워크 연결이 끊겼을 때 정보를 임시 저장하는데 사용할 수 있다.

5.2. 서버 컨트롤러(S.C) – 서버 모델(S.M)

S.C와 S.M의 상호작용은 기존의 MVC의 컨트롤러와 모델 사이의 상호작용과 같다. S.C와 S.M은 클라이언트가 처리하기 힘들거나 부족한 메모리를 극복하기 위해 분리되어 제공되는 기능이기 때문에 S.C은 뷰의 요청을 받는 것이 아니라 C.C의 요청을 받는다. S.C는 요청 받은 처리를 하기 위해서 S.M에 데이터를 요청한다.

5.3. 클라이언트 컨트롤러(C.C) – 서버 컨트롤러(S.C)

C.C와 S.C는 기존의 MVC 컨트롤러의 기능을 분리하여 가지고 있다. C.C에는 필수적인 기능이 존재하고 S.C에는 복잡한 연산에 대한 부분이다. 그리고 이렇게 두 부분으로 나누어 져있기 때문에 각각 요청하고 반환하는 루틴이 추가된다. C.C는 S.C에 자신이 처리할 수 없는 많은 데이터를 읽어와 새로운 데이터를 만들거나 복잡한 연산들을 요청하고 S.C는 요청에 대한 답만을 C.C로 반환한다.

5.4. 클라이언트 컨트롤러(C.C) – 서버 모델(S.M)

C.C와 S.M는 기존의 MVC의 컨트롤러와 모델의 부분과 비슷하다. 실제 C.C가 사용하고 싶은 정보를 아무런 처리 없이 가져오고 싶을 때 발생되는 부분이다. 하지만 C.C와 S.M은 서로 네트워크로 연결되어 있기 때문에 많은 양의 데이터를 가져오는 것은 비효율적이다. 따라서 단일 데이터를 호출하는데 사용되며 많은 양의 데이터를 요청하여 처리하는 부분은 S.C에 맡겨 처리하며 데이터 자체에 대한 정보를 가져오기 위해서는 즉시 처리하는 것이 아니라 C.M과 S.M의 상호작용에 의해 C.M에 저장된 정보를 함께 이용하는 것이 더 효과적이다.

5.5. 클라이언트 모델(C.M) – 서버 모델(S.M)

C.M과 S.M의 관계는 기존 MVC의 모델이 두 부분으로 분리되었기 때문에 생겨났다. 그림 6와 같이 C.M과 S.M가 가지고 있는 데이터는 같은 부분도 있지만 각각 다른 부분이 있다. C.M은 사용자의 개인정보를 S.M로 보내지 않고 보관하고 있으며 S.M의 일부 데이터를 캐시형태로 보관하고 있다. S.M은 모바일 어플리케이션이 사용하는 공용 데이터를 가지고 있다.

여기서 발생되는 문제는 C.M과 S.M이 동시에 가지고 있는 데이터에서 발생한다. 데이터는 항상 동일하게 제공되어야 하기 때문에 데이터 수정이 일어났을 때 이것을 어떻게 동기화 시킬 것인가를 생각해야 한다.

C.M과 S.M의 관계에서 C.C가 요청하는 많은 양의 데이터를 C.M으로 가져와 제공하거나 C.M의 데이터를 백업하는 기능도 제공할 수 있다. 이러한 것을 C.C와 S.C를 통하지 않고 직접 이루어짐으로써 C.C의 부하를 줄일 수 있을 것이다.

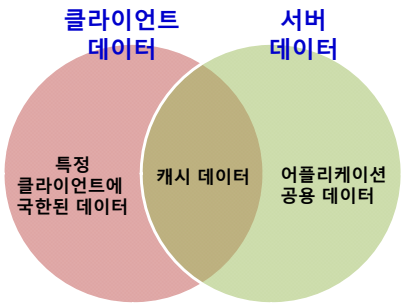

그림 6. 모델의 데이터 저장 관계

6. 사례 연구

본 장에서는 서비스 기반의 모바일 어플리케이션을 위한 Balanced MVC 아키텍처의 적용성을 보여주기 위하여, Mobile Mate Service (MMS)를 도메인으로 사례 연구를 수행한다. MSS는 모바일 디바이스 기반의 사용자의 위치정보를 이용하여 소셜 네트워킹을 지원하는 어플리케이션이다.

그림 7은 기능 뷰(Information View) 관점에서의 MMS 아키텍처를 보여준다. 그림의 왼쪽 부분은 모바일 디바이스에 배포되는 클라이언트 측 어플리케이션을 나타내며, 오른쪽 부분은 서버 측에 배포된 서비스를 나타낸다. Balanced MVC 아키텍처를 적용하여, 클라이언트측에는

모델, 컨트롤러, 뷰 모델로 구성되어 있고, 서버측에는 컨트롤러, 뷰 모델로만 구성되어 있다. 그리고, 클라이언트 컨트롤러와 서버 컨트롤러 간의 네트워크를 통한 상호작용(C.C - S.C)이 존재하고, 클라이언트 컨트롤러와 서버 모델 간의 상호작용(C.C - S.M)도 존재한다. C.C - S.M은 클라이언트 컨트롤러에서 Profile 정보를 접근하는 과정에서 발생한다. 별도의 가공이 필요없이 Profile 정보를 가져오면 되기 때문에, 서버 컨트롤러를 거쳐 성능을 저하시키는 것을 해결하기 위하여 직접 서버 측 모델인 Profile을 접근할 수 있도록 설계하였다.

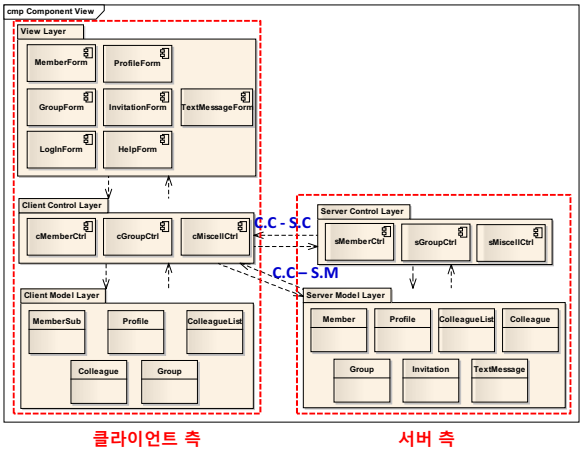

그림 7. MMS의 아키텍처 - 기능 뷰

서버 측에는 모바일 디바이스를 통해 MMS에 접근하는 모든 사용자의 정보가 저장되어 있으며, 다수 사용자 간의 소셜 네트워킹을 위한 기능을 제공한다. 그리고, 여러 사용자의 위치 정보를 검색하는 등의 다소 복잡한 계산을 필요로 하는 기능을 수행한다.

반면에, 모바일 디바이스에 배포된 어플리케이션에는 특정 개인과 관련된 정보와 다소 간단한 계산을 필요로 하는 기능만 가지고 있어 많은 자원을 필요로 하지 않는다.

그러므로, 모바일 디바이스 상에서 다소 복잡한 기능을 많은 자원을 소모하지 않고 사용할 수 있게 된다.

7. 결 론

모바일 디바이스는 제한된 리소스, 물리적인 위험, 풍부한 네트워크 연결성, 한정된 배터리 용량이라는 특징을 가지고 있다. 따라서 모바일 디바이스에서 실행될 수 있는 어플리케이션의 복잡도에 제약 존재한다. 이러한 특성을 보완하기 위해 모바일 디바이스의 다른 특성인 풍부한 네트워크 연결성을 이용한 서비스를 통해 복잡도가 높은 어플리케이션이 실행 가능해지며 풍부한 메모리 공간을 확보할 수 있다.

서비스는 모바일 어플리케이션의 단점을 보완할 수 있는 방법으로 본 논문에서는 기존의 어플리케이션 설계 시 일반적으로 많이 사용하는 MVC 아키텍쳐를 서비스 기반의 어플리케이션을 설계하는데 이용할 수 있도록 Thin-Client MVC 아키텍쳐와 Balanced MVC 아키텍쳐를 제안하고 아키텍쳐의 장단점을 제시하였다. 또한 Balanced MVC 아키텍쳐에서 발생하는 각 구성요소 간 5가지의 상호작용인 C.C-C.M, S.C-S.M, C.C-S.C, C.C-S.M, C.M, S.M을 제시하고, 각 상호작용의 장/단점을 비교하였다.

이렇게 도출된 기법을 숭실대학교 모바일 서비스 소프트웨어 공학센터 내 연구 프로젝트인 MMS 도메인에 적용하여, Balanced MVC 아키텍처의 적용성을 증명하였다.

본 논문에서 제안하는 MVC 아키텍쳐는 어플리케이션 특성에 맞춰 모바일 어플리케이션의 특성들을 극복하여 모바일 디바이스의 한계점을 극복하는 동시에 고성능을 보장하는 모바일 어플리케이션을 개발하는데 적용할 수 있다.

Acknowledgement

이 논문은 정보통신산업진흥원의 SW공학 요소기술 연구개발사업에 의해 지원 되었음을 밝힙니다.

8. 참고문헌

[1] I. Salmre, *Writing Mobile Code: Essential Software Engineering for Building Mobile Applications,* Addison-Wesley Professional, 2005 (chapter 2).

[2] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, 1994.

[3] P. Braun, R. Eckhaus, "Experiences on model-driven software development for mobile applications," *In proceedings of the 15th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems, 2008 (ECBS 2008),* pp.490-493, 2008.

[4] Chen Liyan, "Application research of using design pattern to improve layered architecture," *In proceedings of IITA International Conference on Control, Automation and Systems Engineering 2009 (CASE 2009),* pp.303-306, 2009.

[5] Y. Natchetoi, V. Kaufman, and A. Shapiro, "Service-oriented architecture for mobile applications," *In Proceedings of the 1st international workshop on Software architectures and mobility (SAM '08),* pp.27-32, 2008.

[6] Lai, A.M and Nieh, J., "On the performance of wide-area thin-client computing," *ACM Transaction on Computer System*, Vol.24, No.2, pp.175-209, 2006.