모바일 디바이스의 동적 서비스 디플로이먼트 기법

신동훈, 천두완, 김수동
숭실대학교 컴퓨터학부
e-mail : {shindonghoon7, dwcheun, sdkim777}@gmail.com

Dynamic Service Deployment Method for Mobile Devices

Dong-Hoon Shin, Du Wan Cheun, Soo Dong Kim
Dept. of Computer Science, Soong-Sil University

요 약

모바일 디바이스가 널리 보급됨에 따라 복잡한 기능을 수행하는 모바일 어플리케이션의 수요도 많아지고 있다. 또한, 자원의 제약성을 가지는 모바일 디바이스에서 복잡한 기능의 어플리케이션을 수행하기 위하여 서비스 개념을 이용하는 것은 보편적인 방법이다. 그러나 이런 서비스 기반 모바일 컴퓨팅에서는 안정성 및 성능의 부족과 같은 해결해야 할 중요한 문제들이 있다. 이런 문제점을 해결하기 위하여 본 논문에서는 동적 서비스 디플로이먼트 기법 및 설계를 제안한다. 이를 위해 3 장에서는 동적 디플로이먼트의 개념 및 장점을 설명한다. 4 장에서는 동적 디플로이먼트를 위한 설계를 보여주고, 5 장에서는 동적 디플로이먼트의 구현을 보여준다. 6 장에서는 실험을 통하여 동적 디플로이먼트의 효율성을 보여준다. 본 논문에서 제안하고 있는 동적 서비스 디플로이먼트 기법 및 설계는 서비스 기반 모바일 어플리케이션의 자원 사용량 감소, 성능 향상을 도와준다.

1. 서론

모바일 디바이스는 무선 인터넷 접속 기능을 제공하는 휴대 가능한 모든 종류의 장비들을 총칭하는 용어로서, 유연한 무선 인터넷 접속성과 높은 이동성을 제공한다는 장점을 가지고 있다. 모바일 디바이스가 널리 보급됨에 따라 복잡한 기능을 수행하는 모바일 어플리케이션의 수요도 많아지고 있다 [1]. 또한, 자원의 제약성을 가지는 모바일 디바이스에서 복잡한 기능의 어플리케이션을 수행하기 위하여 서비스 개념을 이용하는 것은 보편적인 방법이다 [2][3].

그러나 이런 서비스 기반 모바일 컴퓨팅에서는 안정성 및 성능의 부족과 같은 해결해야 할 중요한 문제들이 있다. 서비스 지향 컴퓨팅에서는 서비스 실행 요청이 과다하거나, 네트워크 대역폭이 감소하거나, 서비스가 갖고 있는 결함으로 인하여, 안정성이 감소하거나 성능이 낮아질 수 있다. 모바일 디바이스에서 사용하고 있는 3G 와 같은 네트워크 의 대역폭이 기존 컴퓨터에서 사용하고 있는 네트워크 대역폭에 비해 확연히 낮기 때문에 이런 문제점이 부각된다.

이런 문제점을 해결하기 위하여 본 논문에서는 동적 서비스 디플로이먼트 기법 및 설계를 제안한다. 이를 위해 3 장에서는 동적 디플로이먼트의 개념 및 장점을 요약한다. 4 장에서는 동적 디플로이먼트를 위한 설계를 보여주고, 5 장에서는 설계의 구현을 보여준다. 6 장에서는 실험을 통하여 동적 디플로이먼트의 효율성을 보여준다. 본 논문에서 제안하고 있는 동적 서비스 디플로이먼트 기법 및 설계는 다소 복잡한 구조의 서비스 기반 모바일 어플리케이션의 자원 사용량 감소, 성능 향상을 도와준다.

2. 관련연구

Haas 의 연구는 큰 규모의 복잡한 네트워크에서의 서비스 디플로이먼트에 관한 오토나믹 접근 방식을 설명한다[4]. 이 접근 방식은 효율적이고 유연한 서비스 디플로이먼트를 위한 글로벌 레벨과 로컬 레벨에서의 두 가지 방식의 메카니즘을 사용한다.

Konstantinos 의 연구는 큰 규모의 동적인 네트워크 환경에서의 서비스 배치 문제를 해결하는 방법으로서의 서비스 이주를 제안한다[5].

Nuno 의 연구는 동적인 설정 변경을 지원하는 모바일 Agent 를 위한 컴포넌트 기반의 프레임워크를 제시한다. 컴포넌트들은 어플리케이션의 실행에 방해를 최소화하면서 실행중에 더해지거나 제거될 수 있다[6]. 이것은 쉽게 적용 가능한 모바일 Agent 플랫폼들을 만들 수 있도록 해준다. 여기서 제시된 프레임워크는 JavaBeans 컴포넌트들을 위한 것이다.

지금까지의 연구들은 주로 동적 서비스 디플로이먼트의 개념에 초점이 맞추어져 있었지만 이 논문에서는 구체적인 설계 모델을 제시하고 직접 구현해봄으로써 실용성을 보여준다.

3. 동적 디플로이먼트

3.1. 동적 디플로이먼트의 개념

클라우드 서비스는 재사용성, 효율성 그리고 안정성과 같은 장점을 위해 모바일 어플리케이션 개발에 사용될 수 있다. 이를 활용하면, 모바일 디바이스의

자원 한계성을 극복하면서 복잡도가 높은 소프트웨어의 실행도 가능해진다. 그러나 클라우드 서비스를 호출하는 것은 피크타임이나 네트워크 통신량이 많은 시간에 요구되는 네트워크 통신으로 인해 실행시간이 느려질 수 있다. 본 논문에서 제안하는 동적 디플로이먼트는 이러한 경우에 사용되는 기법으로 서버 쪽의 기능을 모바일 디바이스에 동적으로 설치하고, 사용자 입장에서는 설치와 무관하게 디플로이된 기능을 사용할 수 있도록 지원하는 활동을 의미한다.

3.2. 모바일 동적 디플로이먼트의 장점

모바일 동적 디플로이먼트를 사용하여 얻을 수 있는 이점은 다음과 같다. 첫째, 네트워크 통신의 오버헤드를 최소화할 수 있다. 서비스 기반 모바일 어플리케이션에서 사용하는 기능에 따라 네트워크 통신이 지나치게 집중될 수 있고, 이는 CPU, 메모리, 배터리 등 리소스의 과다 소모를 야기시킨다. 이 때 동적으로 서비스를 모바일 디바이스에 디플로이 시켜서 네트워크 통신의 오버헤드를 줄이면, 리소스의 사용을 감소시켜서 안정적으로 관련 어플리케이션을 사용할 수 있다.

둘째, 서비스 기반 모바일 어플리케이션의 성능이 향상될 수 있다. 필요한 로직을 모바일 디바이스에서 직접 실행함으로써 네트워크 통신에 걸리는 시간이 소요되지 않으므로 원격지의 기능을 이용할 때보다 높은 성능으로 서비스를 이용할 수 있게 해준다. 이 밖에도 동적 디플로이먼트는 사용자의 사용성, 모바일 디바이스의 자원 사용 효율성 등을 증가시킬 수 있다.

4. 동적 디플로이먼트 설계

4.1. 아키텍처 설계

동적 디플로이먼트를 지원하기 위해서는 그림 1 에서처럼 서비스를 포함한 서버 쪽의 기능을 모듈로 관리 및 제공하기 위한 스테이션 노드와 관련 기능을 설치하거나 서버 쪽의 기능을 사용하는 모바일 노드로 나뉜다.

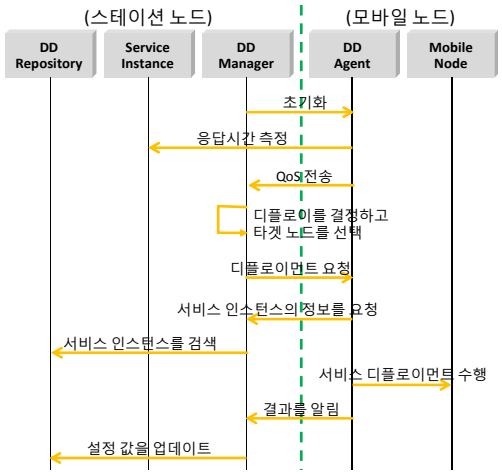

그림 1. 동적 디플로이먼트 지원을 위한 상호연동

그림 1 에서처럼 동적 디플로이먼트 프레임웍은 기능성을 수행한다. 어플리케이션에서 리모트 서비스를 사용하고 있는 동안 서비스의 Quality 가 떨어지게 되면 서버의 DD Manager 가 이를 감지하게 된다. DD Manager 는 노드에서 실행되고 있는 Agent 에게 디플로이를 요청하게 되고 노드의 상황에 따라서 Agent 가 응답을 하게 된다. 디플로이를 하게 되면 노드의 Agent 는 그 결과를 DD Manager 에게 보내고 DD Manager 는 그것을 Repository 에 저장한다. 모바일 노드에서는 변화된 Configuration 을 반영하여 상태값을 데이터베이스에 저장하고 어플리케이션은 리모트 서비스가 아닌 디플로이드된 서비스를 사용하여 결과값을 사용자에게 보여준다.

4.2. 모바일 노드의 클라이언트 설계

클라이언트는 모바일 디바이스에 설치되어 운영되는 것이기 때문에 모바일 디바이스의 제약적인 자원 상황을 고려하여 설계되어야 한다. 그림 2 에서처럼 클라이언트는 3 개의 계층으로 나뉘어져 있다.

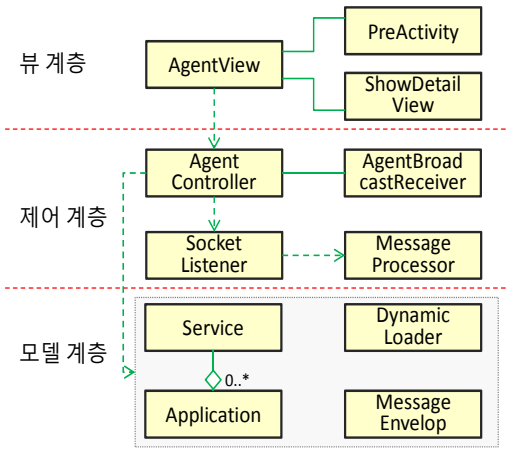

그림 2. 클라이언트 클래스 다이어그램

먼저, 뷰 계층에는 3 개의 클래스가 있다. AgentView 클래스는 클라이언트 어플리케이션의 사용자 인터페이스를 제공하는 기능을 하며 안드로이드 빌딩 블록 중 하나인 액티비티로 설계된다. 추가적으로 이 상세한 내용을 보여주기 위한 액티비티와 Preference 를 설정하기 위한 액티비티를 포함한다.

제어 계층에는 4 개의 클래스가 있다. 먼저, AgentController 는 사용자의 요청, 서비스 기반 모바일 어플리케이션과의 프로세스 간 통신, 스테이션 노드에 배치되어 있는 DD.Manager 와의 통신을 제어한다. 먼저, AgentBroadcastReceiver 는 서비스 기반 모바일 어플리케이션과의 프로세스간 통신을 담당하는 클래스로 기존 어플리케이션과의 의존도를 낮추기 위하여 기존 어플리케이션의 참조를 하지 않도록 설계한다.

SocketListener 와 MessageProcessor 클래스는 스테이션 노드에 배치되어 있는 DD.Manager 와의 소켓 통신을 지원하기 위하여 설계한 클래스로, 메시지를 주고받기 위한 프로토콜을 정의하고 처리하는 기능 및 DD.Manager 의 요청 및 응답을 처리할 수 있는 기능을 포함한다.

모델 계층에는 4 개의 클래스가 있다. 이 4 개의 클

래스 중 Service 와 Application 클래스는 동적 디플로이먼트 정보를 관리한다. DynamicLoader 는 동적 디플로이된 서비스 구현 모듈을 실시간에 로딩하여 인스턴스를 생성하고, 생성된 인스턴스의 레퍼런스를 가져와서 관련 메소드를 호출할 수 있게 해준다. MessageEnvelop 은 클라이언트와 서버 간 데이터를 주고 받을 수 있도록 만든 프로토콜을 위한 클래스이다.

4.3. 스테이션 노드의 서버 설계

서버 어플리케이션은 PC 와 같은 스테이션 노드에 배치되어 구동된다. 여러 개의 모바일 디바이스가 동시에 접근할 수 있어야 하기 때문에 이를 해결하기 위한 클래스를 포함한다. 그림 3 에서처럼 서버 어플리케이션은 3 개의 계층으로 나뉘어진다.

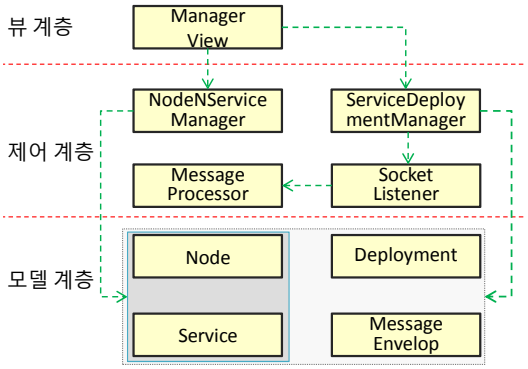

그림 3. 서버 클래스 다이어그램

먼저, 뷰 계층에는 하나의 클래스가 있으며, 서버 어플리케이션의 사용자 인터페이스를 제공한다.

제어 계층에는 4 개의 클래스가 있다. 먼저, NodeNServiceManager 는 node 와 service 에 관한 정보를 등록하거나 보여주거나 업데이트하거나 지우는 기능들을 전체적으로 관리하는데 사용된다. 이 클래스는 Controller 로써의 기능을 수행한다. ServiceDeploymentManager 클래스는 Agent 에게 디플로이먼트를 요청하거나 언디플로이먼트를 요청하는 메세지를 보낼 때 사용되고 그 결과에 따른 상태 값을 데이터베이스에 저장하는 역할을 한다.

SocketListener 와 MessageProcessor 클래스는 모바일 노드에 배치되어 있는 DD.Agent 와의 소켓 통신을 지원하기 위하여 설계한 클래스로, 메시지를 주고 받기 위한 프로토콜을 정의하고 처리하는 기능 및 DD.Agent 의 요청 및 응답을 처리할 수 있는 기능을 포함한다. 특히 SocketListener 의 경우, 클라이언트 서버 간 통신을 최적화하기 위하여 긴 시간 소켓 연결을 유지하는 대신에, 모바일 디바이스의 Agent 와 서버의 Manager 모두가 서버 역할을 할 수 있도록 설계한다. 이를 통하여 소켓 연결을 유지하지 않아도 상시 데이터를 주고 받을 수 있다. 또한, 데이터 전송과 응답을 비동기화 함으로써 네트워크 비용을 감소시키고 안정성을 높일 수 있다.

모델 계층에는 4 개의 클래스가 있다. 먼저, Node, Deployment, Service 클래스는 각각의 관련 정보를 관리하는 클래스이다. MessageEnvelop 은 클라이언트와 서버 간 데이터를 주고 받을 수 있도록 만든 프로토콜을 위한 클래스이다.

5. 구현

모바일 에이전트는 안드로이드 디바이스에서 구동되는 어플리케이션을 일컫는다. 이 어플리케이션은 Android 2.3 API 를 이용하고 있으며, 자원 효율성을 고려하여 구현되었다. 모바일 에이전트의 사용자 인터페이스는 그림 4 와 같다.

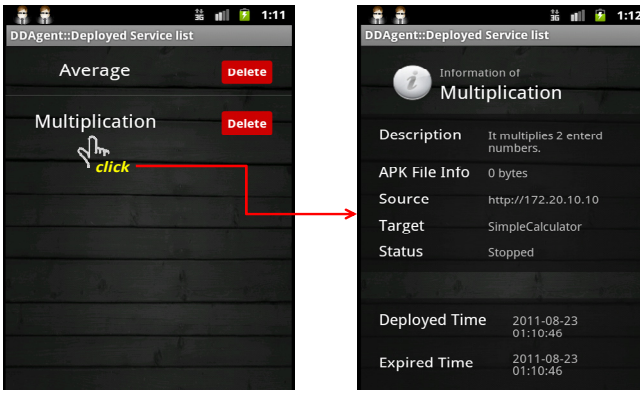

그림 4. 모바일 에이전트 사용자 인터페이스 화면

본 논문에서 제안한 설계에 따라 구현된 모바일 에이전트는 다음과 같은 특징을 갖고 있다. 첫째, 서비스 구현 모듈을 전송받은 모바일 에이전트는 모바일 디바이스에 *Class Loader* 를 이용하여 서비스를 인스턴스화 하여 필요한 기능을 사용한다. 이런 방법을 통하여 서비스 구현 모듈에 종속적이지 않은 방법으로 관련 기능을 사용할 수 있다. 둘째, 바인딩 정보를 업데이트하기 위한 방법으로 *브로드캐스트리시버 (BroadcastReceiver)*를 사용하였다. 이를 위한 구현은 그림 5 와 같다.

```
1.    // 모바일 에이전트
1.        if (serviceName.equals(%servicename%) ) {
2.            Intent intent = new Intent(SERVICE_DEPLOY_CH
    ANGE);
3.            context.sendBroadcast(intent);
4.        }// end of if
5.        else if (serviceName.equals(…)) {
6.            context.sendBroadcast(intent);
7.        }// end of else if.
8.    // 브로드 캐스트 리시버
9.    public class MyBroadcastReceiver extends BroadcastReceiver {
10.       static final String SERVICE_DEPLOY_STATE_CHANGE
11.   = "…";
12.       @Override
13.       public void onReceive(Context context, Intent intent) {
14.           if(intent.getAction().equals(SERVICE_DEPLOY_CHANG
    E)){
15.               %servicename%.setflag();
16.           else if(intent.getAction().equals(…)) {
17.               %servicename%.setflag();
18.           }//end of else if.
19.       }//end of onReceive.
```

그림 5. 바인딩 정보를 업데이트하기 위한 기법

이러한 방법을 사용하여 서비스 기반 모바일 어플리케이션들은 모바일 에이전트를 매번 참조해야 할 필요를 없앨 수 있다. 그것은 프로그램 구성원들 간

의 결합도가 줄어들었다는 것을 의미한다. 또한, 관련 서비스를 하나 이상의 서비스 기반 모바일 어플리케이션에서 사용할 수 있으므로 재사용성이 높아진다.

6. 실험

6.1. 실험환경

그림 6 에서처럼 동적 디플로이먼트를 실험하기 위하여 하나의 모바일 노드와 하나의 스테이션 노드를 사용하였다. 모바일 노드에는 두 개의 서비스 기반 모바일 어플리케이션인 SimpleCalculator 와 GradeCalculator 가 배치되어 있다. SimpleCalculator 는 서버에서 제공하고 있는 Average 서비스와 Multiplication 서비스를 사용하고 있고, GradeCalculator 는 Average 기능를 사용하고 있다. 또한, 동적 디플로이먼트를 지원하기 위하여 DD.Agent 컴포넌트가 배치되어 있다. 스테이션 노드에는 DD.Manager 컴포넌트와 DD.Repository 가 배치되어 있다. 이 컴포넌트와 저장소는 두 개의 서비스와 동적 디플로이먼트에 사용될 apk 파일을 관리하고 있다.

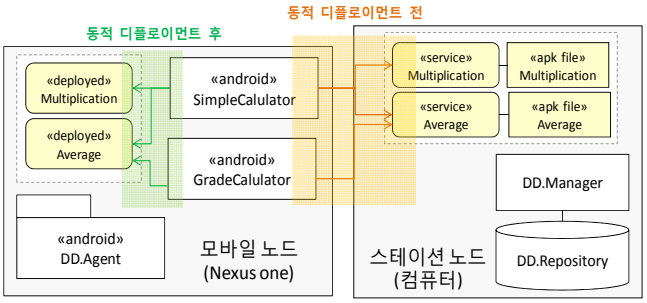

그림 6. 동적 디플로이먼트 실험 환경

6.2. 실험 결과 및 레슨

우리는 동적 디플로이먼트의 성능을 확인하기 위하여 SimpleCalculator 와 GradeCalculator 의 응답시간을 확인하였다. 응답 시간은 서버에 배치된 기능을 사용할 경우, 배치를 한 번 하고 여러 번 사용할 경우, 사용할 때마다 배치하는 경우를 고려하여 측정하였다.

그림 7 의 결과를 통해 다음과 같은 레슨을 얻을 수 있었다.

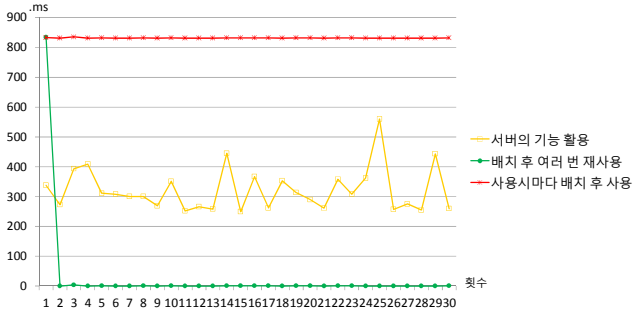

그림 7. 실험에 따른 응답 시간 결과

- 동적 디플로이먼트를 적용하지 않는 경우: 네트워크 통신의 상태에 의존적이다. 네트워크 통신의 상태가 좋은 경우와 좋지 않은 경우의 성능 차는 2 배 이상 나며, 네트워크 통신이 불가능 한 상태에는 가용성이 감소된다.
- 동적 디플로이먼트을 적용하는 경우: 한 번 배치해서 여러 번 사용하는 경우, 서버의 기능을 사용하는 경우보다 성능이 11 배 이상 향상됨을 확인할 수 있다. 그러나 한 번 배치한 다음 한 번만 사용하는 경우에는 동적 디플로이먼트를 하지 않도록 에이전트를 설계하여야 한다

7. 결론

서비스 기반 모바일 컴퓨팅에서는 안정성 및 성능의 부족과 같은 해결해야 할 중요한 문제들이 있다. 이런 문제점을 해결하기 위하여 본 논문에서는 동적 서비스 디플로이먼트 기법 및 설계를 제안하였다. 또한, 제안된 기법과 설계를 기반으로 구현한 동적 디플로이먼트 프레임웍을 활용한 실험을 통하여, 우리는 네트워크 통신의 오버헤드를 최소화할 수 있었고, 이에 따른 서비스 기반 모바일 어플리케이션의 성능 향상을 확인할 수 있었다. 즉, 본 논문에서 제안하고 있는 동적 서비스 디플로이먼트 기법 및 설계는 서비스 기반 모바일 컴퓨팅에서는 안정성 및 성능의 부족과 같은 중요한 문제의 해결에 기여한다.

Acknowledgement

이 논문은 정보통신산업진흥원의 SW공학 요소기술 연구개발사업에 의해 지원되었음을 밝힙니다.

이 논문은 2009년 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구입니다. (No. 2009-0076392)

참고문헌

[1] König-Ries, B. and Jena, F., “Challenges in Mobile Application Development,” *it-Information Technology*, Vol. 52, No. 2, pp. 69-71, 2009.

[2] Tergujeff, R., Haajanen, J., Leppanen, J., and Toivonen, S., “Mobile SOA: service orientation on lightweight mobile devices,” *In Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007)*, pp. 1224-1225, 2007.

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

[4] Haas, R., Droz, P., and Stiller, B., "Autonomic service deployment in networks," *IBM Systems Journal*, vol.42, no.1, pp.150-164, 2003.

[5] Oikonomou, K., and Stavrakakis, I., "Scalable service migration in autonomic network environments", *IEEE Journal*, vol.28, no.1, pp.84-94, January 2010.

[6] Santos, N., Marques, P., and Silva, L., “Dynamic deployment of services on mobile agents systems”, In Proceedings of the 2nd Workshop on Reflective and Adaptive Middleware/Workshop on QoS-enabled Component-Oriented Programming (RM2003/QOSCOP), pp. 130-134, Rio de Janeiro, Brazil, June 2003.

[7] Android Developers, http://developer.android.com/index.html (accessed September 15, 2011).