Journal of Korea Multimedia Society Vol. 11, No. 8, August 2008(pp. 1179-1193)

모바일 환경에서 웹 서비스의 이동

박명우[†], 김연석[††], 이경호[†††]

요 약

무선 네트워크와 모바일 디바이스의 발달과 확산은 웹 서비스 제공자로서의 모바일 디바이스를 가능하게 하였다. 하지만 불안정한 연결성과 빈번한 이동성은 모바일 디바이스에서 서비스를 제공하는데 큰 문제로 작용하고 있으며 이를 해결할 수 있는 가능성인 웹 서비스 이동 기술은 현재 데스크탑 환경을 중심으로 연구되고 있다. 본 논문에서는 코드 분할 기술을 이용한 모바일 웹 서비스의 이동 방법을 제안한다. 제안된 방법은 웹 서비스의 효과적인 이동을 위해서 먼저 웹 서비스 코드를 오퍼레이션의 선호도에 따라 적절히 분할한다. 높은 선호도를 가진 분할 코드는 선호도가 낮은 것보다 먼저 전송된다. 또한 트래픽이 집중되거나 용량이 큰 메시지를 주고 받을 경우, 컨텍스트 정보에 따라 적절한 디바이스로 웹 서비스를 복제하여 효과적인 서비스의 제공을 가능하게 한다.

Migration of Web Services in a Mobile Environment

Myung-Woo Park[†], Yeon-Seok Kim[††], Kyong-Ho Lee[†††]

ABSTRACT

Recently, mobile devices enabled with Web services are being considered as equal participants of the Web services environment. The frequent mobility of devices and the intermittent disconnection of wireless network require migrating or replicating Web services onto adjacent devices appropriately. This paper proposes an efficient method for migrating and replicating Web services among mobile devices through code splitting. Specifically, the proposed method split the source code of a Web service into sub-codes based on users' preference to its constituent operations. The sub-code with higher preference is migrated earlier than others. The proposed method also replicates a Web service to other devices to enhance its performance by considering context information such as network traffic or the parameter size of its operations.

Key words: Web services migration(웹 서비스 이동), Code splitting(코드 분할), Code mobility(코드 이동성)

1. 서 론

웹 서비스(Web services) [1,2] 는 인터넷과 같이 공개적인 네트워크 및 관련 표준기술을 통해 기존의 어플리케이션을 운영체제 및 프로그래밍 언어에 상관없이 상호운용이 가능하도록 하는 표준화된 소프트웨어 기술이다. 이러한 웹 서비스는 다양한 디바이스가 혼재되어 있는 유비쿼터스 환경에서 사용자가 필요로 하는 서비스의 제공을 가능하게 함으로서 유비쿼터스 환경을 위한 필수적인 소프트웨어 기술로

※ 교신저자(Corresponding Author) : 이경호, 주소 : 서울시 서대문구 신촌동 134(120-749), 전화 : 02)2123-3878, FAX : 02)365-2579, E-mail : khlee@cs.yonsei.ac.kr
접수일 : 2008년 1월 18일, 완료일 : 2008년 6월 18일
[†] 준회원, TMAX Soft R&D Center 연구원
(E-mail : myungwoo_park@tmax.co.kr)
[††] 준회원, 연세대학교 컴퓨터과학과 박사과정
(E-mail : yskim@icl.yonsei.ac.kr)
[†††] 종신회원, 연세대학교 컴퓨터과학과 부교수
※ 이 논문은 2006년도 교육과학기술부(MEST)의 재원으로 한국과학재단의 지원을 받아 수행된 연구임(No. R01-2006-000-10864-0)

인식되고 있다. 한편 웹 서비스는 모바일 컴퓨팅 기술과 결합하여 모바일 웹 서비스[3,4] 라는 보다 진화된 형태로 발전하고 있다. 그러나 불안정한 연결성 및 빈번한 컨텍스트 변화 등은 모바일 환경에서 웹 서비스를 제공하는데 많은 문제점을 야기시킨다. 따라서 이를 해결하고 사용자에게 지속적으로 서비스를 제공하기 위해서는 상황에 따라 적절한 디바이스로의 웹 서비스 이동 및 복제가 필수적이다.

무선 모바일 환경에서 웹 서비스를 이동(migration) 또는 복제(replication)시킬 수 있다면 상황에 따른 지속적인 서비스 제공이 가능해질 것이다. 예를 들어, 모바일 디바이스가 특정 장소에서 이탈하여 서비스를 제공하지 못할 때, 적절한 위치에 존재하는 디바이스로 웹 서비스가 이동된다면 지속적인 서비스 제공이 가능할 것이다. 또한 특정 모바일 디바이스로 서비스 요청이 집중될 때에는 다른 디바이스로 서비스를 복제하여 트래픽 집중을 막고 분산을 유도할 수 있을 것이다.

현재 웹 서비스의 이동과 관련한 연구는 주로 유선 네트워크 및 데스크탑 환경을 중심으로 이루어지고 있다. 기존 방법의 경우, 웹 서비스가 이동 중일때 해당 서비스에 대한 호출을 처리하는 방법이 제한적이며 대역폭이 작은 무선 환경에서 오랜 이동 시간을 요구한다. 그러므로 기존의 방법을 사용했을 경우 무선 모바일 환경에서의 웹 서비스 이동은 필연적으로 오랜 시간의 서비스 중단을 담보할 수밖에 없다.

따라서 본 논문은 코드 분할 기술을 이용하여 무선 모바일 환경에 적합한 웹 서비스 이동 방법을 제안한다. 분할된 코드 중에서 사용자의 선호도가 높은 코드를 우선적으로 이동시켜 이용률이 높은 오퍼레이션의 중단 시간을 최소화한다. 또한 다양한 경우에 웹 서비스 복제를 통하여 끊김없는 서비스 제공을 도모한다. 모바일 서버로 트래픽이 집중되면 다른 디바이스로 분할 코드를 복제하여 트래픽을 분산시킨다. 또한 큰 용량의 파라미터를 갖는 서비스가 요청될 경우 요청된 오퍼레이션에 관련된 분할 코드만을 클라이언트에 복제하여 전송에 소비되는 자원을 절약한다.

본 논문은 다양한 크기의 코드를 대상으로 실험을 하여 웹 서비스를 구성하는 오퍼레이션의 복잡성에 따라 분할에 걸리는 시간과 분할된 코드의 크기를 분석하였다. 또한 세 가지의 실세계 응용 시나리오 하에서 제안된 웹 서비스 이동 및 복제 방법이 효과적으로 동작함을 보였다. 한편 본 논문에서는 웹 서비스의 이동 및 복제를 위한 전략을 기술하지는 않으며 웹 서비스를 구성하는 소스 코드를 분할하고 분할한 코드를 통한 웹 서비스 이동 및 복제 방법만을 기술한다.

본 논문의 구성은 다음과 같다. 2절에서는 웹 서비스 이동 및 코드 이동성에 대한 기존 연구의 특징을 살펴본다. 3절에서는 웹 서비스의 코드 분할 방법을 기술하며 4절에서는 분할된 코드를 바탕으로 한 이동 방법을 설명한다. 5절에서는 제안된 방법을 평가하기 위한 실험 결과를 기술하고 구현된 프레임워크를 바탕으로 한 여러 시나리오를 소개한다. 끝으로 6절에서는 결론 및 향후 연구 방향에 대해 논한다.

2. 관련 연구

본 절에서는 기존의 웹 서비스 이동에 관한 연구 결과를 간략히 기술한다. Pratitha 등[5] 은 웹 서비스 이동 전략과 재설정 전략을 포함한 확장된 웹 서비스 기술 표준을 제시하면서 웹 서비스 이동을 지원하는 프레임워크를 제안한다. 웹 서비스를 제공할 수 있는 서버들의 컨텍스트 정보를 바탕으로 웹 서비스를 이동시킬 서버를 선택한다. 또한 각 컨텍스트 상황에 맞는 웹 서비스 모듈을 정의해 두고 이동시킬 때마다 서버의 컨텍스트에 맞는 모듈을 선택하는 방법으로 재설정한다. Hammerschmidt 등 [6] 은 Tomcat Axis를 수정하여 웹 서비스 뿐만 아니라 인스턴스의 이동 또한 지원하는 웹 서비스 프레임워크를 제안한다. 인스턴스가 보존되므로 클라이언트와의 상호작용이 끝나지 않은 상황에서 웹 서비스 이동이 발생하더라도 웹 서비스 연결을 처음부터 다시 시작할 필요가 없다. Ishikawa 등 [7] 은 모바일 웹 서비스를 정의하여 복합 웹 서비스 실행 상의 웹 서비스 이동방법을 제안하고 있다. 서비스 이동은 웹 서비스가 정의된 모바일 에이전트가 이동하는 방법으로 제공된다. 복합 웹 서비스 안의 개별 웹 서비스의 앤드포인트(end-point)는 웹 서비스 이동에 따라 변할 수 있지만 복합 웹 서비스의 앤드 포인트는 변하지 않는다. Mifsud 등 [8] 의 방법은 웹 서비스가 제공될 수 있는 서버들의 컨텍스트 정보를 원격의 모바일 디바이스에서 감시하고, 어떤 서버에서 제공되는 웹 서비스를 다른 서버

로 이동시킬 수 있는 인터페이스를 제공한다.

웹 서비스를 다루고 있지 않지만 웹 서비스 이동과 유사한 코드 이동성 연구로는 [9-11] 등이 있다. Kern 등 [9] 의 방법은 모바일 에이전트를 구현한 코드의 분할과 배치로 모바일 에이전트의 실행을 빠르게 한다. 제안된 방법이 실행되기 위해서는 주어진 서버에서의 응용 프로그램의 함수 실행 순서가 정해져야 하는 등의 제한 사항이 많다. Paolo 등 [10] 은 같은 지역의 주변 디바이스가 소지하고 있는 라이브러리 파일을 참조하여 필요에 따라 라이브러리 파일을 주변 디바이스에서 다운로드하거나 삭제하여 자원 활용의 효율성을 높인다. Montanari 등 [11] 은 응용 프로그램과 코드 이동 정책을 분리하여 구현할 수 있는 프레임워크를 제안하여 이동 가능한 응용 프로그램의 생산성 향상을 꾀한다.

기존 연구는 주로 데스크탑 환경을 기반으로 하며 서비스 이동이 제한적이거나 서비스 이동 시의 중단 시간이 비교적 길기 때문에 웹 서비스 이동이나 응용 프로그램 이동 중의 서비스 호출에 능동적으로 대처할 수 없었다. 그러므로 본 논문에서는 코드 분할을 통한 웹 서비스 이동 방법을 통하여 서비스 중단 시간을 최소화하고자 한다.

3. 웹 서비스 분할

본 절에서는 모바일 환경에서의 웹 서비스 이동 및 복제를 위하여 웹 서비스의 기능을 구현하고 있는 코드를 분할하는 방법을 기술한다. 제안된 코드 분할 방법과 웹 서비스 이동이 이루어지기 위하여 그림 1의 구조를 가진 프레임워크을 제안한다.

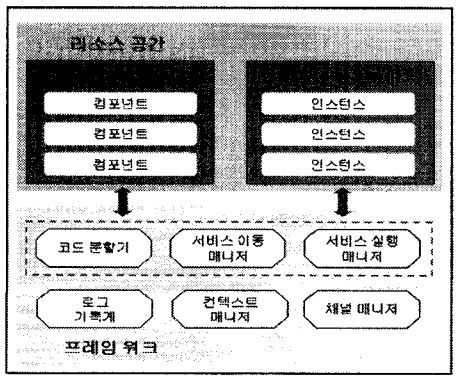


그림 1. 제안된 모바일 웹 서비스 프레임워크

코드 분할기(code splitter)는 구현된 웹 서비스 소스 코드를 분할된 컴포넌트 형태로 코드 저장소(code repository)에 저장한다. 서비스 이동 매니저(migration manager)는 웹 서비스 이동 시에 컴포넌트와 인스턴스를 이동 가능한 형태로 캡슐화(encapsulate)하거나 이를 해제(unpacking)하여 코드 저장소에 컴포넌트를 저장하고 인스턴스를 관리 가능한 형태로 바꾸어 준다. 서비스 실행 매니저(execution manager)는 평상 시에는 웹 서비스의 실행을 담당하고 웹 서비스 이동 시에는 코드 저장소의 웹 서비스를 중단 또는 재구동시킨다. 컨텍스트 매니저(context manager)는 자신과 다른 디바이스의 컨텍스트 정보를 수집하고 웹 서비스의 이동 여부를 결정한다. 채널 매니저(channel manager)는 디바이스 간의 연결을 담당한다. 웹 서비스 요청이나 이동 모두 채널 매니저를 거친다. 로그 기록계(logger)는 프레임워크의 모든 정보를 기록한다.

웹 서비스 코드 분할은 제안된 웹 서비스 이동 방법을 가능케 하는 핵심 기술이며 기본적인 가정은 소스 코드의 크기와 컴파일된 코드의 크기는 비례한다는 것이다. 웹 서비스 코드 분할은 프레임워크의 코드 분할기에서 일어나며 그림 2와 같은 과정을 거친다.

3.1 콜 그래프(call graph) 생성 및 분석

웹 서비스를 구현하고 있는 코드는 함수의 의존 상태를 기준으로 나눈다. 콜 그래프(Call graph)는 그림 3의 중앙과 같다.

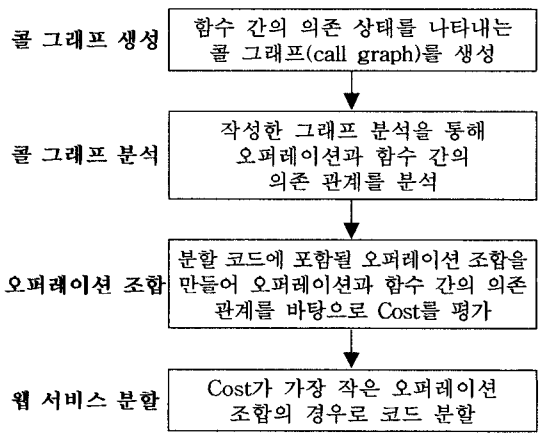


그림 2. 웹 서비스 코드의 분할 과정.

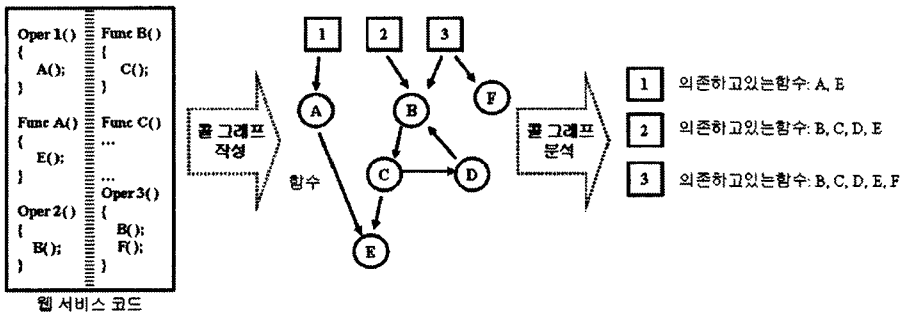


그림 3. 콜 그래프 작성 및 분석의 예

프로그램 코드에서 함수 간의 의존 상태를 기술하고 구성하는 상세한 방법은 [12-14]을 참고한다. 다만, 웹 서비스의 오퍼레이션으로 노출되어 있는 함수는 따로 기록하여 다음 단계인 콜 그래프 분석에 이용할 수 있도록 한다. 콜 그래프를 작성한 후 만들어진 콜 그래프를 분석해서 오퍼레이션이 의존하고 있는 함수를 파악한다. 오퍼레이션 노드에서 참조하고 있는 노드를 따라가는 방식을 취하면서 필요한 정보를 습득한다. 예를 들어, 그림 3은 콜 그래프를 생성 및 분석하여 오퍼레이션이 의존하는 함수를 보여준다.

3.2 오퍼레이션 조합과 웹 서비스 코드 분할

제안된 방법은 코드 분할에 필요한 조건으로서 분할된 코드에 오퍼레이션이 반드시 하나 이상 포함되어 있으며 분할된 코드가 온전한 클래스로 존재한다고 가정한다. 분할된 코드가 온전한 클래스로 존재하기 위해서는 분할된 코드 안에 오퍼레이션이 의존하고 있는 함수가 모두 포함되어야 한다. 그런데 다수의 오퍼레이션이 웹 서비스를 구현하고 있는 함수 중 일부를 공유할 수 있고, 이 경우 함수의 의존성을 기준으로 코드를 분할할 수 없거나 분할이 가능하더라도 비효율적이다. 그림 3의 콜 그래프를 보면 오퍼레이션 1, 2, 3 모두 함수 E를 의존하고 있기 때문에 코드 분할이 불가능하다. 그러나 그림 4를 보면, 함수 노드의 복제로 콜 그래프는 오퍼레이션 1과 오퍼레이션 2, 3의 두 그룹으로 분할된다.

한편 오퍼레이션마다 의존하고 있는 함수의 코드 전부를 복사할 수도 있는데, 복사하는 코드의 양이 많아져서 비효율적일 확률이 크다. 그 예로, 그림 4의 오퍼레이션 2와 3을 분리하려면 함수 코드 B, C, D, E를 복제해야 하고, 각 노드마다 코드의 크기가 비슷

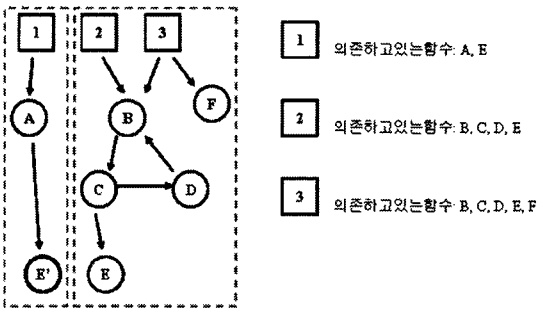


그림 4. 함수 코드 복제의 예.

하다고 가정하면 함수 6개 중 4개를 복사해야 하므로 비효율적이다. 즉, 적당한 함수 코드 복제가 원활한 코드 분할을 가능하게 하는 것이다. 또한 함수 코드 복제의 목적은 코드 분할에 있기 때문에 오퍼레이션이 묶이는 조합으로 어떤 함수가 복제되어야 할지 결정할 수 있다.

최적의 코드 분할을 판단하는 기준은 오퍼레이션의 사용자 선호도와 분할된 코드의 용량과 관련이 깊다. 이동되지 않은 분할 코드가 있다면 이동된 서버에서 오퍼레이션이 실행될 수 없기 때문에 제안된 방법은 이런 경우 오퍼레이션 호출이 있을 때 불만족도가 발생한다고 가정한다. 또한 제안된 방법은 사용자 선호도가 높은 오퍼레이션의 중단 시간을 최소화하기 위해 높은 선호도를 가진 분할 코드를 우선적으로 이동시킨다. 그러므로 어떤 분할 코드에 대한 불만족도가 생길 가능성은 해당 코드보다 선호도가 높은 분할 코드와 해당 코드가 목표 서버로 이동되는 데까지 걸리는 시간과 비례한다. 또한 불만족도가 발생할 확률은 사용자 선호도와도 비례한다. 사용자 선호도는 서비스 제공자가 미리 정의하거나 혹은 단위 시간에 모든 오퍼레이션의 요청 수에 대한 해당 오퍼

레이션의 요청 수의 비로 정의된다. 만약, 불만족도에 대한 가능성을 어떤 수치로 환산할 수 있고 이것을 비용이라고 한다면 웹 서비스 이동 시 모든 비용의 합을 계산하는 식은 다음과 같다.

$$TotalCost = \sum_{i=0}^{n}(p_i \times \sum_{j=0}^{i} S_j) \quad (1)$$

(단, $i<j \rightarrow p_i \geq p_j$, $p_i = p_j \rightarrow S_i \leq S_j$, n은 조합의 개수, p_i $(0 \leq p_i \leq 1)$는 i번째 조합에 포함된 오퍼레이션에 대한 사용자 선호도의 합, S_j $(0 \leq S_j \leq 1)$는 j번째 조합에 해당하는 전체 코드 용량)

최적의 분할 조합은 TotalCost가 가장 적은 조합을 말한다. 모든 조합을 검색하여 최적의 분할 조합을 찾았다면, 다음은 주어진 조합에 따라 코드를 분할하는 것이다. 해당 오퍼레이션과 그 오퍼레이션이 의존하는 함수들을 묶어 소스 코드들을 작성하는 것으로 웹 서비스 분할은 이루어지고 분할된 코드들이 컴파일되어 제안된 프레임워크의 리소스 공간에서 컴포넌트로 저장된다.

4. 웹 서비스 이동 및 복제

제안된 웹 서비스 이동 방법은 컴포넌트 이동과 복제의 두 가지로 나뉜다. 디바이스의 남은 배터리 수준이나 위치 변경 등으로 원래 서버에서 웹 서비스 제공이 불가능할 경우에는 다른 디바이스에 모든 컴포넌트를 이동시켜야 한다. 또한 네트워크 대역폭이 기준에 미치지 못하거나 특정 지역의 특정 웹 서비스 사용이 급증할 경우 다른 디바이스로 컴포넌트를 복제해야 한다. 웹 서비스 오퍼레이션의 호출이 있을 때 해당 오퍼레이션에 관련된 컴포넌트만 복제하여 빠른 실행을 도모할 수도 있다. 이동이 이루어질 경우 웹 서비스의 앤드포인트는 새로운 디바이스로 바뀌고, 복제가 이루어질 경우 사용가능한 앤드포인트가 추가된다. 컨텍스트 매니저에서 수집한 컨텍스트 정보가 웹 서비스의 이동과 복제가 동시에 발생해야 될 경우 웹 서비스 이동을 우선한다. 원래 서버에서 웹 서비스 제공이 불가능해짐에 따라 이동을 우선시해야 하고 이동된 서버의 컨텍스트를 고려하여 복제를 결정하고 복제된 서버들을 관리해야 한다.

4.1 웹 서비스 이동

배터리 수준이 기준에 미치지 못하거나 해당 디바이스의 위치가 웹 서비스를 제공할 수 없을 경우 웹 서비스는 반드시 이동되어야 한다. 그림 5는 웹 서비스 이동을 나타내는 개략도이다.

웹 서비스가 이동될 목표 서버에서 웹 서비스 이동 요청을 수락하게 되면, 전송 가능한 형태로 바꾼 인스턴스와 미리 분할해 둔 컴포넌트 형태의 바이너리 코드가 전송된다. 이들의 전송 순서는 다음과 같다. 서비스 실행 매니저의 실행 대기열에 저장된 오퍼레이션 호출 요청은 원래 서버와 새로운 서버 중 더 빠른 실행이 가능한 쪽을 선택하여 우선 실행시킨다. 기본적인 방법은 웹 서비스 이동 중의 웹 서비스가 호출될 경우와 같다. 이와 병행하여 리소스 공간에 존재하는 인스턴스들을 전송하고, 인스턴스와 관련된 컴포넌트를 사용자 선호도 순으로 전송한 다음, 나머지 컴포넌트도 마찬가지로 선호도 순으로 전송한다. 목표 서버의 서비스 실행 매니저는 각 인스턴스와 컴포넌트

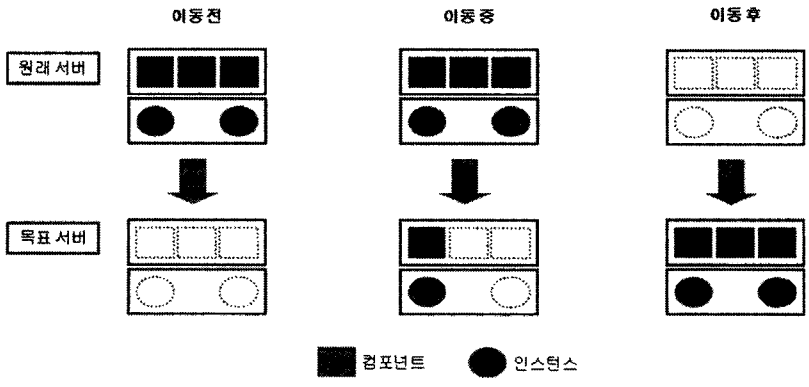


그림 5. 웹 서비스 이동의 예

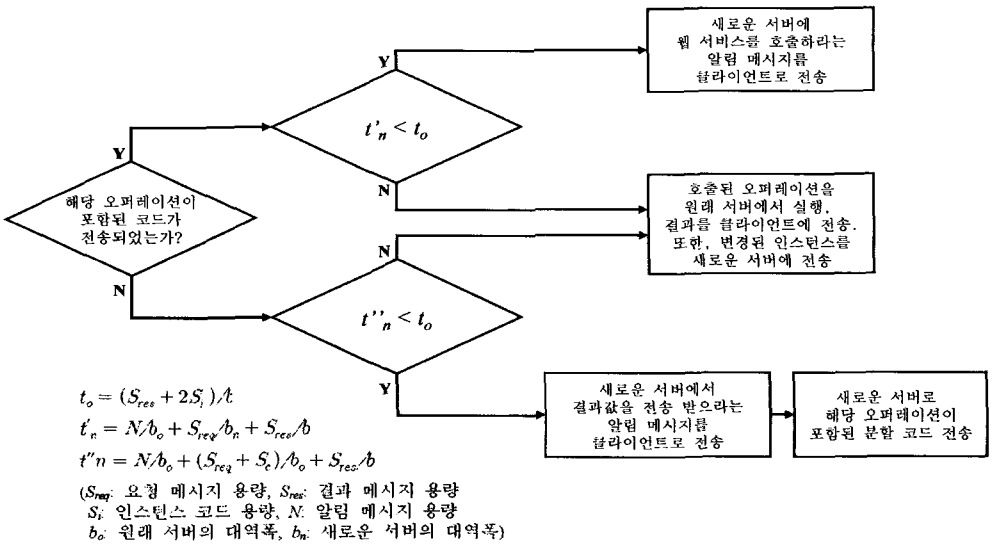


그림 6. 웹 서비스 이동 중에 호출이 발생할 경우의 순서도

가 전송될 때마다 해당 오퍼레이션을 사용가능하도록 구동하고 모든 인스턴스와 컴포넌트의 전송을 마치면 기존 서버의 웹 서비스는 삭제된다.

웹 서비스 이동 중에 웹 서비스 호출을 받게 되면, 전술한 바와 같이 기존 서버와 새로운 서버에서의 웹 서비스 실행 시간을 비교하여 웹 서비스 실행 방법을 결정한다. 그림 6은 웹 서비스 이동 중에 호출에 대처하는 순서도이다.

t_o는, 원래 서버에서 해당 오퍼레이션을 실행하였을 경우, 클라이언트에게 결과값을 전송하는 시간과, 새로운 서버에서 인스턴스를 전달받고 실행한 후 다시 새로운 서버로 전송하는 시간의 합이다. t'_n은, 해당 오퍼레이션이 포함된 분할 코드가 새로운 서버로 이미 전송되었고 클라이언트가 새로운 서버로 웹 서비스를 재호출했을 경우 클라이언트가 결과값을 반환받는 시간을 말한다. 웹 서비스를 새로운 서버로 재호출하라는 알림 메시지가 전달되는 시간과, 새로운 서버와 클라이언트 간의 주고받는 메시지 전달 시간의 합이다. t''_n은 해당 오퍼레이션이 포함된 분할 코드가 아직 전송되지 않았지만 해당 분할 코드가 새로운 서버로 이동된 후 그 오퍼레이션이 실행되는 경우 클라이언트가 결과값을 받기까지의 시간이다.

그림 7의 (a)는 오퍼레이션이 포함된 컴포넌트가 전송되어 있을 때 웹 서비스가 호출된 경우, 클라이언트와 서버 간의 호출 과정을 보여주며 그림 6의

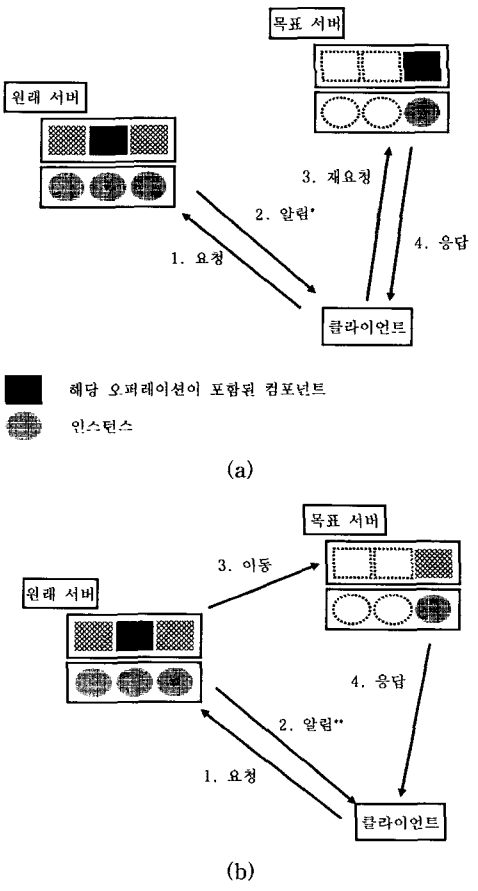


그림 7. 웹 서비스 이동 중의 웹 서비스 호출

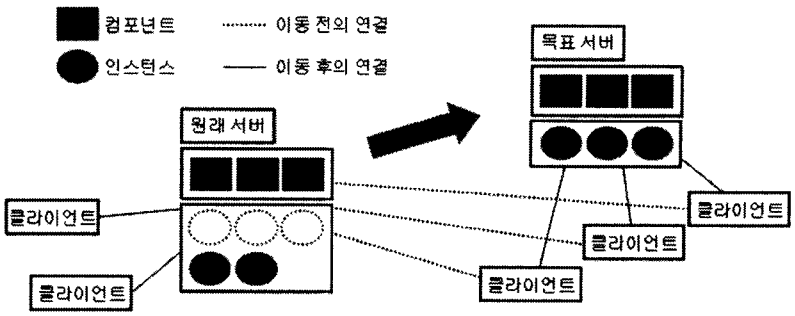


그림 8. 웹 서비스 복제의 예

순서도에서 제일 윗 부분에 해당한다. 원래 서버에 웹 서비스 요청을 받았을 때 목표 서버에 재요청하는 것이 빠르면 클라이언트에 알림(notification) 메시지를 전송하여 목표 서버에 서비스 재요청을 유도한다. 그림 7의 (b)는 그림 6의 제일 아래쪽에 해당한다. 오퍼레이션이 포함된 컴포넌트가 아직 전송되지 않는 경우이므로 목표 서버에 해당 컴포넌트가 전송되어 실행될 수 있을 때까지 대기해야 한다.

4.2 웹 서비스 복제

웹 서비스 복제는 하나의 웹 서비스에 관련된 모든 컴포넌트의 복제와 그 웹 서비스에 대응되는 인스턴스가 이동되는 것, 웹 서비스의 부분을 이루는 일부 컴포넌트의 복제와 인스턴스가 이동되는 것으로 구분되는데, 후자의 경우 클라이언트부터 웹 서비스에 대한 요청이 있을 때만 발생한다.

4.2.1 다른 모바일 서버로의 웹 서비스 복제

이동하게 될 새로운 목표 서버에서 웹 서비스 복제를 수락하게 되면 컴포넌트가 전송되며 웹 서비스 이동과 유사하다. 다른 점은 모든 인스턴스가 이동할 필요가 없고, 인스턴스와 컴포넌트의 전송이 완료된 후, 목표 서버로 복제된 인스턴스는 기존 서버에서 삭제되고 기존 서버에 있는 컴포넌트는 삭제되지 않는다는 점이다. 어디로 복제됐는지 원래 서버에서 관리하고 원래 서버는 목표 서버에 웹 서비스 삭제 요청을 할 수 있다. 웹 서비스 복제 중의 웹 서비스 호출이 발생하면 웹 서비스 이동 중의 웹 서비스 호출과 마찬가지로 처리한다.

그림 8은 목표 서버에 웹 서비스와 세 개의 인스턴스를 복제하여 트래픽을 감소시키는 웹 서비스 복제의 예이다. 컨텍스트 매니저에서 서버를 검색하여 해당하는 컴포넌트와 인스턴스를 복제하고 클라이언트 간의 연결을 재설정한다.

4.2.2 클라이언트로의 컴포넌트 및 인스턴스 복제

본 절은 비교적 용량이 작은 웹 서비스를 클라이언트로 복제하여 실행하는 방법에 대해 설명한다.

그림 9는 클라이언트로의 컴포넌트 및 인스턴스 복제를 나타낸 개략도이다. 클라이언트는 웹 서비스를 호출할 때 분할 코드들의 용량이 기술된 목록을 미리 다운로드 받고, 오퍼레이션을 호출할 때 쓰이는 파라미터의 용량과 해당 오퍼레이션이 포함된 컴포넌트의 용량을 비교하여 웹 서비스 이동을 결정한다. 컴포넌트가 복제되면 서버는 복제된 서비스 목록을 관리하고 클라이언트에게 복제된 컴포넌트의 삭제를 요청할 수 있다.

다른 서버로 웹 서비스 이동이나 복제가 이루어지는 동안에 클라이언트로의 복제 요청이 발생하는 경우 해당 컴포넌트와 인스턴스를 빠르게 다운로드 받

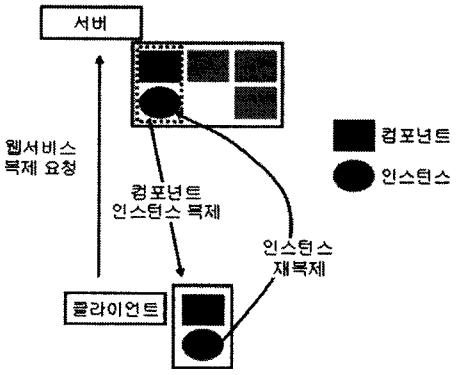
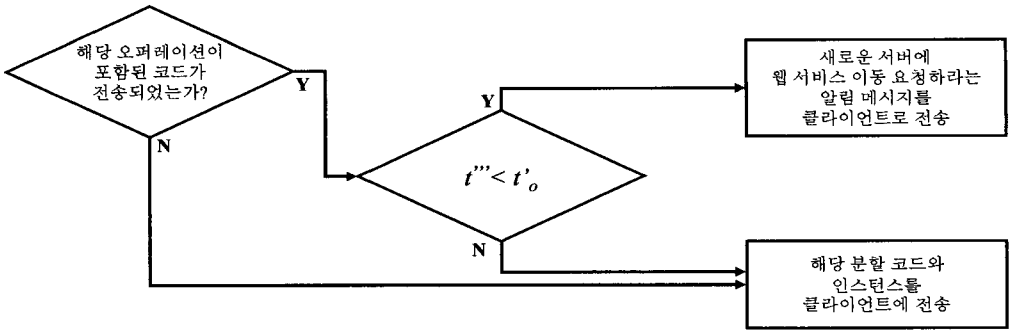


그림 9. 클라이언트로의 컴포넌트 복제



$$t'_o = (S_c + S_i)/b_o$$

$$t'''_n = N/b_o + S_{req}/b_n + (S_c + S_i)/b_n$$

(S_c: 분할된 코드의 용량, S_i: 인스턴스의 코드 용량,
N: 알림 메시지의 용량, S_{req}: 웹 서비스 이동 요청 메시지
b_o: 원래 서버의 대역폭, b_n: 새로운 서버의 대역폭)

그림 10. 클라이언트로의 컴포넌트 및 인스턴스 복제

을 수 있는 서버를 선택하게 되는데 그 과정은 그림 10과 같다. t_o'는 원래 서버에서 클라이언트로 해당 컴포넌트와 인스턴스를 이동시키는데 걸리는 시간이고, t'''_n은 새로운 서버에서 이동시키는데 걸리는 시간과 원래 서버에서 알림 메시지를 클라이언트로 전달하는데 걸리는 시간을 합한 것이다. 새로운 서버에 해당 컴포넌트가 있고, 새로운 서버에서 클라이언트로 해당 파일을 전송하는 것이 빠르면, 새로운 서버에서 클라이언트로 웹 서비스 복제를 수행한다. 그 외의 경우 원래 서버에서 복제를 수행한다.

5. 실험 결과

본 절에서는 제안된 방법의 성능을 평가하기 위해 여러 가지 기준으로 수행한 코드 분할의 실험 결과를 기술하고 제안된 프레임워크의 실행을 세 가지 시나리오를 통해 검증한다. 실험은 Intel(R) PXA270, 62.28M 메인 메모리를 가진 Windows Mobile 2003 환경에서 수행되었다.

5.1 코드 분할

본 절은 코드 분할의 성능을 평가하기 위한 실험 환경과 성능 분석에 대해 기술한다. 실험 환경에서는 실험에 사용된 데이터와 기준을 기술하고, 성능 분석에서는 수행된 실험의 결과를 기술하고 분석한다.

5.1.1 실험 환경

코드 분할의 성능 분석을 위하여 세 가지 기준에서 수행한 실험을 통해 코드를 분할하는데 걸리는 시간을 측정하였다. 첫 번째는 웹 서비스에서 제공하는 오퍼레이션 간의 의존도와 관련된 실험으로, 10개의 오퍼레이션을 제공하고 각 오퍼레이션은 3개의 함수를 호출하는 웹 서비스 코드를 대상으로 하였다. 두 번째는 오퍼레이션에서 호출하고 있는 함수의 수와 관련된 실험으로, 실험 데이터는 10개의 오퍼레이션과 20개의 함수로 구성된 웹 서비스로서 각 오퍼레이션이 호출하고 있는 함수의 수를 2에서 20까지 조정하여 실험을 수행하였다. 세 번째는 웹 서비스를 구성하는 오퍼레이션의 수이다. 10개의 함수로 이루어져 있고 각 오퍼레이션은 10개의 함수를 호출하고 있는 웹 서비스를 사용했으며, 오퍼레이션의 수를 1부터 10까지 조정하여 수행하였다.

실험에 사용된 웹 서비스에 포함된 오퍼레이션과 함수의 의존성은 임의로 작성되었으며, 각 기준의 실험은 오퍼레이션에서 호출하고 있는 함수의 크기를 1, 5, 10, 50, 100Kbyte의 다섯가지로 조정하여 수행하였다.

5.1.2 오퍼레이션 간의 의존도에 따른 성능 평가

오퍼레이션의 의존도는 자신이 호출하고 있는 함수를 공유하는 오퍼레이션들의 수를 백분율로 표시한 것으로 식은 (2)와 같다.

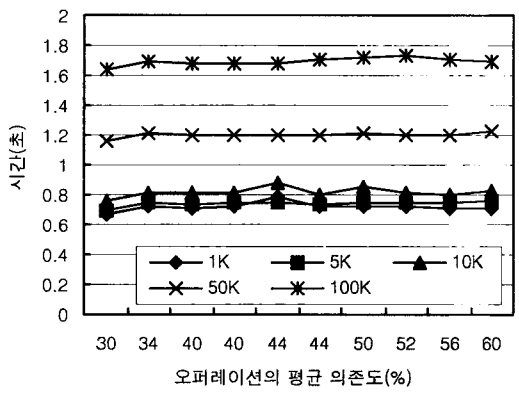
$$\text{오퍼레이션의 의존도} = \frac{\text{함수를 공유하는 오퍼레이션수}}{\text{전체 오퍼레이션수}} \quad (2)$$

그림 11(a)는 웹 서비스에 포함된 오퍼레이션의 평균 의존도와 코드 분할에 걸리는 시간의 관련성에 대한 그래프이다. 해당 그래프의 결과는 오퍼레이션의 평균 의존도와 코드 분할에 걸리는 시간은 거의 관련이 없다는 것을 보이고 있다. 공통으로 호출하고 있는 함수가 많으면 분할될 확률이 적고, 적으면 분할될 확률이 크다. 코드가 분할될 때 공통으로 호출하고 있는 함수가 복제되기 때문에 복제되는 함수의 수는 제한된 범위 내에 있다. 또한, 제한된 횟수의 함수 복제는 코드 분할 시 제한된 시간의 지연만을 가져온다. 따라서 오퍼레이션의 평균 의존도와 코드 분할에 걸리는 시간은 관련이 없다.

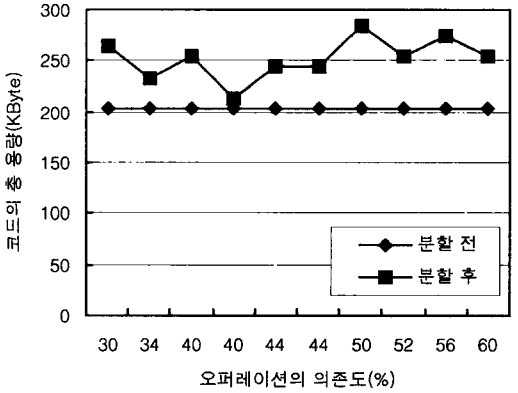
그림 11(b)는 역시 오퍼레이션의 평균 의존도와 분할된 코드의 전체 크기와 상관없다는 것을 보여주고 있다. 분할된 코드의 전체 크기가 증가하는 것은 다른 요인에서 찾는 것이 타당하다. 그래프에서 해당하는 웹 서비스는 각 오퍼레이션과 함수 코드 크기가 10Kbyte 정도되는 것을 대상으로 한 것이다.

5.1.3 오퍼레이션에서 호출하는 함수의 수에 따른 성능 평가

그림 12(a)는 오퍼레이션에서 호출한 함수의 수와 코드 분할 시간의 관계를 보여준다. 호출한 함수의 수와 코드 분할 시간이 비례하는 것을 볼 수 있는데, 콜 그래프를 만들 때, 호출하는 함수의 수에 따라 노드와 노드 간의 연결이 늘어나면 각 노드에 대응되는

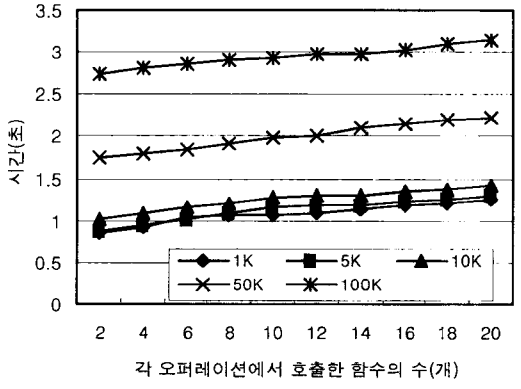


(a) 코드 분할에 걸리는 시간

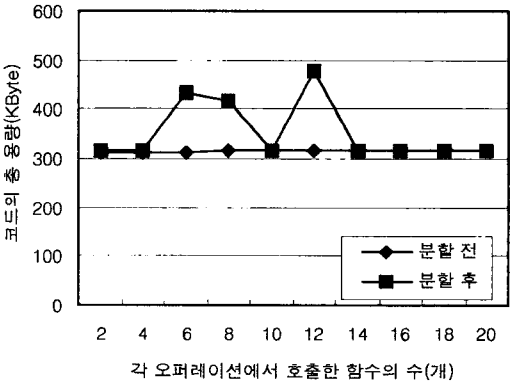


(b) 분할된 코드의 크기(10Kbyte)

그림 11. 의존도에 따른 실험 결과



(a) 코드 분할에 걸리는 시간



(b) 분할된 코드의 크기(10Kbyte)

그림 12. 오퍼레이션에서 호출한 함수의 수에 따른 실험 결과

연결의 수가 많아지게 되므로 콜 그래프를 분석할 때 더 많은 시간이 소요된다.

그림 12(b)는 오퍼레이션에서 호출한 함수의 수와 분할된 코드 크기의 관련성을 보여준다. 해당 그래프의 데이터는 각 오퍼레이션과 함수의 크기가 10Kbyte인 것이다. 호출한 함수의 수가 처음의 2와 4일 경우 사용된 실험 데이터에서 각 오퍼레이션이 호출한 공통된 함수가 없기 때문에 분할 전과 분할 후의 코드 크기가 같다. 호출한 함수를 계속 증가시키면 호출한 함수의 수가 12개가 될 때 까지 코드 크기와의 규칙성을 발견할 수 없다. 호출한 함수가 14개 이상일 경우는 오퍼레이션이 전체 함수의 70% 이상을 호출하고 있다. 이 경우 제안된 방법을 사용하면 각 오퍼레이션이 공통으로 호출하고 있는 함수를 복제하는 것보다 분할하지 않는 편이 낫다는 것을 의미한다.

5.1.4 오퍼레이션 수에 따른 성능 평가

그림 13은 오퍼레이션 수에 따른 코드 분할 시간을 나타낸다. 웹 서비스에서 사용한 오퍼레이션의 수가 증가할 수록 코드 분할 시간이 급격히 상승됨을 알 수 있다. 최소 비용의 분할 코드를 구하기 위해 모든 오퍼레이션의 조합을 계산하기 때문이다.

실험 결과에서 공통적으로 발견된 점은 코드 분할 시간이 웹 서비스의 크기에 비례한 것이다. 오퍼레이션 간의 의존도는 코드 분할 시간과 거의 관계가 없으며, 웹 서비스의 오퍼레이션에서 호출하는 함수의 수와 오퍼레이션 수가 코드 분할 시간에 영향을 미친다. 특히 오퍼레이션 수의 증가에 따라 분할 시간이 지수 승으로 증가하는데, 향후 연구에서 개선해야할 부분이다.

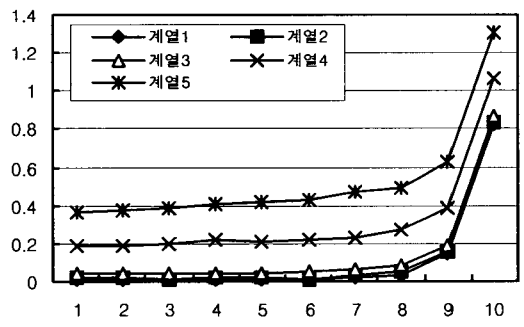


그림 13. 오퍼레이션의 수에 따른 코드 분할 시간.

분할된 코드의 크기는 오퍼레이션에서 호출한 함수의 수에 영향을 받는다. 분할 전과 분할 후의 코드 크기가 거의 같은 경우가 크게 두가지 경우에 발생할 수 있다. 오퍼레이션에서 호출한 함수의 수가 작은 경우, 각 오퍼레이션이 공통으로 호출하는 함수가 없거나 수가 작기 때문에 복제되는 함수가 없다. 또한 호출한 함수의 수가 많은 경우 공통으로 호출하는 함수가 함수를 복제하여 코드를 분할하는 것이 오히려 더 소모적일 수 있다.

5.2 이동 및 복제

본 절은 세 가지 시나리오를 통해 제안된 프레임워크를 검증한다. 제안된 프레임워크를 이용하여 웹 서비스 이동, 웹 서비스 복제, 그리고 클라이언트로의 컴포넌트 및 인스턴스 복제에 적용되는 세 가지의 시나리오를 구현하였다.

5.2.1 웹 서비스의 이동

본 절은 웹 서비스 이동이 이루어지는 시나리오 구현을 기술한 것으로서 교통 환경 정보를 제공하는 웹 서비스가, 이동하는 자동차 사이에서 이동하는 것으로 가정한다. 그림 14는 시나리오를 간략히 나타낸다. 이 시나리오는 특정 교량에 관한 교통 정보를 제공하는 자동차가 교량을 이탈하면서 새로 교량에 진입하는 차량에 웹 서비스를 이동시켜 적합한 서비스를 제공하는 상황이다. 이탈하는 차량에 연결하고 있던 웹 서비스 사용자는 웹 서비스 이동이 이루어진 새로운 차량에 연결을 재설정한다.

그림 15를 보면 교량을 이탈하기 전(좌측)의 차량에 평균 속도(getAveVel)와 기온(getTemperature)에 대한 정보를 요구하는 웹 서비스가 호출되었고

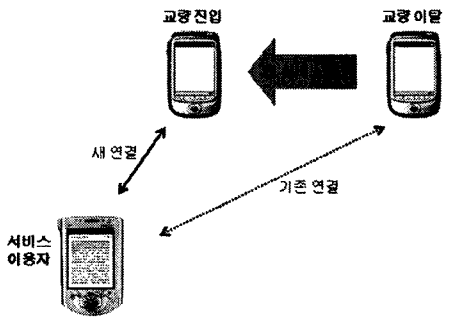


그림 14. 교통 정보 서비스의 이동 예

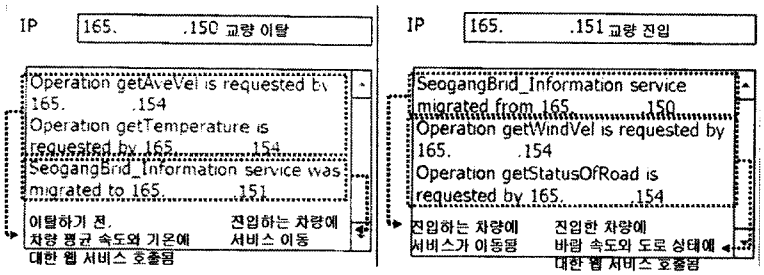


그림 15. 교통 정보 서비스를 제공하는 디바이스의 로그 정보

교량에 진입하는 차량에 웹 서비스를 이동시켰다. 웹 서비스가 이동되었던 차량의 로그 정보(우측)에는 교량을 벗어나는 차량에서 웹 서비스가 이동되어 왔다는 정보가 기술되어 있고, 웹 서비스 이동 후 바람 속도와 도로 상태 등을 제공하는 웹 서비스가 요청된 것을 볼 수 있다.

5.2.2 다른 모바일 서버로의 웹 서비스 복제

본 절은 미술관에서 제공하는 작품 정보 제공 웹 서비스를 이용한 웹 서비스 복제 시나리오를 기술한다. 미술관은 작품 정보를 제공하는 웹 서비스를 운영하고 있으며 필요에 따라 미술관을 방문하는 관광객들을 위한 가이드의 PDA에 웹 서비스를 복제하여 트래픽 분산을 유도한다. 그림 16은 시나리오의 개략도이다. 먼저 관광객은 미술관에서 운영하고 있는 기기에 연결되어 있고, 미술관에서 가이드의 PDA에 웹 서비스 복제를 요청한다. 가이드의 PDA에서 복제를 수락하게 되면 작품 정보 웹 서비스가 복제되고 관광객은 자동으로 가이드의 PDA에 새로 연결을 설정하여 끊김없는 서비스를 제공받는다.

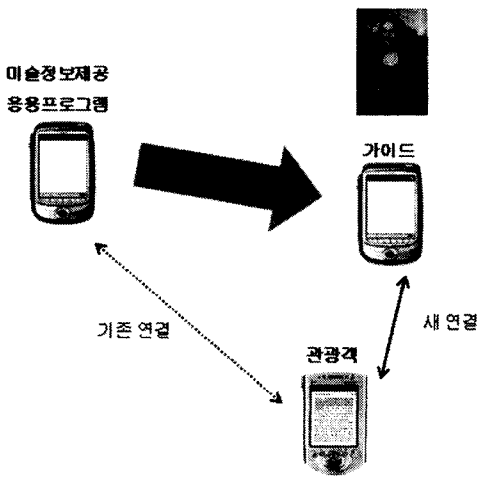


그림 16. 작품 정보 서비스의 복제

그림 17의 (a)와 (c)는 웹 서비스가 복제되기 전에 관광객의 PDA에서 웹 서비스를 호출하여 결과를 얻는 모습이다. (b)와 (d)는 웹 서비스가 복제된 후의 모습이다. 어떤 작품을 검색했을 때 인스턴스가 보존되어 파라미터없이 웹 서비스를 호출하면 그 전에 선택했던 작품과 관련된 정보를 반환한다. 작품의 제작 연도에 대한 정보를 요청(getYear)하면 해당 작품 요청에 대한 인스턴스가 저장되어 파라미터 없이 작품의 상세와 제작자를 요청하는 오퍼레이션(getDescriptionOfArtWithoutParam, getArtistWithoutParam)을 호출할 수 있다. 웹 서비스가 복제될 때 원래 서버의 인스턴스 역시 가이드의 PDA에 복제되어 파라미터 없는 오퍼레이션(getBornYear, getDeadYear, getDescriptionOfArtistWithoutParam)이 호출되어도 관련된 정보를 반환시킨다.

그림 18은 복제된 웹 서비스를 원래 서버에서 관리하고 있고 가이드의 PDA에 삭제 요청을 하여 복제된 웹 서비스를 삭제하는 것을 보여주고 있다. 그림 18의 (a)와 (b)는 원래 서버의, (c)는 목표 서버의 로그 정보를 나타낸다. (a)에서는 목표 서버로 웹 서비스를 복제하였다는 정보와 서비스가 복제된 목표 서버의 주소와 서비스 이름의 목록이 나타나 있다. (b)에서는 목표 서버로 복제한 웹 서비스의 삭제를 요청했다는 정보가 기술되어 있고 삭제된 후 복제된 서비스의 목록이 없다는 것을 확인할 수 있다. (c)에서는 제작자의 상세 정보를 요청하는 오퍼레이션(getDescriptionOfArtistWithoutParam)이 호출된 후 원래 서버의 요청에 의해 해당 웹 서비스가 삭제되었다는 로그 정보가 기술되어 있다.

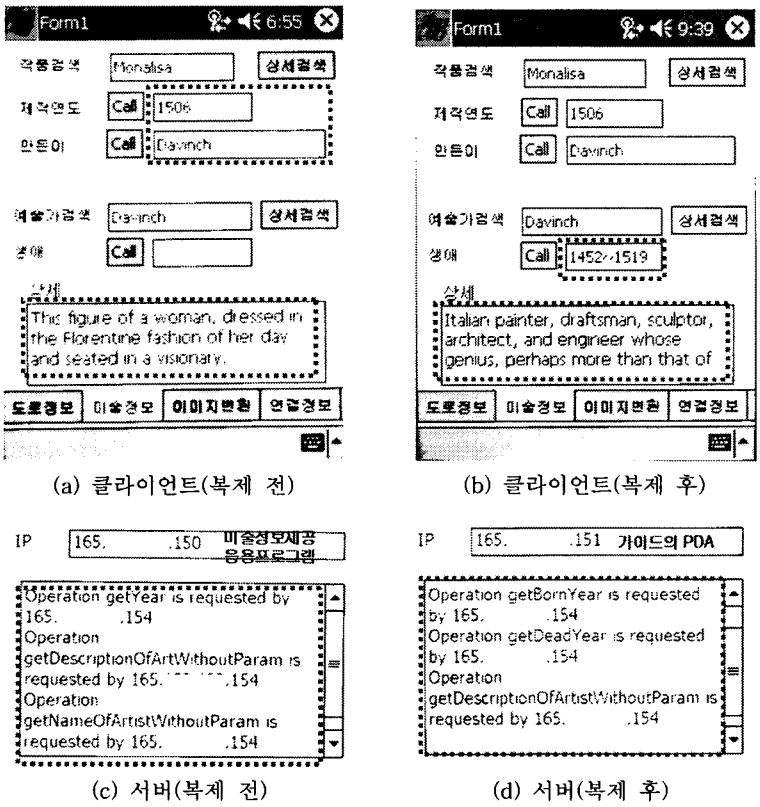


그림 17. 클라이언트와 서버의 스크린샷

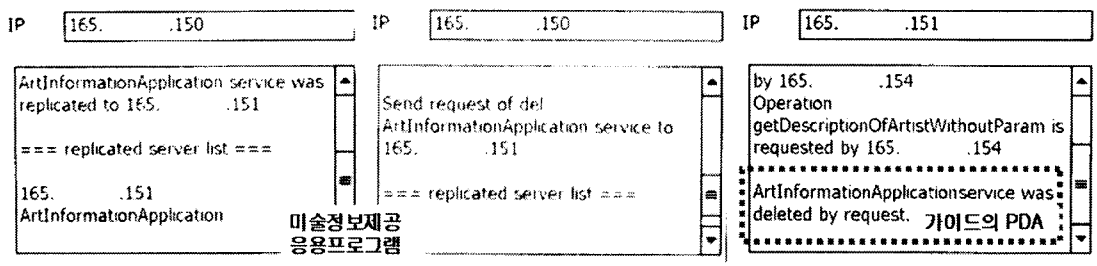


그림 18. 원래 서버에서 목표 서버로의 삭제 요청에 대한 스크린샷

5.2.3 클라이언트로의 컴포넌트 복제

웹 서비스를 호출할 때 용량이 큰 파라미터를 전송하는 것보다 클라이언트로 컴포넌트를 복제하여 효율적으로 웹 서비스를 이용할 수 있는 경우가 발생한다. 그림 19는 그 예를 간략히 표현한 그림이다. 해당 시나리오에서 서버는 이미지를 변환하는 서비스를 제공하고 클라이언트는 이미지의 확대 및 축소의 기능을 가진 오퍼레이션을 요청한다. 클라이언트는 용량이 큰 이미지를 파라미터로 보내지 않고 상대적으로 용량이 작은 컴포넌트를 자신의 기기에 복제하여 원하는 오퍼레이션을 실행한다.

그림 20의 (a)는 이미지 변환 웹 서비스를 제공하는 원래 서버의, 그리고 (b)는 이미지 확대 및 축소 오퍼레이션을 이용하는 클라이언트의 로그 정보이

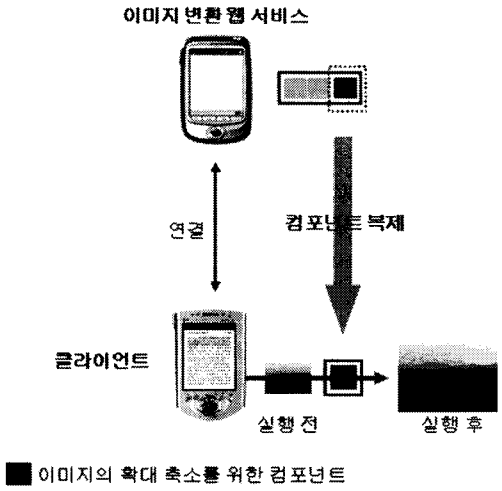
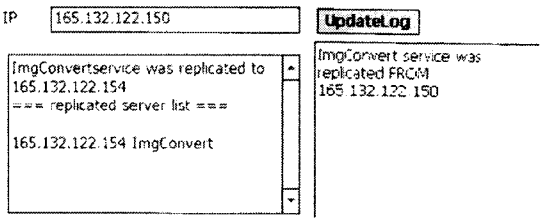


그림 19. 이미지 변환 서비스의 컴포넌트 일부 복제



(a) 서버의 로그 정보 (b) 클라이언트의 로그 정보

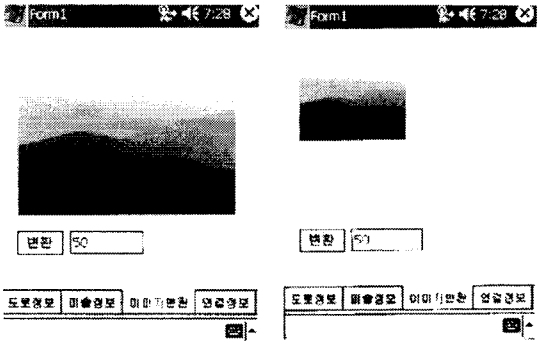
그림 20. 컴포넌트의 복제

다. 원래 서버의 로그 정보에서는 클라이언트의 주소와 복제된 웹 서비스의 목록을 볼 수 있으며, 우측의 클라이언트에서는 웹 서비스가 복제되었다는 로그 정보를 확인할 수 있다. 복제된 컴포넌트는 이미지를 축소하는데 쓰이는데, 그림 21이 그것을 나타낸다.

그림 22는 복제된 컴포넌트가 관리되었고 삭제 요청을 하여 클라이언트에서 해당 컴포넌트가 삭제된 것을 보여준다. (a)는 원래 서버의, 그리고 (b)는 클라이언트의 로그 정보를 뜻한다. 원래 서버의 로그 정보에서는 클라이언트로 이미지 변환 웹 서비스의 삭제 요청을 한 것을 기술하였고 우측의 클라이언트에서는 복제된 웹 서비스가 원래 서버의 요청에 의해 지워졌다는 정보를 기술하고 있다.

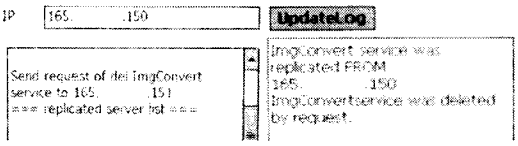
5.3 기존 연구와의 비교

본 절은 실험을 통해 보인 점을 기존 연구와 비교하여 분석한다. Pratitha 등이 제안한 방법의 핵심은



(a) 복제된 컴포넌트 활용 전 (b) 복제된 컴포넌트 활용 후

그림 21. 복제된 컴포넌트를 사용한 이미지 축소



(a) 서버의 로그 정보 (b) 클라이언트의 로그 정보

그림 22. 복제된 컴포넌트 삭제

Supporting Module을 통하여 디바이스의 컨텍스트에 맞게 웹 서비스를 변형시킬 수 있다는 것이다. 하지만 변형시킬 Supporting Module이 많아지면 웹 서비스의 크기가 커져 이동시키는데 많은 자원이 소모된다. 본 논문은 웹 서비스 코드를 쪼개어 선호도가 높은 조각을 먼저 보내는 방법을 선택하여 오히려 선호도가 높은 오퍼레이션을 더 빨리 이용할 수 있다. 또한 Pratitha 등이 지원하지 못하는 인스턴스 이동 및 복제를 지원한다.

웹 서비스 인스턴스의 이동을 지원하는 것은 Hammerschmidt 등의 방법이 있는데 웹 서비스와 인스턴스의 이동을 통해 원래 서버의 자원 소모의 집중을 방지한다. 이에 반하여 제안된 방법은 분할된 조각을 이동시켜 특히 클라이언트로의 컴포넌트 복제의 시나리오에서 더 좋은 성능을 보인다. 한편, Ishikawa 등은 개별 웹 서비스를 포함한 모바일 에이전트가 호스트를 이동하며 복합 웹 서비스를 실행하는 방법으로서 제안된 방법과는 달리 모바일 환경을 대상으로 하지 않는다.

코드 이동성의 측면에서 관련되는 연구는 Kern 등, Paolo 등, Montanari 등의 방법이 있다. Kern 등의 방법은 여러 서버를 거쳐 에이전트가 얻을 결과를 빨리 얻고자 하는 것이기 때문에 비교 대상이 되지

표 1. 제안된 방법과 기존 연구의 비교

	웹 서비스 대상 여부	인스턴스 보존 여부	재설정 여부	모바일 환경 대상 여부	Context -aware 여부
Pratitha 등	○	×	○	×	○
Hammerschmidt 등	○	○	×	×	×
Ishikawa 등	○	×	×	×	×
Mifsud 등	○	×	×	×	×
Kern 등	×	○	×	×	×
Paolo 등	×	×	×	○	○
Montanari 등	×	×	×	×	○
제안된 방법	○	○	×	○	○

못한다. Paolo 등과 Montanari 등의 방법은 바이너리 코드의 이동은 지원하나 인스턴스의 이동은 지원되지 않는다.

제안된 방법은 모바일 환경에서의 웹 서비스 이동 및 복제를 지원한다. 웹 서비스 이동 및 복제 시 인스턴스 역시 보존되며 모바일 디바이스의 컨텍스트에 대응하여 웹 서비스 이동 및 복제가 결정된다. 웹 서비스 이동 후 재설정 기능은 포함되지 않지만 재설정 기능은 필연적으로 각기 다른 컨텍스트 상황에 대응되는 코드를 모두 만들어야 하기 때문에 코드의 크기가 커져 모바일 환경에 맞지 않다. 표 1은 제안된 방법과 기존 연구의 차이를 간략히 기술한다.

6. 결론 및 향후 연구 방향

본 논문에서는 모바일 디바이스에서 웹 서비스의 효과적인 이동 방법을 제안하였다. 전통적인 클라이언트-서버 모델은 웹 서비스의 형태를 어느정도 강제할 수 밖에 없기 때문에 무선 모바일 환경에 적합하지 않다. 기존의 모델에서 탈피한 이동 가능한 웹 서비스를 제안함으로써 무선 모바일 환경에서의 좀 더 자유로운 웹 서비스 활용을 가능케 한다.

또한 제안된 방법은 강제된 환경에서 웹 서비스를 구현하는 것이 아닌 구현되어 있는 웹 서비스 코드를 잘게 쪼개어 웹 서비스 이동이 무선 모바일 환경에서 수월하고 효율적으로 운용될 수 있도록 하였다. 선호도가 높은 오퍼레이션을 유사시 다른 모바일 디바이스로 보다 빨리 이동시켜 서비스 중단 시간을 최소화 시킬 수 있는 효과를 갖는다.

제안된 코드 분할 방법의 성능을 분석하기 위해 세 가지 기준으로 실험한 결과 웹 서비스 코드의 크기, 오퍼레이션에서 호출한 함수의 수에 비례하여 분할 시간이 증가하였고, 특히 오퍼레이션의 수가 증가할 수록 분할 시간이 지수 승으로 증가함을 확인할 수 있었다. 분할된 코드의 용량은 오퍼레이션과 함수의 구현 방식에 따라서 변화가 크기 때문에 규칙성을 찾기 어렵지만 오퍼레이션에서 호출하는 함수의 수가 작거나 매우 많으면 분할된 코드는 원본 웹 서비스 코드와 크기가 비슷하다.

또한 제안된 프레임워크를 활용한 세 가지 시나리오를 통해 제안된 웹 서비스 이동 방법을 검증하였다. 웹 서비스 제공이 불가능해질 때는 웹 서비스 이동으로, 트래픽 분산이 요구될 때는 웹 서비스 복제로 끊임없는 서비스 제공을 도모한다. 그리고 크기가 큰 파라미터의 전송을 필요로 할 경우, 클라이언트로 컴포넌트를 복제하여 자원의 효율적 활용이 가능함을 확인하였다.

제안된 방법은 사용자 선호도에 따라 웹 서비스를 구현하고 있는 코드를 분할하여 컴포넌트 형태로 컴파일해야 하는 형태를 띄고 있고, 컴포넌트는 분리나 병합이 불가능하다. 그러므로 선호도가 크게 변하는 환경에서는 적절치 못하다. 이 한계를 극복하기 위하여 원시 코드의 분할보다는 바이너리 코드의 동적인 분할이나 병합 기술이 필요하다. 또한 무선 모바일 환경에서 웹 서비스 이동 및 복제를 위한 정교한 컨텍스트 모델을 정의하고, 동적으로 컴포넌트를 분할 및 병합할 수 있는 기술을 연구할 계획이다.

참 고 문 헌

[1] World Wide Web Consortium, Web Services, http://www.w3.org/2002/ws

[2] World Wide Web Consortium, SOAP, http://ww.w3.org/TR/soap12-part1/

[3] S. N. Sirirama, M. Jarke, and W. Prinz, "Mobile Web Services Provisioning," Proc. Int'l Conf. Internet and Web Application and Services/Advanced & Int'l Conf. Telecommunication(AICT/ICIW 2006), pp. 120-126, 2006.

[4] D. Schall, M. Aiello, and S. Dustdar, "Web

Services on Embedded Devices," *Web Information System*, Vol.2, No.1, pp. 1-6, 2006.
[5] I. M. Pratistha and A. B.Zaslavsky, "Fluid: supporting a transportable and adaptive web service," Proc. ACM Symp. Applied Computing, pp. 1600-1606, 2004.
[6] B. C. Hammerschmidt and V. Linnemann, "Migrating Stateful Web Services using Apache AXIS and P2P," Proc. the IADIS Int'l Conf. Applied Computing, pp. 433-441, 2005.
[7] F. Ishikawa, N. Yoshioka, Y. Tahara, and S. Honiden, "Toward Synthesis of Web Services and Mobile Agents," Proc. AAMAS Workshop on Web Services and Agent-based Engineering(WSABE), pp. 227-245, 2004.
[8] T. Mifsud and P. Stanski, "Measuring Performance of Dynamic Web Service Migration using LAMS," Proc. 10th Int'l Conf. Software, Telecommunications and Computer Networks (SoftCOM), pp. 214-218, 2002.
[9] S. Kern and P. Braun, "Towards Adaptive Migration Strategies for Mobile Agents," *Lecture Notes in Computer Science(LNCS)*, Vol.3825, pp. 334-345, 2006.
[10] P. Bellavista, A. Corradi, and E. Magistretti, "Lightweight code mobility for proxy-based service rebinding in MANET," Proc. 1st Int'l Symp., Wireless Communication Systems, pp. 208-214, 2004.
[11] R. Montanari, E. Lupu, and C. Stefanelli, "Policy-based dynamic reconfiguration of mobile-code applications," *Computer*, Vol.37, Issue.7, pp. 73-80, 2004.
[12] S. Kern, P. Braun, C. Fensch, and W. Rossak, "Class Splitting as a Method to Reduce Migration Overhead of Mobile Agents," Proc. Int'l Conf. Ontologies, Databases and Application of Semantics, pp. 1358-1375, 2004.
[13] D. Grove and C. Chambers, "A framework for call graph construction algorithms," *ACM Trans. Programming Languages and Systems*, Vol.23, Issue.6, pp. 685-746, 2001.
[14] D. Grove, G. DeFouw, J. Dean, and C. Chambers, "Call Graph Construction in Object-riented Languages," Proc. ACM SIGPLAN Conf. Object-Oriented Programming Systems, Languages and Applications, pp. 108-124, 1997.

박 명 우

2004년 연세대학교 물리학과 졸업(학사)
2007년 연세대학교 컴퓨터과학과 졸업(석사)
2007년~현재 TMAX Soft R&D Center 연구원

관심분야 : XML, 모바일 웹 서비스, SOA

김 연 석

2003년 명지대학교 전자정보통신공학부 졸업(학사)
2005년 연세대학교 컴퓨터과학과 졸업(석사)
2005년~현재 연세대학교 컴퓨터과학과 박사과정

관심분야 : 웹문서 분석, 정보 추출 및 통합, XML, 모바일 웹 서비스

이 경 호

1995년 연세대학교 전산과학과 졸업(학사)
1997년 연세대학교 컴퓨터과학과 졸업(석사)
2001년 연세대학교 컴퓨터과학과 졸업(박사)
2001년 National Institute of Standards and Technology (NIST) 객원연구원
2002년~현재 연세대학교 컴퓨터과학과 부교수
관심분야 : 멀티미디어 문서처리, XML, 웹 서비스