모바일 디바이스에서 외부 데이터 영역의 확장을 위한 자원관리시스템의 설계 및 구현

Design and Implementation of Resources Management System for Extension of outside Data Space in Mobile Device

나승원(Seung-Won Na), 오세만(Se-Man Oh)

초 록

이동통신과 인터넷 기술의 결합으로 탄생한 무선 인터넷은 휴대의 편리성을 제공하고 있으나 모바일 환경의 제약사항 때문에 대중적인 인터넷 서비스로는 발전하지 못하고 있다. 무선 환경의 제한 요소 중 협소한 메모리 공간으로 모바일 디바이스에서는 효율적인 자원 관리를 수행하지 못하는 단점을 가지고 있다. 휴대성이 고려되어야 하는 하드웨어 특성상 충분한 메모리 용량을 확보하는 데에는 한계가 있으므로 향후에는 디바이스 내부의 저장 장치에서 외부의 영역까지 메모리 공간을 확장하여 데이터 처리가 수행될 수 있는 플랫폼 구조로 발전되어야 할 것이다.

본 논문에서는 모바일 디바이스의 내부에서 외부의 서버까지 메모리 공간을 런타임 액세스(Run-time Access)에 의해서 확장하여 데이터 활용을 가능하게 하고 디바이스 내부의 파일을 효율적으로 관리할 수 있는 모바일 에이전트를 제안하며 이를 실현하기 위한 자원 관리시스템(RMS: Resources Management System)을 설계하고 구현하였다. 제안된 RMS를 적용한 디바이스는 '모바일 공간확장'으로 확대된 프로세스 적용이 가능하며 내부 파일을 효과적으로 관리하여 최적의 메모리 공간을 유지하는 효과가 있다.

Abstract

Wireless Internet, created through the merging of mobile communication with Internet technology, provides the advantage of mobility, but the restrictions of the mobile environment are deterring it from growing into a mass public service. Of the restricting factors of the wireless environment, narrow memory space creates the disadvantage of not being able to manage resources in mobile devices efficiently. Because there is a limit to obtaining sufficient memory space from hardware made with consideration of portability, future devices will need to have a platform design with storage area extended from internal storage to external storage space.

In this paper, we present a mobile agent that extends the memory space from only the inside of a mobile device to an external server making it possible to use data by on-line Run-time, and can also manage internal files efficiently. We have designed and implemented a RMS(Resources Management System) as a realization. Devices using the proposed RMS will be able to apply extended processes with the 'Mobile Space Extension' and will be benefited with optimal memory space through efficient internal file management.

키워드 : 모바일 에이전트, 자원관리시스템, 모바일 공간확장, 런타임 액세스
Mobile Agent, Resources Management System, Mobile Storage Extension, Run-time Access

* 본 연구는 동국대학교 논문게재연구비 지원으로 수행되었음.

1. 서 론

과거 모바일 디바이스는 휴대 전화 위주의 서비스로 성장해 왔던 반면에 현재의 모바일 디바이스는 전화 서비스뿐 아니라 무선 인터넷 서비스를 지원하는 기기로 제공되고 있는 상황이다. 이동 통신과 인터넷 기술의 결합으로 탄생한 무선 인터넷은 휴대의 편리성을 제공하며 성장해가고 있는 추세이다[10]. 따라서 과거는 통화 위주의 폰 전용 데이터 처리가 주된 내용이었으나 무선 인터넷이 활성화 되면서 요구되는 파일들과 데이터 처리 공간이 추가로 요구되고 있다. 국내의 무선 인터넷 서비스 가입자가 2002년말 현재 2천9백만명으로 전체 휴대전화 가입자의 89%를 점유하고 있다[11]. 즉, 무선 인터넷 사용이 가능한 디바이스를 가지고 있는 가입자가 89%에 이른다. 그러나 월평균 1시간 이상의 이용자는 전체의 10%이하 라는 통계가 있다[11]. 이것은 모바일 환경에서 무선 인터넷이 가지고 있는 불규칙한 통신 대역과 작은 디바이스 처리 능력 등의 제약사항 때문에 대중적인 서비스로 발전하지 못하고 있는 실정이다. 특히 협소한 메모리 공간으로 인한 적은 프로세스 수행 능력 때문에 디바이스에서 효율적인 자원 관리를 수행하지 못하는 단점을 가지고 있다. 기술의 발전은 메모리 공간을 확대시켜 기존의 문제점을 해결할 수는 있겠지만 소형화 되어야 하는 하드웨어 특성상 충분한 메모리 용량을 무한대로 확보하는 데에는 한계가 있다[3]. 따라서 무선 인터넷을 지원하는 모바일 디바이스의 데이터 처리 영역은 내부의 공간뿐 아니라 외부의 영역까지 메모리 공간을 확대하여 수행되는 플랫폼 구조로 발전될 것이다.

본 논문에서는 모바일 디바이스의 내부에서 외부의 영역까지 메모리 공간을 확장하여 데이터 활용이 가능하고 디바이스 내부의 파일을 관리할 수 있는 자원 관리시스템(RMS: Resources Management System)을 설계하고 구현하였다. RMS는 기존의 다운로드 시스템의 한계를 보안하는 개념으로 접근하였으며 설계 방법을 에이전트 기술과 분산객체 기술을 이용하여 구축하였다. 제안된 RMS를 적용한 디바이스는 '모바일 공간 확장'과 '내부 파일 관리'의 기능이 적용되어 데이터 영역의 확장을 통해서 프로세스의 확대 수행이 가능해지며 내부 파일을 효과적으로 관리하여 탄력적인 메모리 공간을 유지하는 기능이 수행된다.

본 논문의 2장에서는 관련연구 사항으로 모바일에서 파일 전송 방식과 분산 에이전트 기술에 대해서 살펴보고, 3장에서는 자원 관리를 위한 RMS를 설계하였다. 4장에서는 제안에 따른 구현 내용을 설명하였으며, 5장에서는 결론을 맺는다.

2. 관련 연구

관련 연구 사항으로는 모바일 환경에서 기존의 메모리 확장 방법과 파일 전송 방식에 대해서 설명하고 디바이스에서 수행되는 에이전트 기술과 분산객체 기술에 대해서 조사하였다.

2.1 디바이스의 메모리 확장 방법

기존 모바일 디바이스에서 메모리 공간을 확장 시킬 수 있는 방법으로는 첫째, 고성능의 메

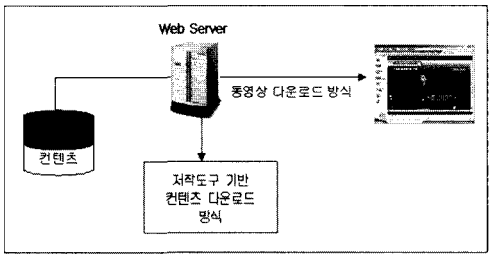
모리로 대체하는 방안이다. 그러나 기술이 발달한다고 해도 소형화 되어야 하는 하드웨어 특성상 메모리 공간 확대는 한계가 있으며 기존 사용자에게는 적용될 수 없는 방법이다. 둘째, 메모리 공간의 한계를 극복하기 위한 파일 전송 방식으로는 다운로드 시스템과 스트리밍 방식의 시스템을 적용하고 있으나 프로세스가 이동하는 개념이 아닌 제한적 형태의 공간 점유이다. 셋째, 통신 속도가 보장된다는 가정하에 분산된 파일들을 온라인 런타임 액세스가 가능한 구조로 설계하는 방안에 대해서 고려해 볼 수 있다. 리모트의 공간을 최소화하고 로컬의 파일을 리모트로 적시에 적재되어 수행되는 구조를 모바일 환경에서 적용하는 개념은 아직 구체화되지 않았으나 본 논문에서 제안하고자 한다.

2.2 모바일 환경에서 파일 전송방식

모바일 환경에서 파일 전송 방식은 크게 다운로드 방식과 스트리밍 방식으로 나누어지며 그 특징은 다음과 같다.

2.2.1 다운로드 방식

다운로드 방식은 인코딩된 동영상 파일이나 컨텐츠를 웹 서버에 준비하여 전송해 주는 시스템으로서 사용자는 필요한 파일들을 자신의 모바일 공간에 저장하여 수행하는 구조이다. 다운로드 받는 파일은 컴파일되어 가공이 모두 끝난 최종 파일을 다운받아 수행하는 형태이다. 한번 다운받은 파일은 추후에 오프라인 수행을 통해서 재사용 될 수 있는 장점이 있으나 디바이스 메모리의 하드웨어 특성상 저장 공간이 부족하고 저장된 파일은 저작권의 보호를 받을 수 없다는 단점이 있다. 다운로드 시스템의 구성은 다음과 같다[8].

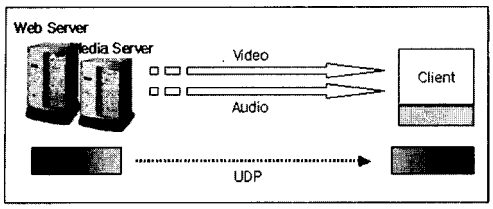


〈그림 1〉 다운로드 방식의 시스템 구성도

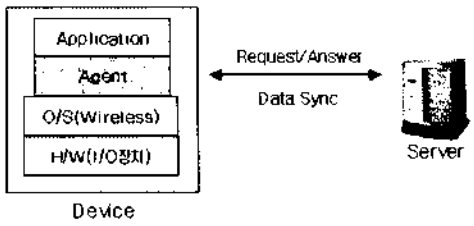
2.2.2 스트리밍 방식

스트리밍 방식은 동영상 컨텐츠를 다운로드 하면서 재생하는 방식으로 전체 파일을 모두 전송 받는 다운로드 방식에 비해서 수행 대기 시간이 짧은 것이 특징이다. 스트리밍 방식의 파일 정보는 영상과 음성으로 이루어져 있으며 전송을 위해서는 웹 서버와 별도로 미디어 서버를 필요로 하는 구조이다. 이 방식은 컨텐츠의 재생 시간을 줄이는 장점과 컨텐츠 파일의 무단 복제를 제한할 수 있는 특징으로 온라인 서비스의 기본 모델로 쓰여지고 있으나 전송 품질이 보장되지 않는다는 것과 미디어 변환용의 인코더 시스템을 구축하기 위한 고가의 구축비용이 요구되는 단점이 있다. 동영상 스트리밍 방식의 구성은 다음과 같다[8].

기존의 방법들은 진정한 의미에서의 모바일 공간 확장이라고 볼 수는 없으며 모바일 공간의 부분적 점유라고 할 수 있다.



〈그림 2〉 스트리밍 방식의 시스템 구성도



〈그림 3〉 무선 환경에서의 에이전트 시스템

2.3 분산 에이전트 기술

에이전트 기술은 사용자가 수행 할 작업이나 반복적인 일 들을 자동적으로 처리해 주는 시스템이다[6].

분산 컴퓨팅 환경에서의 에이전트는 인터페이스 에이전트, 태스크 에이전트, 정보 에이전트로 구분한다. 인터페이스 에이전트는 사용자와 직접적으로 상호 작용을 하면서 사용자의 요구 사항을 분석한다. 태스크 에이전트는 영역 지식을 기반으로 사용자의 요구 사항을 직접 수행한다. 그리고 정보 에이전트는 여러 곳에 흩어져 있는 정보에 접근하는 기능을 제공하는 에이전트이다[2].

무선 환경에서의 에이전트는 필요한 파일을 서버로부터 호출하여 수행시키는 형태로서 앞에서 언급한 다운로드 시스템의 구조를 보여주고 있다. 무선 에이전트는 〈그림3〉과 같이 운영체제 위에서 존재하며 해당 애플리케이션과 인터페이스 관계를 형성하며 수행을 한다.

2.4 분산 객체 기술

분산 객체(Distributed Object)는 다른 메모리에 존재하고 있는 객체를 네트워크를 이용하여 호출하는 형태로서 일반 객체와 마찬가지로 데이터베이스와 메소드의 형태로 이루어져 있으며 플랫폼과 프로그램 언어에 대한 독립성을 제공한다. 분산 객체의 배경 기술로는 객체지향 및 미들웨어 기술 등이 있으며 대표적인 분산 객체로는 RMI와 DCOM등이 있다.

2.4.1 RMI(Remote Method Invocation)

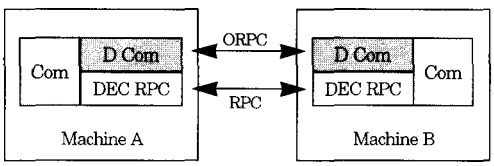
RMI는 자바 기반의 분산 객체 기술로서 분산되어 있는 객체를 로컬에 있는 것처럼 사용할 수 있는 기술로 기존의 소켓 프로그램과 같은 복잡한 설계 절차와 에러를 발생시키는 요인을 최소화한 것이 특징이다[3]. 소켓 API를 사용한 부분은 RMI 패키지, 클라이언트 스터브(Client Stub) 그리고 서버 스터브(Server Stub)에 의해서 포함된다. RMI는 RPC(Remote Procedure Call)와는 달리 JVM 환경에서 실행되므로 자바 이외의 다른 언어는 지원하지 않는다. RMI의 주요 기능은 원격 호출을 제공하고, 서버에서 애플릿의 콜백(Callback) URL 적용이 가능하며 분산 객체와 비분산 객체의 명확한 구분이 가능한 기능을 제공한다.

2.4.2 DCOM(Distributed COM)

DCOM은 원격 객체 액세스를 지원하는 마이크로 소프트의 기술이다. 윈도우 환경에서 컴포넌트 간의 연동을 위한 분산 객체 시스템으로 네트워크에서 원격 객체가 제공하는 서비스들을 요청하기 위해서 클라이언트 응용 프로그램을 사용하도록 지원한다.

COM(Component Object Model)은 같은 기계에서 컴포넌트 간의 상호 작용만을 지원하지만 DCOM은 이러한 COM의 제약을 해결하여 분산 컴퓨팅 환경에서도 컴포넌트 간의 상호 작용을 가능하게 하였다. COM에서는 제공하지 않는 위치 투명성과 원격 활성화, 객체들 간의 연결 관리, 그리고 보안 기능이 DCOM에서 추가된 사항이다[7].

<그림4>에서 DCOM의 컴포넌트 간 연결 프로토콜은 RPC(Remote Procedure Call)를 기반으로 하며 DCOM 프로토콜의 상위 계층인 ORPC(Object RPC)는 객체 지향RPC를 지원하는 프로토콜을 의미한다.

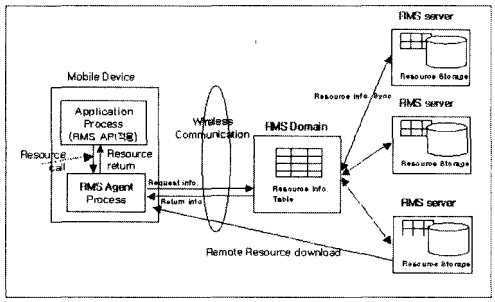


<그림 4> ORPC 기반의 DCOM 구조

3. RMS의 설계

3.1 RMS의 개요

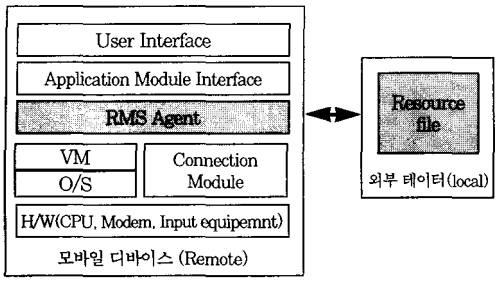
RMS는 데이터 저장 공간을 외부의 영역까지 확장할 수 있으며 디바이스 내부의 자원 관리를 효율적으로 수행하기 위한 모바일 에이전트 기술이다. RMS의 기본 구조는 RMS 에이전트, RMS 도메인, RMS 서버 등 세가지 영역의 시스템 구성으로 이루어졌으며, <그림5>와 같다.



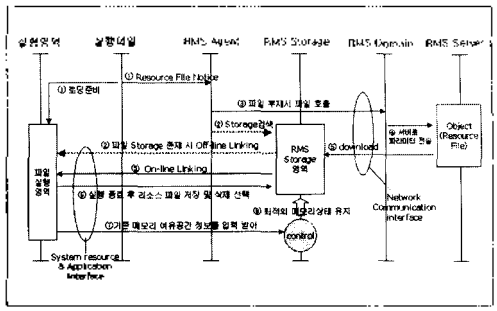
<그림 5> RMS 시스템 구성도

RMS 에이전트는 디바이스 내부에서 수행되며 실행 파일과 링크 될 리소스 파일을 RMS 도메인에게 호출한다. RMS 도메인은 등록된 리소스 파일을 포함한 오브젝트의 원격 주소를 관리하여 에이전트와 서버간의 원활한 로드 밸런싱 기능을 제공한다. RMS 서버는 해당 리소스 파일이 존재하는 서버로서 에이전트에게 해당 파일을 제공한다. 이때 에이전트로 제공되는 리소스 파일은 DLL(Dynamic Linking Library) 파일과 OCX(OLE Control Extension) 파일, 그리고 컨텐츠 파일을 대상으로 한다

RMS 에이전트는 디바이스의 운영체제 위에서 수행되고 애플리케이션과의 호환성을 위해서 VM(Virtual Machine)과도 인터페이스 한다. 제안하는 RMS의 아키텍쳐 구조는 다음 <그림6>과 같다.



〈그림 6〉 RMS 아키텍처 구조



〈그림 7〉 RMS 실행 다이어그램

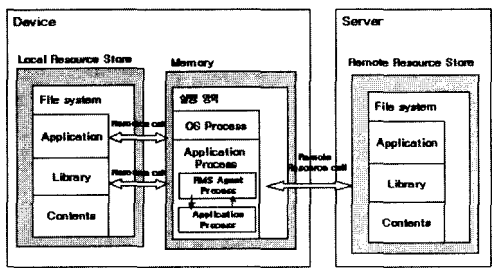
하드웨어는 디바이스의 CPU, 배터리, 무선 모뎀, 입/출력 장치 등을 의미하며 커넥션 모듈(Connection Module)은 무선 접속을 원활한 수행을 위한 접속 API를 의미한다. RMS 에이전트가 애플리케이션 모듈 인터페이스를 통해서 수행되는 구조이다.

다음 〈그림7〉은 RMS를 수행하는 일련의 과정들을 다이어그램으로 나타낸 사항이다. 실행 파일은 에이전트에게 필요한 리소스 파일을 호출하고 에이전트는 해당 리소스 파일을 RMS 도메인과 서버로 요청하여 다운로드를 수행하는 절차이다. 요청시 사전에 해당 파일이 RMS 내부에 존재하는 지의 유무에 대한 검색을 실시하며 RMS 내부에 존재할 경우에는 캐쉬 형태의 오프라인 수행을 제공하게 된다. 그리고 수행 종료 후 리소스 파일은 저장과 삭제가 가능하며 에이전트는 내부의 저장 공간을 자율적으로 관리한다.

3.2 RMS 에이전트의 설계

RMS 에이전트는 디바이스 내부에 존재하는 애플리케이션 프로세스의 리소스 호출 범위를 로컬에서 리모트로 확장하기 위한 프로세스이다. 이것은 데몬(Demon) 형태의 프로세스로 메모리에 상주하며 디바이스와 서버간의 중계 역할을 수행한다.

〈그림8〉은 제안하는 RMS의 리모트 프로세스의 수행 구조로서 리모트의 프로세스 수행 영역을 로컬의 영역까지 확대하여 실행되는 구조로 설계하였다. 에이전트의 기능을 구현하기 위해서 RMS 에이전트를 모바일 공간 확장 영역과 내부파일 관리 영역으로 나누어서 설계 모델을 만들었으며 이를 적용하기 위한 RMS API set을 구성하였다.

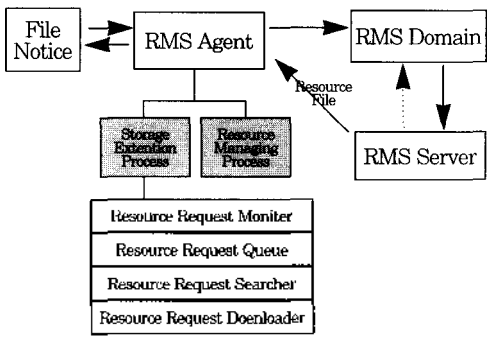


〈그림 8〉 RMS 리모트 프로세스 수행구조

3.2.1 RMS 에이전트 상세 설계

RMS 에이전트는 디바이스에서 애플리케이션 프로세스의 리소스 요청에 대한 처리를 수행하는 스토리지 확장 프로세스 영역과 로컬에 저장된 리소스 파일을 관리하는 리소스 매니징 프로세스 영역으로 구성된다.

첫째, 스토리지 확장 프로세스는 '모바일 공간확장' 이 수행되는 영역으로 모바일 디바이스에서 수행되는 실행 모듈이 메모리 환경에서 애플리케이션 프로세스 형태로 수행되는 것이며 리소스 파일을 호출 할 경우 RMS 에이전트 프로세스가 리소스 호출 부분의 수행을 대신하여 리소스의 위치 정보를 검색한 후에 해당 리소스 핸들을 애플리케이션 프로세스로 반환하는 역할을 수행하며, 〈그림9〉와 같은 구조로 구성된다. 리소스 파일은 RMS가 관리하는 별도의 스토리지에 다운로드 되어 실행된다.

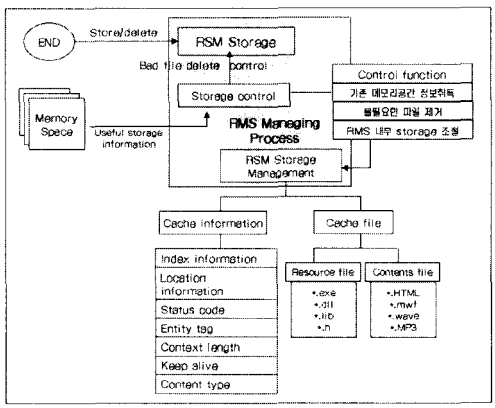


〈그림 9〉 Storage Extension Process

둘째, 리소스 매니징 프로세스는 다운로드된 리소스 파일을 관리하는 프로세스 영역으로 RMS 스토리지 영역의 리소스 파일을 저장하거나 삭제할 수 있다. 삭제한 경우에도 에이전트는 파일 호출시 적시에 공급받아 수행되는 것을 원칙으로 한다. 그러나 저장된 파일은 추후 재 요청시에 캐쉬(cache)의 off-line 서비스 형태로 제공되며, 다음 〈그림10〉과 같다.

내부 파일을 관리하는 RMS 저장 장소 관리 기능은 데이터 공간을 조절하여 최적의 메모리 상태를 유지하여 효과적인 자원관리 기능을 수행한다. 즉, 내부 파일 만을 효율적으로 관리하는 것으로도 '모바일 공간 확장' 의 효과를 기대할 수 있다.

스토리지 내부의 파일 관리 기준은 오래된 파일부터 삭제하고 동일 조건시 버전 정보나 파일 크기의 순으로 관리하는 일반적인 방법론을 적용하였다.



〈그림 10〉 Storage Managing Process

3.2.2 RMS 에이전트 API set의 구성

RMS가 모바일 공간을 확장하고 내부 파일을 관리하는 기능을 제공할 수 있는 것을 RMS 에이전트 API Set의 구성을 통해서 적용하였다. RMS 에이전트 API set은 기존 디바이스

의 운영 체제에 종속적인 네이티브(native) API의 일부 부분에 대해서 독립적인 기능을 수행하도록 보완하는 개념이다. RMS 에이전트 API Set은 〈표1〉과 같다.

〈표 1〉 RMS 에이전트 API set

RMS API Set	기 능
RMS_Create()	RMS 실행을 위한 메인 인터페이스 함수
RMS_Load_Resource()	실행 파일이 수행하기 위한 리소스 파일 호출
RMS_Search_Location()	리소스의 위치를 검색하여 위치 정보 제공
RMS_Link_App()	Storage에 있는 리소스 파일이 기존의Application 과 Link
RMS_Get_Resource()	RMS_Search_Localtion의 위치 정보에 따른 리소스 핸들 반환
RMS_Process_Type()	리소스의 Type 반환 및 관련 Application 정보 제공
RMS_VM_Interface()	리소스 호출 후 기존 내장되어 있는 VM과의 호환성 유지
RMS_Storage_Store()	실행 후 RMS 스토리지에 저장
RMS_Storage_Delete()	실행 후 불필요한 파일의 삭제
RMS_File_Compare()	RMS 내부 공간의 파일비교
RMS_File_Managing	내부공간 파일 비교후 중복 데이터 및 과거데이터 삭제

추가된 API의 기능은 네이티브 API에서 로컬 영역의 리소스 호출에 대한 리모트 호출 부분을 지원하는 기능으로 확장되어 있다. 〈그림 11〉은 RMS API Set을 적용한 RMS Agent 소스 코드의 일부이다

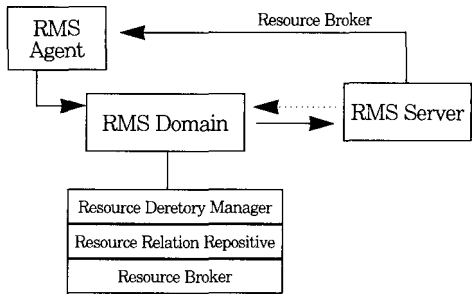
```
// test.exe 프로그램 프로세스에서 test.dll을 remote
 call 하여
// test.exe 프로그램 프로세스에 Load하는 프로그램

#include "stdafx.h"
#include "RMS_Library.h"

RMS_Create()//RMS 실행을 위한 메인 인터페이스 함수
int RMS_Load_Resource(szfileName,
szfileName,
        szCmdLine,
        NULL,
        NULL, FALSE,
        dwCreationFlags,
        NULL,NULL,NULL, &pi)
{
        INT rc;
        switch (RMS_Search_Location(szfileName)) {
// RMS Agent에 Resource의 위치 검색요청
// RMS Agent는 위치 정보 파악 후 관련 위치 정보 및
// Resource 모듈 정보를 Return함
case IS_Remote_TCP :  // Remote 위치일 경우
RMS_Get_Resource(szfileName);
// 도메인 서버의 중재로 RMS Agent와 RMS 서버와의
// Resource download 시행 후 Resource memory load
if(RMS_Process_Type(szfileName) == IS_PROCESS)
  // 메모리에 Load된 Resource 타입이 Program일 경우
rc = CreateProcess (szfileName, szCmdLine,
      NULL, NULL, FALSE,
      dwCreationFlags, NULL,
      NULL,NULL, &pi); // 신규 out-process 생성
if (RMS_Process_Type(szfileName) == IS_LIBRARY){
  //메모리에 Load된 Resource 타입이 Library일 경우
    HINSTANCE hInst;
    int (*pFunc)(LPCTSTR szName);
if ((hInst = LoadLibrary(szParameter)) == NULL)
  // 신규 in-Process load.
{
  LPVOID lpMsgBuf;
  Message(Message_FORMAT);
  LocalFree( lpMsgBuf );
  return 0;
    }
}
f (RMS_Process_Type(szfileName) == IS_CONTENTS)
    // 메모리에 Load된 Resource 타입이 Contents일 경우
  CreateProcess(L_MOUNT_APP, szParameter, NULL,
NULL, NULL, 0, NULL, NULL, NULL, &pi);
 // contents file과 연계되는 외부 out-process 생성 후
    // 컨텐츠 파라미터 전달
```

```
break;
case IS_LOCAL_CACHE : // Local Cache 위치일 경우
  break;
case IS_LOCAL_IN : //  Local 위치일 경우
break;
case IS_NON : // 위치 정보 값이 없을 경우
break;
          }
return rc;}
int WINAPI WinMain( HINSTANCE hInstance,
HINSTANCE hPrevInstance,
LPTSTR    lpCmdLine,
int       nCmdShow)
{
   TCHAR szfileName[MAX_PATH];
   TCHAR szCmdLine [64];
   DWORD dwCreationFlags;
   PROCESS_INFORMATION pi;
   INT rc;
   lstrcpy (szfileName, TEXT ("test.dll"));
/* 메인 프로세스에서 test.dll 호출
 lstrcpy (szfileName, TEXT ("test.bmp")); //Image 호출
 lstrcpy (szfileName, TEXT ("test.mp3")); //Sound 호출
 lstrcpy (szfileName, TEXT ("test.wmv")); //movie 호출
*/
    lstrcpy (szCmdLine, TEXT (""));
    dwCreationFlags = 0;
  rc = RMS_Load_Resource
(szfileName, szCmdLine,NULL,
     NULL, FALSE,
        dwCreationFlags, NULL,NULL,NULL, &pi);
// 메인 프로세스에서 test.dll을 로드하기 위해
// RMS Agent API의 Resource Load 함수를 호출
          if(rc) {
CloseHandle (pi.hThread);
              CloseHandle (pi.hProcess);
  }
return 0;
}
```

〈그림 11〉 RMS Agent 코드설계

3.3 RMS 도메인의 설계

RMS 도메인에서는 RMS 에이전트로부터 호출 받은 리소스의 위치 정보를 분류하고 해당 리소스 파일이 있는 서버와 RMS 에이전트를 연결하여 주는 미들웨어의 역할을 수행한다.



〈그림 12〉 RMS 도메인의 구조

RMS 도메인은 〈그림12〉와 같이 세 가지의 기능이 있다. 첫째, 리소스 디렉토리 매니징의 기능은 등록된 객체 파일의 원격 서버 주소나 로컬 디렉토리, 파일이름, 리소스 파일의 주소 값을 관리하는 기능을 제공한다. 둘째, 리소스 릴레이션 레퍼지토리(repository)의 기능은 리소스 파일의 GUID와 버전 정보, 부가 정보 및 관련 객체의 GUID를 제공하여 준다. 세째, 리소스 브로커의 기능은 RMS 에이전트와 RMS 서버간의 최적의 경로를 검색하여 해당 파일을 에이전트에게 연결하는 기능을 지원한다. 그리고 다수의 에이전트가 동시에 접속 할 경우 접속 순위를 결정하여 네트워크 경로 및 RMS 서버 부하에 대한 에러처리 복구 기능과 확장성 지원 기능, 그리고 파일 전송에 대한 로드 밸런싱 기능을 제공한다.

리소스 디렉토리 매니징 기능 중 RMS 에이전트로부터 호출 받은 접속 데이터 값을 RMS

서버로 전송하기 위해서는 해당 리소스 파일 정보를 관리하는 파일 ID 관리 테이블이 존재해야 하며 대략적인 ID 구조를 〈표2〉에서 나타내었다.

〈표 2〉 RMS 파일 ID 관리 테이블

[illegible]	ID	[illegible]
접속ID	01	고객 ID, 전화번호(MIN)
디바이스	02	PDA, Hand-phone, other
디바이스 O/S	03	PPC 2002, .Net, Palm, Linux
리소스 파일	04	▷ Resource : dll, ocx ▷ contents : App, 컨텐츠

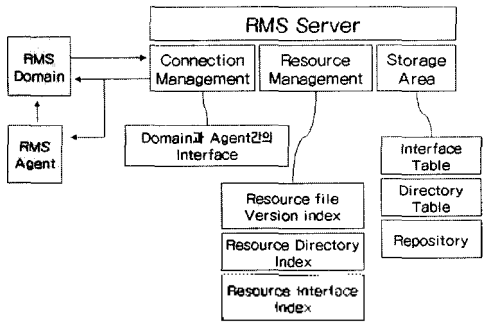
〈그림13〉은 에이전트로부터 호출받은 리턴 값을 서버로 연동하기 위한 브로커 영역에서의 코드 표현이다.

```
//Resource Broker부분 code
InitServerPath();
  //도메인에 등록되어 있는 서버 리스트 초기화
InitInfoSync();
  // 도메인에 등록되어 있는 서버와의 정보 동기화
while(FaultDomain()) //에러가 없는 동안 Loop 수행
{
rqParam= Get_Message(rqMessage);
  //도메인에 요청된 리소스 call 메시지 수신
if(ServerLoadBalance(rqParam))
  /* 수신된 메시지에서 요청한 file이 위치하는 서버
  검색 및 최적 경로 설정*/
{
Broke_AgentServer(rqParam);
//에이전트와 서버간의 연결 설정
{
else }
   Message(Message_FORMAT);
  }
}
```

〈그림 13〉 RMS Domain 코드 설계

3.4 RMS 서버의 설계

RMS 서버는 RMS 도메인으로부터 리턴받은 호출에 대해서 해당 리소스 파일을 RMS 에이전트에 제공하는 역할을 한다. RMS 서버를 구성하는 세가지 주요 기능은 접속 모듈을 담당하는 커넥션 매니지먼트(Connection Management)기능과 파일 버전 정보의 관리를 위한 리소스 매니지먼트(Resource Management)기능, 그리고 RMS 에이전트로 전송될 리소스 파일이 실제로 존재하는 스토리지 영역(Storage Area)의 역할을 담당하고 있다.



〈그림 14〉 RMS 서버의 구조

첫째, 커넥션 매니지먼트의 기능은 RMS 도메인을 통해서 호출한 RMS 에이전트와 연결관리를 수행한다. 이것은 모든 RMS 에이전트에 개방되어 RMS 서버가 응답할 때 전체 네트워크의 성능 저하를 초래하기 때문에 RMS 도메인에서 연결된 RMS 에이전트와 전용으로 인터페이스 된다. 그리고 RMS 도메인의 리소스 릴레이션 레퍼지토리와 리소스 버전 컨트롤

을 수행하여 최신의 파일 정보를 실시간으로 제공해 준다. 둘째, 리소스 매니지먼트에서는 리소스 파일의 등록 및 버전 관리 기능과 디렉토리 인덱스 관리, 인터페이스 인덱스 관리 등의 기능이 수행되어 모바일 디바이스 타입, 리소스 파일 타입, 리소스 파일의 버전 및 파일 크기 등을 관리한다. 셋째, 스토리지 영역은 제공 될 리소스 파일이 위치하는 곳으로 디렉토리 테이블과 인터페이스 테이블, 레퍼지토리 등을 관리하는 기능을 제공한다.

다음 〈그림15〉는 RMS 서버에서 컨넥션 매니지먼트에 관련된 코드의 일부분이다.

```
//Connection Management code 부분
while(FaultServer())
{
Init_PathInfo(); // 도메인에 연결 설정 초기화
if(Init_ConnServerAgent(rqParam))
//에이전트와 서버간의 연결 초기화 및
// 파일 전송 준비
{
  if(Init_VersionControl(rqParam)){
 // 요청한 파일과 전송할 파일의 버전
//컨트롤 수행
    if (Sendfile(rqParam)){
 // 신규 버전 파일을 에이전트에 전송
 // 파일 전송 에러 처리 구문}
  }
 }
else {
  Message(Message_Format));
}
}
```

〈그림 15〉 RMS Server 코드설계

4. RMS의 구현

4.1 개발환경 및 구현도구

RMS를 구현하기 위해서 PDA(Personal Digital Assistants)를 선정하였다. 휴대폰 보다 확대된 기능을 제공하는 것은 PDA가 우수하며 향후의 무선 인터넷 모바일 디바이스는 휴대폰의 크기와 PDA의 성능이 결합된 모델로 제공될 것이다. RMS를 적용하기 위한 개발환경 및 구현도구는 〈표3〉과 같다.

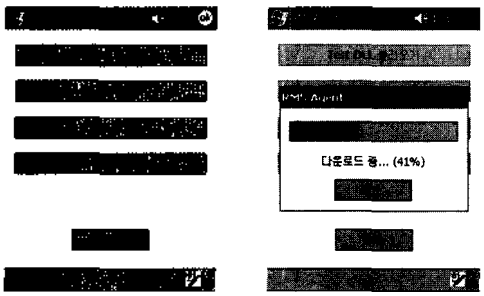
〈표 3〉 개발환경 및 구현도구

구 분		세부내용
Language		Embedded Visual C++3.0
Communication		CDMA 2000 1x
PDA O/S		PPC 2002
PDA	Kind	Poz, iPAQ
	CPU	Xscale 200MHz StrongArm 206MHz
	RAM	36Mbytc
	ROM	36Mbyte
RMS 도메인 서버		Custom Service Server
RMS 서버		Custom Service Server

4.2 구현결과

다음은 RMS의 구현 결과로서 〈그림16〉의 좌측은 RMS 에이전트의 메인 화면이고 우측은 리소스 파일을 호출하여 다운로드 받는 과정을 나타낸 화면이다.

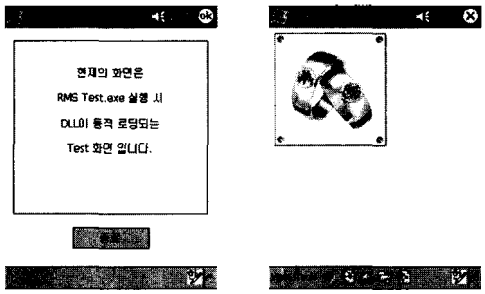
이하에서 나타내는 모든 화면은 PPC2002의 운영 체제를 사용하는 POZ라는 PDA에서 구현된 결과의 화면이다.



〈그림 16〉 RMS의 메인 화면과 다운로딩 동작 화면

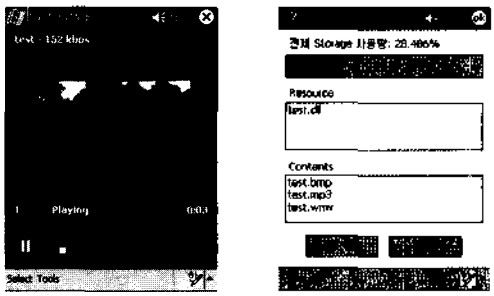
위의 〈그림16〉에서 좌측의 호출 화면은 라이브러리인 Test.dll과 이미지 파일인 Test.bmp, 그리고 음악 파일인 Test.mp3, 동영상 파일인 Test.wmv 파일을 호출하기 위한 화면의 구성이다.

다음 〈그림17〉은 RMS를 통해서 구현된 결과로서 좌측은 라이브러리 파일인 Test.dll 파일이 실행 파일과 동적 로딩되어 수행되는 결과를 보여주고 있으며 오른쪽 화면은 그림 파일인 Test.bmp 파일을 호출하여 보여준 결과이다.



〈그림 17〉 dll과 bmp 파일을 호출하여 수행한 화면

다음 〈그림18〉의 좌측은 동영상 파일의 구현 결과이고 우측 그림은 다운로드 받은 파일들이 RMS 스토리지 영역에 존재하고 있는 화면이다. 현재는 RMS 스토리지 공간의 크기를 5M 바이트로 설정하였으나 탄력적으로 조절이 가능하다.



〈그림 18〉 동영상 화면과 RMS의 스토리지 영역

구현 결과와 같이 무선 접속을 통한 다운로딩 시간은 필요하나 완료 후 수행되는 것과 오프라인시 수행되는 것을 비교하여 볼 때 작동하는 데에는 전혀 문제가 없었으며 단일 파일을 수행시키는 관점에서는 내부의 메모리 공간에 다른 파일들을 포함하고 있을 때 수행되는 것 보다는 RMS의 수행시 퍼포먼스가 상대적으로 높았다. 이와 같은 실험 결과는 다음과 같다.

〈표 4〉 메모리 측정 실험파일

NO	실험파일	사이즈
1	Test.dll	5 Kb
2	Test.bmp	17 Kb
3	Test.MP3	1,171 Kb
4	Test.wmv	267 Kb
5	Calc.exe	73.5 Kb
6	Ebook.exe	441 Kb
7	Pword.exe	105 Kb

〈표 5〉 메모리 측정 방법

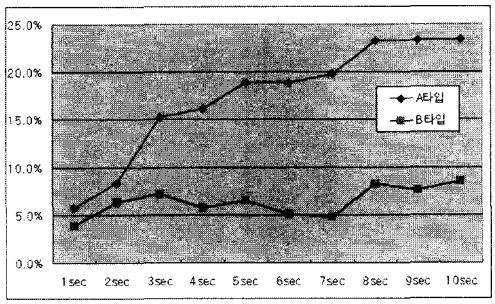
구분	확장조건
저장공간	32Mb중 16Mb를 책정
Free Memory	11Mb로 책정
측정시간	1초 단위로 메모리 측정
측정방법	메모리 할당량 비교(A, B)

위의 〈표4〉와 〈표5〉와 같이 7개의 파일을 선정하여 두 가지의 타입으로 실행하여 보았다. A의 타입은 기존 메모리 공간에 1부터 7개의 파일을 순차적으로 실행하며 이전에 실행되었던 파일을 저장 공간에 모두 가지고 있는 상태에서 수행한 경우이고 B타입은 RMS 개념으로 7개중 각각의 파일을 랜덤하게 선택하여 실행시킨 경우이다. 이 경우의 수행시에 Free Memory에 할당되는 점유율에 대해서 1초 단위로 체크되는 메모리 양과 점유율에 대해서는 다음 〈표6〉과 같은 실험 결과를 보였다.

〈표 6〉 메모리 할당량 실험결과

1sec	10,358,784	10,567,680	5.8%	3.9%
2sec	10,084,352	10,297,344	8.3%	6.4%
3sec	9,318,400	10,201,728	15.3%	7.3%
4sec	9,224,192	10,358,784	16.1%	5.8%
5sec	8,925,184	10,272,768	18.9%	6.6%
6sec	8,925,184	10,432,512	18.9%	5.2%
7sec	8,822,784	10,469,376	19.8%	4.8%
8sec	8,445,952	10,092,544	23.2%	8.2%
9sec	8,441,856	10,155,392	23.3%	7.7%
10sec	8,431,846	10,056,792	23.3%	8.6%

A의 타입보다는 B의 타입으로 수행시 메모리 할당량이 적은 것을 알 수 있다. 이 내용의 메모리 할당 점유율을 그래프로 표현하면 〈그림19〉와 같다.



〈그림 19〉 메모리 할당 점유율

메모리 할당 점유율 높은 것은 수행시 퍼포먼스에 영향을 주어 저조한 실행 속도를 제공하는 원인이 된다. 따라서 속도 측면에서 B의 타입이 우수한 것을 알 수 있다.

종합적으로 정리하여 볼 때 RMS를 적용한 연구 결과의 장점은 다음과 같다.

첫째, RMS를 적용한 디바이스는 모바일 공간 확장으로 효과적인 자원관리를 수행할 수 있다.

둘째, 최소의 공간 확보로도 저가의 디바이스를 공급하여 가격 부담의 해소와 대중적 사용을 통한 모바일 언터넷 서비스의 활성화를 기대할 수 있다.

셋째, 메모리의 저 용량을 사용하던 기존 사용자까지 수용할 수 있는 시스템이다. 메모리가 증대되는 기술은 최신의 디바이스에만 적용되는 사항이지 기존 사용자까지 적용할 수 없는 경우가 대부분이다.

반면에 RMS를 적용할 경우 우선적으로 선행되어야 하는 문제점은 다음과 같다.

첫째, 통신속도의 개선 및 안정적인 서비스의 대역폭이 보장되어야 한다. 2003년 5월 현재 국내의 통신망은 CDMA 2000 1X-EVDO 상용화로 2.4Mb의 데이터 전송속도 구현이 가

능해졌다. 통신 기술의 발전은 지속될 것이고 향후 2년 내에는 보다 개선된 통신 대역을 기대할 수 있을 것이다[9].

둘째, RMS 적용시 리얼 타임으로 파일 액세스를 수행하기 위해서는 에이전트와 서버간 세션이 항상 오픈 되어야 한다. 이 기능을 수행하기 위해서는 추가적인 수행 능력이 필요하게 되나 현재의 디바이스에서 처리하는 것에는 무리가 있다. 실험단계가 아닌 상용화 단계에서는 반드시 고용량의 CPU를 제공하는 디바이스가 필요한 상황이다. 향후에는 CPU의 파워와 메모리 용량의 증가로 추가적인 성능 개선이 가능할 것이다.

5. 결론 및 향후 연구계획

무선 인터넷을 제공하는 모바일 디바이스는 이동의 편리성을 가지고 있는 대신에 메모리 용량이 협소하여 효과적인 자원 관리를 수행하지 못하는 단점을 가지고 있다. 향후의 기술 개발은 이러한 단점을 보완하여 추가적인 메모리 공급이 이루어 질 것이다. 그러나 소형화 되어야 하는 모바일 디바이스의 특성상 메모리 영역을 무한대로 확대한다는 것은 한계가 있다. 따라서 모바일 디바이스의 메모리 공간을 디바이스 내부뿐만 아니라 외부의 영역까지 확장하여 사용할 수 있는 모바일 플랫폼 구조로 변경되어 갈 것이다.

본 논문에서 제안한 '모바일 공간 확장'은 제한된 메모리 공간의 한계를 극복하는데 그 목적을 두고있다. 이 기능을 위해서 외부의 영역까지 데이터 공간을 확장하여 프로세스 수행이 가능하고 디바이스 내부의 파일을 효율적으로 관리하여 최적의 메모리를 확보할 수 있는 모바일 에이전트를 제안하였다. '모바일 공간 확장'을 통한 프로세스의 확대는 향후 모바일 플랫폼의 설계 모델이 될 것이다. 그러나 RMS 에이전트와 서버간에 지속적인 통신이 발생해야 하는 상황에서 프로세스가 수행되어야 하기 때문에 디바이스에서의 파일 처리가 과다하다. 이러한 문제를 효과적으로 보완할 수 있는 추가적인 연구가 필요하다.

향후의 연구 계획은 RMS 에이전트의 기능을 고도화하고 RMS 도메인을 포함한 RMS 서버의 설계를 보완할 예정이며 무선 접속시 디바이스에서 발생하는 트래픽을 최소화 할 수 있는 최적화의 방법을 추가로 연구할 계획이다.

참 고 문 헌

[1] 곽용재역, COM/DCOM 플라이머 플러스, 인포북, 1999.

[2] 김종완, "PDA용 소프트웨어를 위한 무선 에이전트 클래스의 설계 및 구현", 정보과학회 춘계 학술대회 Vol.28, No.1, pp.637-639, 2001.

[3] 김현호, "무선 환경의 RMI 성능개선", 정보과학회 춘계학술대회, Vol.29, No.1, pp.322~324, 2002.

[4] 노영선역, Programming Microsoft WindowsCE 2Edition, 정보문화사, 2001.

[5] 안태균, 포켓 PC와 함께하는 모바일 프로그래밍, 정보게이트, 2002.

[6] 이재호, "에이전트 시스템의 연구 및 개발 동향", 정보과학회지, Vol.18, No.5, pp.4-9, 2000.

[7] 전재철, "DCOM 기술 분석 및 전망", 한국정보처리 학회지, Vol.7, No.4, pp.53~59, 2000.

[8] 최용준, "Qos보장형 스트리밍 서비스를 위한 분산 원격강의 컨텐츠에 대한 연구", 정보처리학회 논문지 A, Vol.9, No.4, pp.603~614, 2002.

[9] 에스케이텔레콤, Mobile Test Lab, <http://www.sktelecom.com/company/telecom_lab/>

[10] 인트로모바일, MobileMultimedia Technology, Technology. <http://www.intromobile.co.kr/solution/>

[11] 정보통신진흥국, 무선인터넷 가입자현황 <http://www.mic.go.kr/jsp/mic_d/d700-0002-1.jsp>

[12] Doug Riecken, "Intelligent Agent", CACM, Vol.37, No.7, pp.18-21, 1994.

[13] James Y. Wilson, Building Powerful Platforms with Windows CE, Aspi Havewala, 2002.

[14] Michael R. Genesereth and Steven P. Ketchpel, "Software Agent", CACM, Vol.37, No.7, pp.48-53, 1994.

[15] Microsoft, PocketPC Official site, <http://www.microsoft.com/mobile/pocketpc/>

[16] Microsoft WinCE, Mobile Solutions, <http://www.microsoft.com/windows/embeded/>

저 자 소 개

나승원 (E-mail : nasw@dgu.ac.kr)
1999. 동국대학교 컴퓨터공학과 박사과정 재학중
1997 ~ 현재 SK텔레콤 플랫폼연구원 재작
2002 ~ 현재 명지전문대학 정보통신과 겸임조교수 재직
관심분야 : 모바일 프로그래밍, 모바일 환경과 인터넷 기반에 중심을 둔 시스템 설계

오세만 (E-mail : smoh@dgu.ac.kr)
1979. 한국과학기술원 석사
1985. 한국과학기술원 공학박사
현재 동국대학교 컴퓨터공학과 교수 재직,
『컴파일러 입문』, 『자바 입문』등의 저서
관심 분야 : 컴파일러 및 프로그래밍 언어, 모바일 환경의 게임프로그래밍과 인터넷 기반의 언어