맞춤형 통신 프로토콜과 서비스를 위한 액티브 네트워크 실행환경

· 이 화 영[†] · 강 보 영[††] · 임 경 식[†††]

요 약

본 논문에서는 프로토콜 조합 및 서비스 참조 기법을 지원하는 Customizable Architecture for Flexible Execution Environment(CAFEs) 실행환경을 설계 및 구현한다. CAFEs 실행환경은 프로토콜 및 서비스 소프트웨어의 재사용성을 높이기 위하여 기존 프로토콜의 기능 또는 알고리즘을 구현한 마이크로 프로토콜과 기존 응용과 호환하고 액티브 네트워크 서비스를 배포하기 위한 수단으로 동작하는 액티브 서비스라는 새로운 개념을 제시한다. 제안된 액티브 네트워크 실행환경은 기능에 따라 세 부분으로 설계되어 있다. 첫째, 가상 네트워크 시스템은 기존의 다양한 프로토콜 계층을 기반으로 가상 네트워크 채널을 형성하고 이를 바탕으로 액티브 노드를 연결한다. 둘째, 프로토콜 및 서비스 관리자는 마이크로 프로토콜 및 액티브 서비스를 합성함으로써 새로운 프로토콜과 서비스를 제공하는 역할을 담당한다. 마지막으로 이벤트 엔진은 이벤트 위임 모델을 사용하여 시스템에 존재하는 컴포넌트의 상태전이를 자동으로 감지한다. 그리고 개발된 CAFEs 실행환경을 검증하기 위하여 무선 인터넷 환경에서 사용자 단말에 적합한 웹 컨텐츠를 전달하기 위한 실험을 수행하여 가용성을 확보하였다.

An Active Network Execution Environment for on Demand Customization of Communication Protocols and Services

Hwa Young Lee[†] · Bo-young Kang[††] · Kyungshik Lim[†††]

ABSTRACT

In this paper, we present the design and implementation of new execution environment named Customizable Architecture for Flexible Execution Environment(CAFEs) that supports the mechanism of protocol customization and service referencing. We introduce a new concept as micro protocols and active services to enhance software reusability. Micro protocol represents a specific algorithm or functionality of existing network protocol and the active service is in charge of binding legacy application and releasing the active network oriented services. The proposed active network execution environment is made up of three parts, virtual network system, protocol and service manager, event engine. First, the virtual network system is used to connect each active nodes using virtual network channels which are based on multiple existing protocol layers. Second, the protocol and service manager is responsible for composing micro protocols and active services to develop new network protocol and service easily. Finally, the event engine is used to detect the automatic transition of system components using event delegation model. To verify the CAFEs, we have an experiment about the delivery of web contents which are suitable for the user's terminals in the wireless Internet environment. As a result, we are able to obtain the availability of developed execution environment.

키워드 : 실행환경(Execution Environment), CAFEs, 마이크로 프로토콜(Micro Protocol), 액티브 서비스(Active Service), 액티브 네트워크(Active Network)

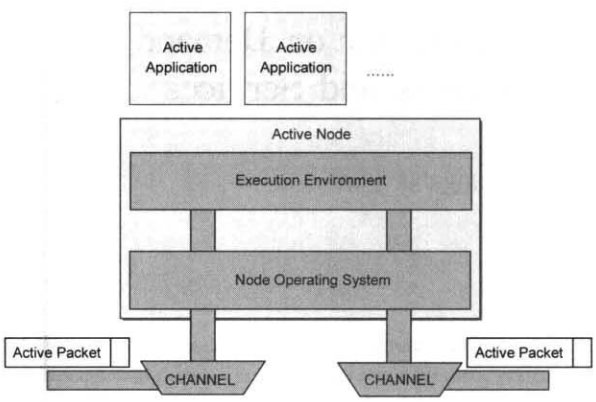
1. 서 론

인터넷 사용의 폭발적인 증가로 인하여 네트워크의 노드가 많아지고 복잡해지면서 네트워크를 관리하는데 소요되는 비용과 시간이 급속히 증가하고 있으며 새로운 프로토콜의 신속한 설치 요구 또한 증가하고 있다. 그러나 폐쇄적이고 유연하지 못한 현재의 네트워크 구조는 사용자 요구 조건의 변화 속도와 이를 지원하기 위한 네트워크 시스템의 변화 속도 간에 차이가 발생하게 되어 사용자의 망에 대한 요구 기능을 시기적절하게 반영하는데 한계가 있다. 뿐만 아니라 프로토콜의 표준화 및 호환성의 문제는 프로토콜을 실제 네트워크에 설치하기까지 많은 시간을 요구함으로써 문제를 더욱 가속화 시킨다[1]. 이를 극복하기 위해 네트워크 노드 구조를 프로그래밍이 가능하도록 하고, 사용

※ 본 연구는 한국과학재단 특정기초연구(R10-2003-000-10562-0) 지원으로 수행되었음.
† 정 회 원 : LG전자 홈넷사업팀 시스템개발그룹 연구원
†† 준 회 원 : 경북대학교 대학원 컴퓨터과학과
††† 정 회 원 : 경북대학교 컴퓨터과학과 교수
논문접수 : 2004년 10월 26일, 심사완료 : 2004년 11월 18일

자 요구 기능을 수행할 수 있는 코드를 전송, 실행함으로써 통신망에 새로운 서비스를 보다 신속하고 경제적으로 도입하고 망 자원을 보다 적절하게 활용할 수 있도록 하는데 목표를 두고 연구되고 있는 분야가 액티브 네트워크 분야이다.

액티브 네트워크는 네트워크 노드 구조상의 폐쇄성, 경직성, 설치의 복잡성을 개선하기 위해 DARPA를 중심으로 1997년부터 활발한 연구가 진행되어왔다. 액티브 네트워크는 노드 구조를 개방형으로 설계하여 프로토콜의 표준화에서 개발, 설치 및 서비스 단계까지의 기간을 단축시킴으로써 네트워크 서비스 실현에 새로운 개념을 정립하였다. 기존의 네트워크 노드가 단순히 패킷을 저장 후 포워딩하는 식의 단순한 네트워킹 기능을 하는 것과는 달리 액티브 네트워크는 사용자가 원하는 프로그램을 패킷을 통해 전송하여 실행하거나 미리 설치된 프로그램 중 해당 기능을 이용하여 패킷을 처리해주는 기능을 가진다.



(그림 1) 액티브 네트워크 노드 구조

액티브 네트워크의 노드 구조는 (그림 1)과 같이 노드 운영체제, 실행환경 그리고 액티브 응용으로 구성되어 있다. 노드 운영체제는 액티브 노드의 자원을 할당 및 보호하는 역할을 담당하며 실행 환경은 액티브 응용이 정상적으로 수행되기 위한 런타임 환경을 제공한다. 그리고 액티브 응용은 기존의 프로토콜에 해당하는 것으로 액티브 네트워크에서 필요한 프로토콜 서비스를 제공한다. 액티브 네트워크는 이러한 노드 구조를 바탕으로 패킷 내에 코드를 포함한 액티브 패킷을 전달함으로써 정적인 네트워크에 지능을 부여한다[2]. 하지만 기존의 실행환경은 프로토콜의 계층화 및 조합 그리고 단대단 응용 계층 서비스와의 연동에 있어서 그 한계점을 보이고 있다. 먼저 액티브 응용은 독립적으로 수행되기 때문에 계층화를 통한 프로토콜 개발 및 수정의 용이성을 기대하기 어려우며 상호참조를 바탕으로 새로운 프로토콜을 개발할 수 없다. 그리고 단대단 응용계층에서 액티브 네트워크를 사용하기 위한 표준화된 인터페이스를 정의하고 있지 않기 때문에 액티브 네트워크로의 접근이 어렵다. 이러한 문제의 해결을 위하여 프로토콜 합성 및 응용과의 연동을 위한 많은 연구가 진행되고 있다[3-5].

본 논문에서는 프로토콜의 조합을 통하여 소프트웨어의 재사용성을 높이고 서비스 인터페이스를 통하여 쉽게 액티브 네트워크에 접근할 수 있는 새로운 실행환경인 Customizable Architecture for Flexible Execution Environment (CAFEs)를 제안한다. 2장에서는 기존의 실행환경인 Active Network Transfer System(ANTS)과 Active Signaling Protocol(ASP)의 구조에 대한 장단점 분석하여 새로운 실행환경의 요구사항을 정리하고 3장에서는 제안하는 실행환경의 구조를 설명하며 4장에서 실제 구현 부분에 대하여 기술한 후 5장에서 결론을 맺는다.

2. CAFEs의 요구사항 정의

2.1 기존 실행환경 분석

2.1.1 ANTS(Active Network Transfer System) 실행환경

ANTS 실행환경은 Java oriented Active Network Operating System(JANOS)[6]를 기반으로 동작한다. 탑재되는 액티브 응용은 캡슐 프로그래밍 모델을 바탕으로 개발되고 코드 분배 기법을 통하여 액티브 노드에 프로토콜을 설치한다. 프로토콜은 비슷한 역할을 수행하는 코드의 집합인 코드 그룹들로 구성되며 각각의 코드 그룹은 액티브 패킷에 해당하는 다수의 캡슐로 구성된다[7]. ANTS 실행환경의 장점은 분산 방식의 프로토콜 설치 모델을 바탕으로 간편한 프로토콜 관리방안을 제시하고 있으며 쉽게 네트워크 정보 및 가상 네트워크 채널에 접근할 수 있다. 그리고 네트워크 트래픽 폭주를 제어할 수 있는 방법을 제공한다. 반면에 인증되지 않은 액티브 응용이 네트워크에 설치될 위험성이 높고 노드에 설치할 수 있는 코드의 크기가 제한됨에 따라 복잡한 형태의 프로토콜의 개발 및 분배가 어려우며 설치된 액티브 응용들은 완전히 분리되어 동작함으로써 상호간의 연동 및 조합이 불가능한 단점이 있다.

2.1.2 ASP(Active Signaling Protocol) 실행환경

ASP 실행환경은 특정 노드운영체제 없이 동작하며 탑재되는 액티브 응용은 프로토콜 프로그래밍 인터페이스에서 제공하는 채널을 통하여 실행환경 및 다른 액티브 응용과 데이터를 교환할 수 있다. 각각의 액티브 응용은 영속적으로 수행되는 Persistent 액티브 응용과 필요에 따라 일시적으로 수행되는 Transient 액티브 응용으로 구분된다. 보안관리자는 접근제어 정책파일과 액티브 응용이 로딩될 때

설정된 보안등급을 기반으로 액티브 응용의 동작을 제한한다[8]. ASP 실행환경의 장점은 통신채널을 통하여 액티브 응용들 사이의 정보교환이 가능하고 가상 네트워크 채널이 하나의 프로토콜 계층에 종속적이지 않으므로 다양한 계층을 기반으로 확장할 수 있다. 그리고 접근제어정책을 사용하여 액티브 응용의 실행환경 내부에 대한 정보접근을 제어할 수 있으며 액티브 응용에서 사용하는 코드의 공유를 통하여 필요한 전체 메모리의 양을 줄일 수 있다. 단점은 시스템 컨피규레이션이 복잡하며 접근제어정책이 경직된 구조로 작성되어 확장이 불가능하다. 그리고 설치된 액티브 응용들 사이의 정보교환 방식이 복잡하여 새로운 프로토콜의 개발이 어렵다.

2.2 CAFEs 실행 환경의 요구사항 도출

기존 실행환경의 분석을 통하여 도출한 CAFEs의 요구사항은 크게 네 가지로 요약할 수 있다. 첫째, 실행환경은 런타임에 다양한 액티브 응용이 설치, 삭제 및 실행되는 동적인 환경이므로 관리의 용이성과 액티브 응용의 개발부터 탑재 및 실행에 이르기까지 단순한 절차와 인터페이스를 요구한다[9]. 둘째, 실행환경에 설치되어 있는 액티브 응용을 조합 및 참조함으로써 새로운 서비스를 제공할 수 있는 유연한 구조를 지녀야 한다. 또한 프로토콜 및 서비스를 재조립 또는 참조함으로써 새로운 네트워크 서비스를 제공할 수 있어야한다[10-12]. 셋째, 플랫폼 독립적인 실행환경을 개발함으로써 높은 확장성을 제공해야한다. 즉, 특정 플랫폼에 독립적으로 동작하고 적은 메모리 소모량과 다양한 기존 프로토콜 계층에 접근함으로써 높은 이식성을 확보할 수 있어야한다. 마지막으로 인증, 접근제어 및 암호화 등을 통하여 액티브 응용의 탑재, 실행 및 관리의 안전성을 확보할 수 있어야하며 시스템 자원을 보호할 수 있어야한다[13]. 이렇게 도출된 요구사항을 기반으로 새로운 프로토콜 및 서비스의 개발을 용이하게 하고 조합 및 상호참조를 통하여 프로토콜의 개발 및 배포에 대한 새로운 방법인 CAFEs 실행환경의 상세한 구조에 대해서는 3장에서 다룰 것이다.

<표 1>은 기존의 실행환경인 ANTS와 ASP 그리고 새로운 제안하는 CAFEs 실행환경을 비교 분석한 것이다. CAFEs는 기존의 실행환경의 장점을 모두 수용하고 단점을 개선하였다. CAFEs는 간단한 시스템 컨피규레이션, 간편한 액티브 노드 관리 그리고 액티브 응용의 개발부터 탑재 및 실행에 이르기까지 단순한 절차와 쉬운 실행환경 인터페이스를 제공함으로써 액티브 시스템 관리자 및 응용 개발자에게 편리한 실행환경을 제공한다. 특히 기존 프로토콜의 기능 또는 알고리즘을 구현한 마이크로 프로토콜은 기존 응용과 호환하고 액티브 네트워크 서비스를 배포하기 위한 수단으로 동작하는 액티브 서비스라는 새로운 개념을 제시함으로써 마이크로 프로토콜 및 액티브 서비스의 조합 및 상호 참조를 통하여 프로토콜 및 서비스 소프트웨어의 재사용성을 높였다.

〈표 1〉 실행환경 비교 분석

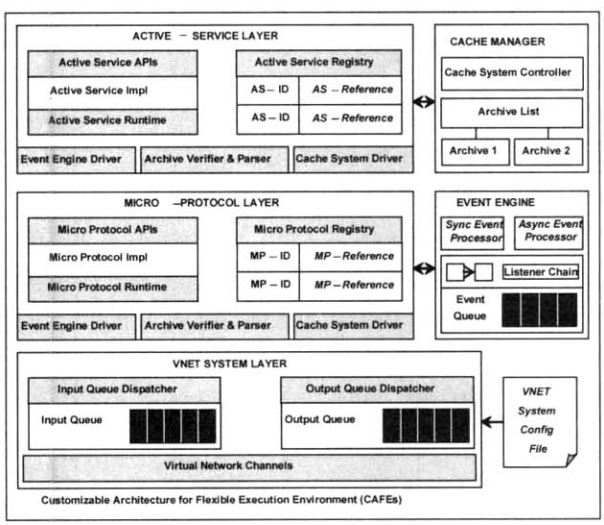
	ANTS	ASP	CAFEs
Multiple Protocol Layer Access	X	O	O
Easy of Use, Management, Development	Middle	Hard	Easy
Software Reusability	X	△	O
Protocol Customization	X	Slightly Support	O
Legacy Application Interoperability	X	O	O
Security	△	△	O

3. 맞춤형 통신 프로토콜과 서비스를 위한 액티브 실행환경 구조

맞춤형 통신 프로토콜과 서비스를 위한 액티브 실행환경인 CAFEs 실행환경은 (그림 2)와 같이 가상적인 하부 네트워크를 구성하기 위한 VNET 시스템 계층과 이를 기반으로 네트워크 서비스를 제공하기 위한 마이크로 프로토콜 계층 그리고 응용과의 연동을 위한 액티브 서비스 계층으로 구성되어 있다. **VNET 시스템 계층**은 네트워크, 전송 및 응용 계층 등의 다양한 프로토콜 계층을 기반으로 추상적인 가상 네트워크 채널을 구성 및 관리하고 시스템 공통 입·출력 큐와 연계하여 액티브 프레임의 송·수신 및 네트워크 정보를 제공함으로써 마이크로 프로토콜 계층에 대하여 하부 네트워크에 대한 추상화된 모델을 제공하는 역할을 담당한다. **마이크로 프로토콜 계층**은 마이크로 프로토콜 컴포넌트를 등록 및 해지하고 등록된 마이크로 프로토콜들을 조합하기 위한 참조자 및 API를 관리함으로써 마이크로 프로토콜의 조합을 바탕으로 새로운 프로토콜을 액티브 서비스 계층에 제공하기 위한 기반을 마련한다. 마지막으로 **액티브 서비스 계층**은 기존의 응용과 액티브 네트워크 사이의 연동을 위한 계층으로서 응용에서 마이크로 프로토콜 계층에 쉽게 접근할 수 있는 통로의 역할을 수행하며 동시에 응용의 개발에 필요한 라이브러리의 관리를 담당한다. 그 외 실행환경을 구성하는 컴포넌트로는 캐쉬 관리자와 이벤트 엔진이 존재하며 **캐쉬 관리자**는 액티브 서비스와 마이크로 프로토콜을 파일 시스템에 설치 및 삭제하고 **이**

벤트 엔진은 실행환경에서 정의한 다양한 이벤트를 처리한다.

실행환경을 설계하는 대표적인 방법으로는 통합 방식과 분리 방식이 있다. 통합 방식은 중간노드에서 수행될 코드를 포함한 패킷을 전달하여 실행환경에서 이러한 코드를 실행시킴으로써 액티브 패킷을 처리하는 방식이다. 분리 방식은 액티브 노드의 로컬 영역에 패킷의 처리에 필요한 코드가 이미 설치되어 있고 각각의 액티브 패킷은 중간노드에서 자신을 처리할 수 있는 코드의 식별자를 전달하는 방식이다. 통합 방식은 패킷과 함께 전달할 수 있는 코드의 크기에 한계가 있으므로 복잡한 형태의 서비스를 제공할 수 없고 패킷에 포함된 코드의 크기만큼 불필요한 트래픽을 유발할 수 있다. 반면에 분리방식은 코드의 크기에 영향을 받지 않으며 패킷을 구성하는 방법이 간단하기 때문에 다양한 서비스를 제공할 수 있기 때문에 효율성 또한 높다. 그러므로 설치된 프로토콜 및 서비스를 재사용하여 새로운 네트워크 서비스를 제공하기 위한 CAFEs 실행환경은 분리 방식을 사용하여 설계하였다.



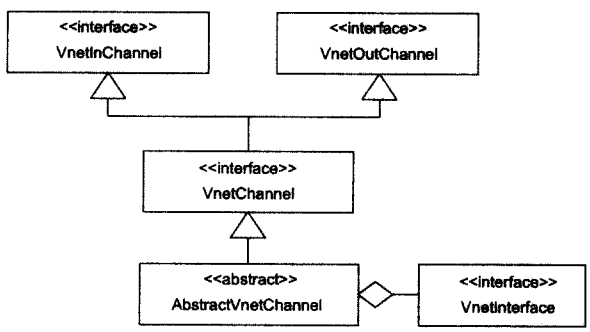
(그림 2) CAFEs 실행환경 구조

3.1 가상 네트워크 모델

액티브 네트워크에서 네트워크 모델에 대한 접근방식은 두 가지로 나누어진다. 첫 번째는 네트워크에 존재하는 모든 노드가 액티브 노드로 구성된 환경이며, 두 번째는 기존의 IP 노드와 액티브 노드가 공존하는 방식이다. 첫 번째의 경우 모든 네트워크 노드를 액티브 노드로 변경시키기 위해서는 많은 비용과 노력이 필요할 뿐만 아니라 IP 노드를 기반으로 동작하는 기존 응용과의 호환성에 대한 문제점이 발생한다. 그러므로 CAFEs 실행환경은 후자의 접근방식을 기반으로 논리적인 가상 네트워크 채널을 사용하여 액티브 노드 사이의 네트워크를 형성한다.

3.1.1 가상 네트워크 채널

가상 네트워크 채널은 기존 IP 네트워크에서 CAFEs 실행환경이 탑재된 액티브 노드사이에 네트워크를 형성하기 위한 추상화된 네트워크 인터페이스이다. 실행환경은 이것을 이용하여 구성된 논리적인 전용 네트워크인 오버레이 네트워크를 통하여 마이크로 프로토콜 계층에서 생성한 액티브 패킷을 다음 노드로 전달할 수 있다. 가상 네트워크 채널은 기본적으로 UDP를 사용하여 형성되며 전송계층의 다른 프로토콜인 TCP나 네트워크 계층의 IP 그리고 응용계층 등의 다양한 기존의 프로토콜 계층을 기반으로 채널 구성을 확장할 수 있다.



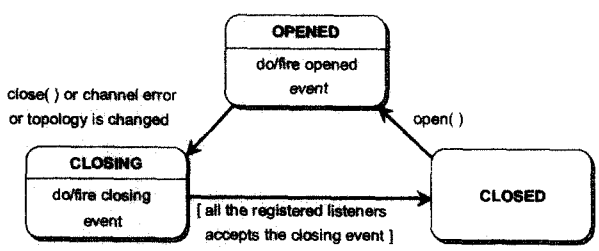
(그림 3) 채널 클래스 다이어그램

(그림 3)의 가상 네트워크 채널은 액티브 패킷의 송·수신을 위한 입·출력 채널로 구분되며 네트워크 인터페이스 정보를 제공하는 VnetInterface를 포함하고 있다. 채널 구현부는 의존하는 기존 네트워크 프로토콜 계층을 바탕으로 형성되지만 외부에는 VnetChannel 인터페이스만을 제공함으로써 상위 계층에서는 실제 가상 네트워크 채널의 구현에 관계없이 액티브 패킷을 송수신하며 네트워크 정보를 얻을 수 있다. 이러한 구조를 가진 가상 네트워크 채널은 마이크로 프로토콜 계층에서 전달된 액티브 패킷을 액티브 프레임으로 캡슐화하여 다음 노드로 포워딩하고 이웃 노드로부터 입력된 액티브 프레임을 원본 액티브 패킷으로 전환한다. 액티브 프레임은 가상 네트워크 채널이 의존하고 있는 기존의 프로토콜 계층에서 사용하는 패킷을 구현한 것으로 실제 IP 노드에서 라우팅 가능한 패킷을 의미한다. 가상 네트워크 채널이 제공하는 API는 크게 세 부분으로 나누어질 수 있다. 첫 번째는 실행환경에서 생성된 채널의 개방 및 폐쇄를 위한 API이다. 두 번째는 가상 네트워크 채널 및 네트워크 인터페이스에 대한 정보를 마이크로 프로토콜 계층에 전달하기 위한 API이며, 마지막으로 액티브

프레임을 이용하여 액티브 패킷의 송수신하기 위한 것으로 마이크로 프로토콜 계층에서 직접적으로 접근하는 것이 아니라 VNET 시스템 계층에서 마이크로 프로토콜 계층의 액티브 패킷 송수신에 대한 요청을 일괄적으로 처리하기 위한 API이다.

3.1.2 가상 네트워크 채널 상태전이 모델

가상 네트워크 채널은 액티브 노드를 관리하는 관리자에 의하여 강제적으로 개방 및 종료될 수 있으며 그 외 채널상의 오류 발생, 네트워크 토폴러지의 변화와 같은 경우 자동적으로 닫히게 되는데 이러한 동적인 상태 변화를 마이크로 프로토콜 계층에 전달할 수 있어야 한다. VNET 시스템 계층과 연계하여 동작하는 마이크로 프로토콜의 경우 현재 개방된 채널을 런타임에 발견할 수 있어야할 뿐만 아니라 채널의 상태변화를 동적으로 발견할 수 있는 기법을 필요로 한다. 그러므로 가상 네트워크 채널의 상태전이 모델 및 채널 상태변화에 따른 마이크로 프로토콜 계층에서의 이벤트 수신 방법을 정의하고 채널 참조자를 통하여 채널에 간접적으로 접근하는 방식을 제공해야 한다. (그림 4)는 가상 네트워크 채널의 상태 전이도로서 채널의 개방 및 폐쇄에 해당하는 OPENED와 CLOSED 상태 사이에 CLOSING 상태가 존재하며 채널이 완전히 폐쇄되기 전에 채널 종료 이벤트를 전송하기 위한 중간 단계로서 이벤트 전송을 완료하면 자동으로 CLOSED 상태로 전이된다.



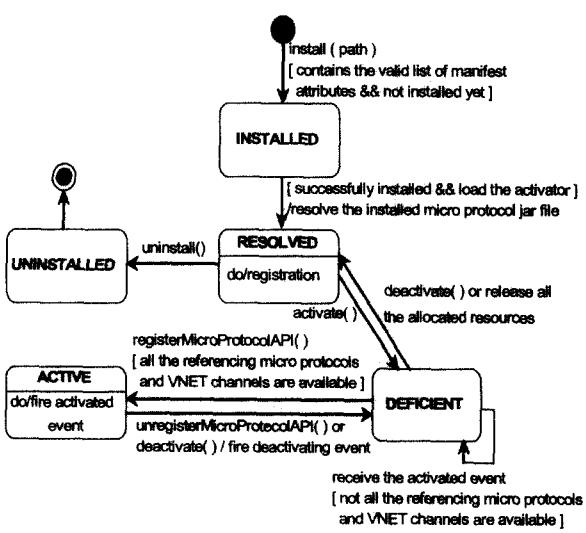
(그림 4) 가상 네트워크 채널 상태 전이도

3.2 프로토콜 및 서비스 모델

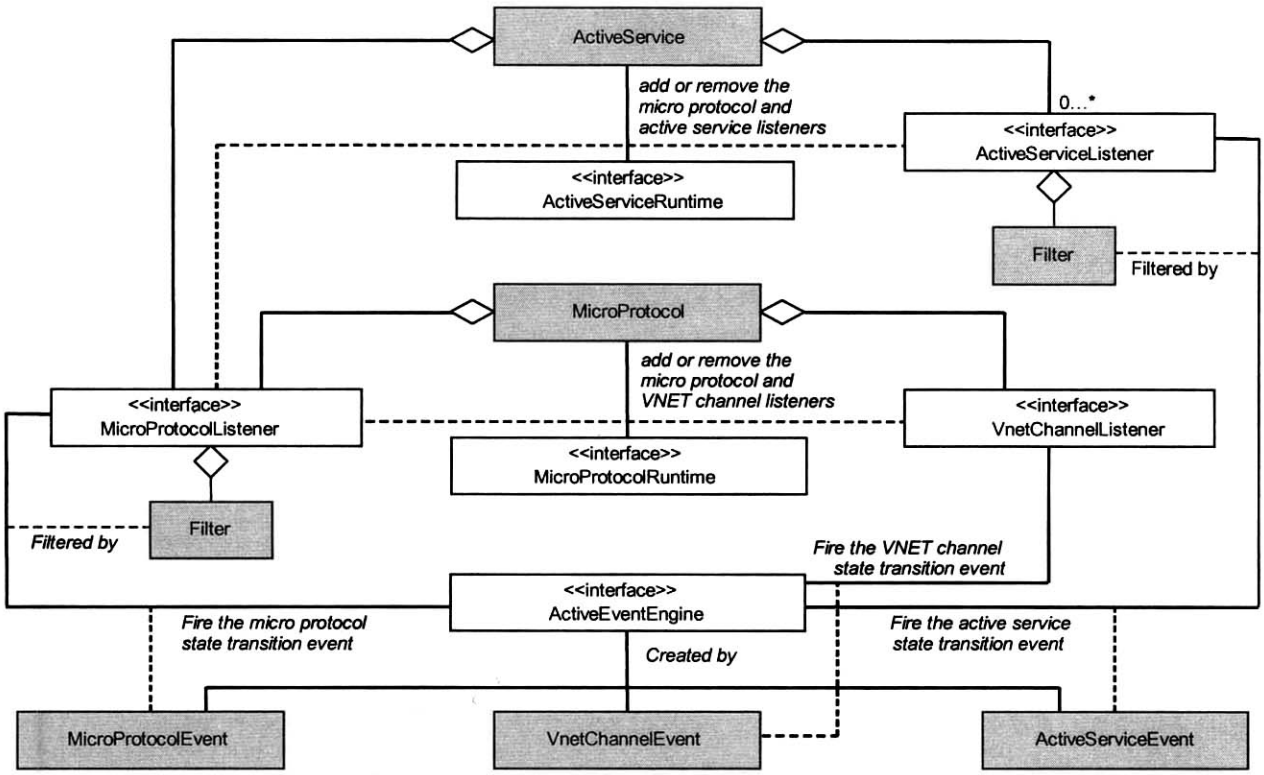
전통적인 프로토콜 개발방법은 비슷한 역할을 수행하는 프로토콜의 그룹화를 바탕으로 계층화하고, 각각의 프로토콜을 계층간의 인터페이스를 통하여 연동함으로써 서로 다른 시스템간의 통신을 원활히 하고 복잡한 프로토콜의 개발과정을 단순화시켰다. 하지만 이러한 개발방법은 특정 계층에 해당하는 프로토콜을 개발하는데 필요한 모든 기능을 하위 계층에서 완벽하게 지원하지 못한다. 그리고 모든 프로토콜이 하나의 완성된 소프트웨어 패키지로 구성되어 소프트웨어의 재사용성을 떨어뜨리는 문제점을 가지고 있다. 그러므로 기존의 프로토콜을 기능별로 분할하여 개발한 마이크로 프로토콜과 라이브러리 설치 및 응용과의 연동에 필요한 액티브 서비스를 조합 및 참조함으로써 새로운 프로토콜 및 서비스를 개발하기 위한 모델이 필요하다.

3.2.1 마이크로 프로토콜 및 액티브 서비스 상태전이 모델

CAFEs 실행환경은 런타임에 새로운 마이크로 프로토콜 및 액티브 서비스가 설치, 실행 및 삭제되는 동적인 환경을 지향한다. 이러한 환경에서 조합 및 참조를 통한 새로운 프로토콜과 서비스를 제공하기 위해서는 마이크로 프로토콜과 액티브 서비스가 내외부적인 수행관련 요소를 바탕으로 자신의 수행상태를 결정하기 위한 상태전이 모델이 필요하다. (그림 5)는 마이크로 프로토콜의 상태전이 모델을 기술한 것으로 액티브 서비스도 이와 동일한 상태전이 모델을 가진다. 마이크로 프로토콜과 액티브 서비스는 다섯 가지의 동일한 상태변수를 가지는데, 가장 중요한 ACTIVE 및 DEFICIENT 상태는 조합 및 참조가능 여부를 결정한다. 마이크로 프로토콜의 경우 ACTIVE 상태는 자신의 참조중인 가상 네트워크 채널과 다른 마이크로 프로토콜이 모두 정상적으로 수행되고 있으며 자신 또한 정상적으로 동작중인 상태를 말한다. DEFICIENT 상태는 참조 대상이 활성화되지 않아 대기상태에 있는 것을 의미한다. 액티브 서비스의 경우는 활성화를 위하여 마이크로 프로토콜 및 다른 액티브 서비스를 요구하며 참조 대상이 정상 동작 중이면 ACTIVE 상태로 전이하고 그렇지 않을 경우 DEFICIENT 상태에서 대기한다. 그 외 INSTALLED는 마이크로 프로토콜 및 액티브 서비스가 실행환경에 탑재 완료된 상태이며 RESOLVED는 활성화 명령을 받기위한 준비단계이다.



(그림 5) 마이크로 프로토콜 상태 전이도



(그림 6) 상태전이 감지 모델

3.2.2 상태전이 감지 기법

가상 네트워크 채널, 마이크로 프로토콜 및 액티브 서비스는 자신의 내부적인 수행환경 및 외부적인 의존 요소를 바탕으로 자신의 상태가 전이된다. 그러므로 마이크로 프로토콜은 자신이 참조하는 가상 네트워크 채널 및 조합하고 있는 다른 마이크로 프로토콜의 상태 변화를 동적으로 감지할 수 있어야하며 액티브 서비스의 경우 참조하는 마이크로 프로토콜 및 액티브 서비스에 대한 상태 변화를 알아낼 수 있어야 한다. 예를 들면, 특정 가상 네트워크 채널이 개방 또는 폐쇄될 경우 그 채널을 통하여 액티브 패킷을 송·수신하던 마이크로 프로토콜은 채널의 상태가 변화함에 따라 네트워크 인터페이스 접근에 대한 변경이 필요하다. 특정 마이크로 프로토콜의 상태가 활성화 또는 비활성화될 경우 동일한 기능의 다른 마이크로 프로토콜을 찾아 재조합 과정을 수행하거나 재조합이 불가능한 경우 자신의 상태를 비활성화로 변경시켜야 한다. 액티브 서비스 또한 참조중인 마이크로 프로토콜이나 액티브 서비스의 상태변화에 따라 자신의 상태를 재정립하여야한다.

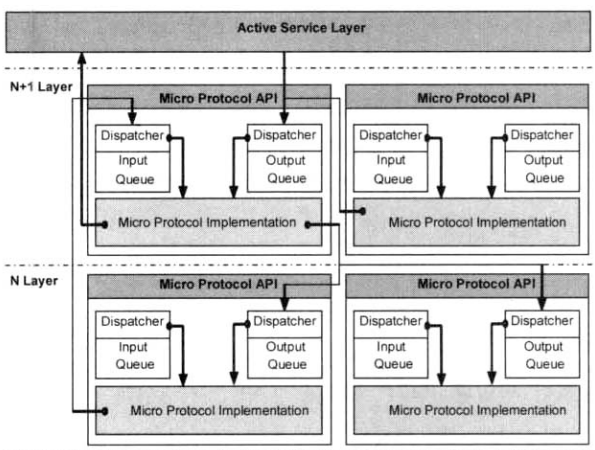
(그림 6)은 가상 네트워크 채널, 마이크로 프로토콜 및 액티브 서비스의 동적인 상태전이 감지 모델이다. 마이크로 프로토콜은 가상 네트워크 채널 및 다른 마이크로 프로토콜의 상태변화를 감지하기 위하여 VnetChannelListener와 MicroProtocolListener를 등록한다. 특히 조합하고자 하는 마이크로 프로토콜의 상태변화를 감지할 경우 필터를 사용하여 원하는 마이크로 프로토콜에 대한 리스너를 등록할 수 있다. 액티브 서비스의 경우 마이크로 프로토콜과 액티브 서비스에 대하여 MicroProtocolListener와 ActiveServiceListener를 필터와 함께 등록한다. 그 후 리스너에서 감지중인 이벤트가 발생할 경우 리스너의 콜백 메소드가 호출된다. 리스너를 등록한 마이크로 프로토콜이나 액티브 서비스는 메소드의 인자로 전달된 이벤트를 분석하여 참조대상의 상태변화를 동적으로 감지할 수 있다. 그리고 등록한 필터를 통하여 자신의 조합 및 참조 대상에 대한 상태변화만을 확인할 수 있다.

3.2.3 마이크로 프로토콜 조합

마이크로 프로토콜은 기존의 패키지화된 프로토콜과는 달리 프로토콜의 특정 기능 또는 알고리즘을 구현한 것이다. 그러므로 특정 네트워크 서비스를 제공하기 위해서는 관련된 마이크로 프로토콜을 조합하여 캡슐화함으로써 하나의 패키지를 구성할 수 있다. 이전의 프로토콜이 특정 계층을 타겟으로 하여 인접한 계층의 API를 이용하여 네트워크 서비스를 제공하는 반면에 마이크로 프로토콜은 자신이 구현하고 있는 네트워크 서비스를 이미 개발된 마이크로

프로토콜을 동적으로 조합하여 응용에서 원하는 형태의 주문형 프로토콜을 제작하는 것이다.

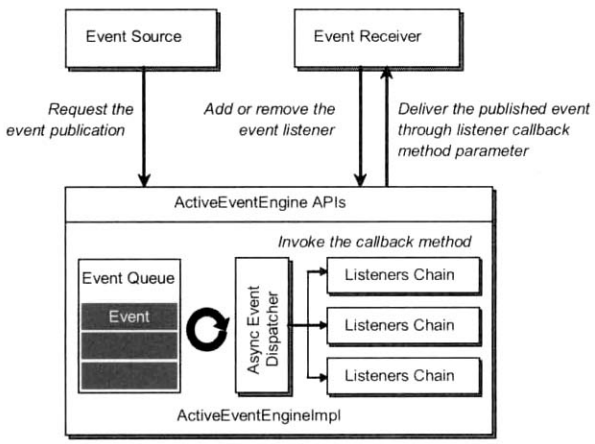
(그림 7)은 프로토콜을 분해하여 각각의 기능별로 개발하여 실행환경에 설치하고 새로운 네트워크 서비스를 제공하고자 할 경우 기존에 개발된 마이크로 프로토콜을 재조립함으로써 개발기간의 단축시키고 응용계층에서 원하는 네트워크 서비스를 완벽하게 제공하기 위한 조합 모델을 기술한 것이다. 기존의 프로토콜 개발 방법론은 인접 계층의 API를 이용하여 하부 프로토콜 계층의 기능을 추상화함으로써 프로토콜 개발을 간편하게 하고 계층간의 원활한 통신을 제공한 반면에 마이크로 프로토콜의 조합을 이용한 프로토콜 개발 방법론은 마이크로 프로토콜이 제공하는 API를 인접계층의 존재와 상관없이 자유롭게 호출함으로써 응용이 원하는 형태의 프로토콜 계층화가 가능하며, 응용에서 요구하는 네트워크 서비스를 기능 또는 알고리즘에 따라 개발된 마이크로 프로토콜을 동적으로 조합함으로써 소프트웨어의 재사용성을 높일 수 있다. 액티브 서비스의 조합 모델 역시 이와 동일하다.



(그림 7) 마이크로 프로토콜 조합 모델

3.3 이벤트 처리 모델

CAFEs 실행환경은 (그림 8)과 같은 이벤트 위임(delegation) 모델을 기반으로 실행환경에서 발생하는 모든 이벤트를 처리한다. 이벤트 위임 모델은 이벤트 생성자와 수신자 사이에 존재하는 중재자를 통한 간접적인 처리 방법으로써 이벤트 처리에 대한 권한을 중재자에게 위임하는 방식이다. 이벤트 발생 요청은 중재자에서 제공하는 API 호출을 통하여 이루어지며 수신은 특정 이벤트 발생에 대한 콜백 메소드를 포함하고 있는 리스너를 통하여 이루어진다. 즉, 수신자는 리스너를 중재자에게 등록하고 생성자는 특정 이벤트가 발생할 경우 역시 중재자에게 이벤트 발생 요청을 전달한다. 요청을 받은 중재자는 리스너 체인에 연결된 각각의 리스너에 존재하는 콜백 메소드를 호출함으로써 이벤트 처리를 완성한다. 이러한 방식은 이벤트 생성자와 수신자의 입장에서 처리 모델 자체가 간단하여 사용하기 쉽고 리스너의 등록 및 해지를 통하여 다수의 이벤트를 수신할 수 있을 뿐만 아니라 하나의 이벤트를 여러 수신자에게 전달할 수 있는 장점이 있다.



(그림 8) 이벤트 위임 모델

CAFEs 실행환경에서는 이러한 이벤트 처리 모델을 기반으로 중재자의 역할을 수행하는 이벤트 엔진과 상태전이 감지 기법을 연계하여 가상 네트워크 채널, 마이크로 프로토콜 및 액티브 서비스의 상태변화 이벤트를 처리한다. 각각의 마이크로 프로토콜과 액티브 서비스는 자신이 참조 또는 조합중인 가상 네트워크 채널, 마이크로 프로토콜 또는 액티브 서비스에 대한 상태변화 리스너를 필터와 함께 이벤트 엔진의 리스너 체인에 등록한다. 그리고 이벤트 엔진은 각각의 상태변화에 대한 처리요청을 받아들여 이벤트 큐에 해당 이벤트를 생성하여 저장한다. 큐와 관련되어 동작하는 이벤트 분배자는 큐로부터 이벤트를 출력하여 해당하는 리스너 체인을 검색한다. 마지막으로 리스너에 등록된 필터를 통하여 관련 이벤트에 대한 필터링을 수행한 후 리스너의 콜백 메소드를 호출함으로써 이벤트 전달을 완료한다.

4. CAFEs 실행환경 구현

4.1 VNET 시스템 관리자

CAFEs 실행환경의 구현부는 가상 네트워크, 마이크로 프로토콜 및 액티브 서비스 각각을 구성하고 관리하기 위한 VNET 시스템 관리자, 마이크로 프로토콜 관리자 그리고 액티브 서비스 관리자 컴포넌트로 구성된다. 그 중에서 VNET 시스템 관리자는 실행환경의 최하위 계층에 속하는

VNET 시스템 계층을 통합 관리하는 컴포넌트로서 하부 네트워크에 대한 논리적인 인터페이스를 제공하는 가상 네트워크 채널을 구성 및 관리한다. 그리고 동시 다발적인 액티브 패킷의 송수신을 조정하기 위한 시스템 공통 입출력 큐를 관리함으로써 마이크로 프로토콜 계층과 가상 네트워크 채널사이에서 액티브 패킷의 송수신에 대한 중재자의 역할을 수행한다.

〈표 2〉 가상 네트워크 채널 컨피규레이션 프로파일 XML 스키마

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd : schema targetNamespace="http : //ccmc.knu.ac.kr/xml/Acti-
      veNet work/CAFEs"
 xmlns="http://ccmc.knu.ac.kr/xml/ActiveNetwork/CAFEs"
       elementFormDefault="qualified"
 xmlns : xsd="http://www.w3.org/2001/XMLSchema">
  <xsd : element name="VnetSystem">
    <xsd : complexType>
      <xsd : element name="VnetChannelList">
        <xsd : complexType>
          <xsd : element name="VnetChannel" minOccurs="1"
                maxOccurs="unbounded">
            <xsd : complexType>
              <xsd : sequence>
                <xsd : element name="Bandwidth" type="xsd :
                      float" minOccurs="1" maxOccurs="1"/>
                <xsd : element name="MTU" type="xsd : positive-
                      Integer" minOccurs="1" maxOccurs="1"/>
                <xsd : element ref="VnetInterface"/>
              </xsd : sequence>
            </xsd : complexType>
          </xsd : element>
        </xsd : complexType>
      </xsd : element>
    </xsd : complexType>
  </xsd : element>

  <xsd : element name="VnetInterface" minOccurs="1" maxOc-
   curs="1">
    <xsd : complexType>
      <xsd : sequence>
        <xsd : element name="VnetInterfaceType" type="xsd :
         string" minOccurs="1" maxOccurs="1"/>
        <xsd : element name="SourceAddress" type="xsd : string"
         minOccurs="1" maxOccurs="1"/>
        <xsd : element name="DestAddress" type="xsd : string"
         minOccurs="1" maxOccurs="1"/>
        <xsd : element name="SourcePort" type="xsd : positive-
         Integer" minOccurs="0" maxOccurs="1"/>
        <xsd : element name="DestPort" type="xsd : positiveInte-
         ger" minOccurs="0" maxOccurs="1"/>
      </xsd : sequence>
    </xsd : complexType>
  </xsd : element>
</xsd : schema>
```

4.1.1 가상 네트워크 채널 컨피규레이션

기존에 개발된 액티브 네트워크 실행환경은 특정 프로토콜 계층에 종속적인 채널을 제공한다. 이러한 방식은 실행환경의 하부 네트워크 계층이 특정 프로토콜 계층에 의존하게 되어 현재 의존하는 프로토콜 계층 이하에 접근할 수 없으므로 다양한 프로토콜 계층 기반의 가상 네트워크를 제공할 수 없게 된다. 그러므로 가상 네트워크 채널은 전송계층 뿐만 아니라 네트워크 계층 그리고 응용계층을 기반으로 생성될 수 있어야 한다. VNET 시스템 관리자 컴포넌트는 가상 네트워크 채널 컨피규레이션을 위하여 정의된 XML 스키마를 바탕으로 기술된 프로파일을 분석하여 액티브 네트워크 채널을 구성한다. 가상 네트워크를 위한 XML 스키마는 특정 프로토콜 계층에 종속적인 채널을 기술하는 것이 아니라 다양한 계층을 기반으로 가상 네트워크를 구축할 수 있도록 구성되어 있다.

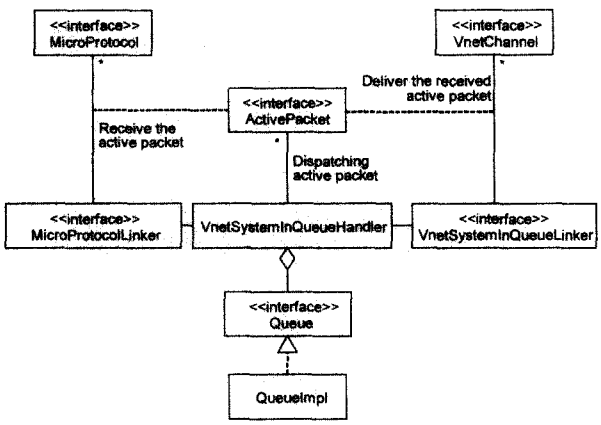
<표 2>는 채널 컨피규레이션 프로파일 XML 스키마를 나타낸 것으로 최상위 요소인 VnetSystem은 유일한 자식 요소인 VnetChannelList를 가지고 있다. VnetChannelList 요소는 각각의 가상 네트워크 채널과 대응되는 다수의 VnetChannel 요소를 가지고 있으며 VnetChannel 요소는 가상 네트워크 인터페이스에 해당하는 VnetInterface 요소를 포함하고 있다. VnetInterface는 현재의 채널이 의존하고 있는 프로토콜 계층에 의존적인 정보를 표현하기 위한 것으로써 프로토콜 계층 식별자를 의미하는 VnetInterfaceType 요소와 네트워크 인터페이스 자체에 대한 정보를 기술하기 위한 요소를 모두 포함하고 있다. VNET 시스템 관리자는 스키마를 따르는 가상 네트워크 컨피규레이션 프로파일을 분석하여 논리적인 가상 네트워크 채널을 생성함으로써 오버레이 네트워크를 형성한다.

4.1.2 시스템 공통 큐 관리 및 제어

VNET 시스템 관리자는 다수의 가상 네트워크 채널을 통하여 입력된 액티브 패킷을 VNET 시스템과 접합된 여러 마이크로 프로토콜로 전달하고 마이크로 프로토콜로부터 생성된 액티브 패킷을 입력받아 다음 노드로 포워딩하기 위한 중재자의 역할을 담당한다. 시스템에서 동작하는 각각의 가상 네트워크 채널과 마이크로 프로토콜은 모두 동시 다발적으로 액티브 패킷을 송수신하게 되므로 특정 마이크로 프로토콜이 시스템의 채널을 독점하거나 특정 채널에 동시에 접근함으로써 발생할 수 있는 충돌현상을 방지 할 수 있어야 한다. 그러므로 VNET 시스템 관리자는 공통 입출력 큐와 액티브 패킷 분배자를 통하여 동기 방식의 액티브 패킷 송수신 기능을 제공한다.

(그림 9)는 VNET 시스템의 공통 입력큐 및 핸들러를 이용하여 마이크로 프로토콜과 가상 네트워크 채널 사이의 액티브 패킷 수신방법을 기술한 것이다. 먼저 VnetChannel로부터 입력된 액티브 패킷은 VnetSystemInQueueLinker

를 통하여 입력큐에 저장된다. 입력큐 핸들러는 Queue 인터페이스를 통하여 수신된 액티브 패킷에 접근하며 MicroProtocolLinker를 이용하여 실제 액티브 패킷이 수신되어야 할 마이크로 프로토콜에게 전달한다. 송신과정은 출력큐 및 핸들러를 사용하여 수신과정의 역순으로 액티브 패킷을 전송한다. 그리고 입·출력큐 핸들러가 입력된 액티브 패킷을 처리하는 방법은 모두 Queue 인터페이스를 통하여 수행되므로 Queue의 구현방식에 따라 다양한 분배 기법을 사용할 수 있다.



(그림 9) 액티브 패킷 수신 방법

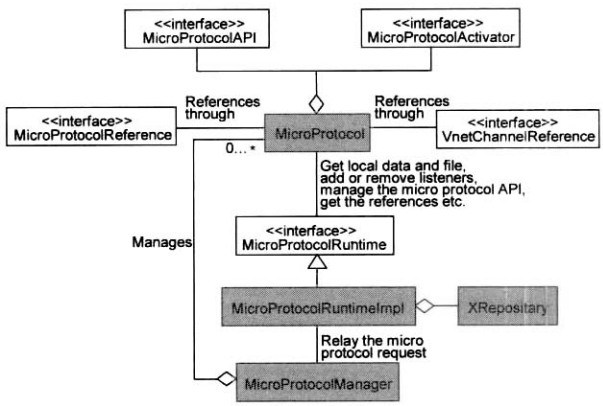
4.2 마이크로 프로토콜 및 액티브 서비스 관리자

마이크로 프로토콜 및 액티브 서비스 관리자는 CAFEs 실행환경에서 기존의 프로토콜을 세분화하여 모델링한 마이크로 프로토콜과 응용과 프로토콜을 연결하기 위한 액티브 서비스를 관리하기 위한 컴포넌트이다. 기본적으로 각각의 마이크로 프로토콜과 액티브 서비스가 정상적으로 동작하기 위한 독립적인 런타임 환경을 지원하고 설치된 마이크로 프로토콜과 액티브 서비스의 조합 및 참조를 통하여 새로운 마이크로 프로토콜과 액티브 서비스를 개발하기 위한 방법을 제공한다.

4.2.1 마이크로 프로토콜 및 액티브 서비스 런타임 환경

마이크로 프로토콜 및 액티브 서비스 관리자는 실행환경에 탑재된 다양한 마이크로 프로토콜 및 액티브 서비스에 대하여 서로간의 비합법적인 간섭을 배재시키고 독립적으로 수행할 수 있는 환경의 제공을 위하여 런타임 인터페이스를 제공한다. (그림 10)는 마이크로 프로토콜에 해당하는 MicroProtocol 클래스와 마이크로 프로토콜 관리자를 표현하기 위한 MicroProtocolManager 클래스 및 MicroProtocolRuntime 인터페이스의 상관관계를 표현한 것이다.

런타임 인터페이스의 역할은 크게 두 가지로 나누어진다. 첫째, 마이크로 프로토콜이 런타임 인터페이스만을 통하여 시스템에 접근하도록 유도함으로써 내부 구현부를 격리시켜 소프트웨어의 의존성을 낮추고 정의되지 않은 다른 방법을 통한 시스템 접근을 봉쇄한다. 둘째, 각각의 마이크로 프로토콜에게 독립적인 런타임 인터페이스를 할당함으로써 조합 및 참조를 위하여 등록한 MicroProtocolAPI 및 ActiveServiceAPI 이외의 다른 방법을 통하여 현재 수행중인 마이크로 프로토콜 및 액티브 서비스에 접근할 수 없다. 액티브 서비스의 런타임 환경 역시 마이크로 프로토콜의 경우와 동일하다.



(그림 10) 마이크로 프로토콜 런타임 환경

<표 3>은 마이크로 프로토콜 및 액티브 서비스 런타임 인터페이스의 전체 API를 기술한 것이다. 각각의 런타임 인터페이스의 API는 크게 세 부분으로 나누어진다. 첫째, 임시 데이터 저장소 및 영구 데이터 저장소 접근을 위한 API이다. 마이크로 프로토콜과 액티브 서비스는 실행 중에 필요한 임시적인 데이터의 저장을 위하여 로컬 데이터 저장소가 필요하며 이것은 종료할 때 모두 삭제된다. 런타임 인터페이스는 로컬 데이터 저장소에 임시 데이터를 입·출력하기 위한 두 개의 메소드를 제공하며 키와 값의 쌍으로 저장소를 관리한다. 또한 영구 데이터 저장을 위해 파일 자원을 획득하기 위한 메소드를 제공하며 시스템 내부적으로는 각각의 마이크로 프로토콜과 액티브 서비스마다 고유의 파일을 저장하기 위한 경로가 분리되어 있다. 둘째, 자신이 참조하는 가상 네트워크 채널, 마이크로 프로토콜 및 액티브 서비스의 상태변화를 감지하기 위한 상태 변화 이벤트 리스너의 등록 및 해지 API이다. 마이크로 프로토콜은 자신이 필요로 하는 가상 네트워크 채널 및 다른 마이크로 프로토콜의 상태변화에 따라 자신의 상태 또한 변화하며 액티브 서비스의 경우 자신이 참조하는 마이크로 프로토콜과 다른 액티브 서비스의 상태 변화에 따라 자신의 정상적

인 활성화 상태를 결정한다. 그러므로 런타임 환경은 각각의 참조대상의 상태변화에 대한 이벤트를 수신할 수 있는 리스너의 등록 및 해지를 위한 API를 제공하고 있다. 셋째, 현재 활성화된 가상 네트워크 채널, 마이크로 프로토콜 및 액티브 서비스의 참조자 탐색 및 획득을 위한 API이다. CAFEs 실행환경에 탑재되는 마이크로 프로토콜과 액티브 서비스는 실행 시에 자신이 조합 및 참조하는 다른 가상 네트워크 채널, 마이크로 프로토콜 및 액티브 서비스를 요청한다. 그러므로 실시간에 조합 및 참조하는 참조대상에 대한 접근을 처리하기위한 API를 제공한다.

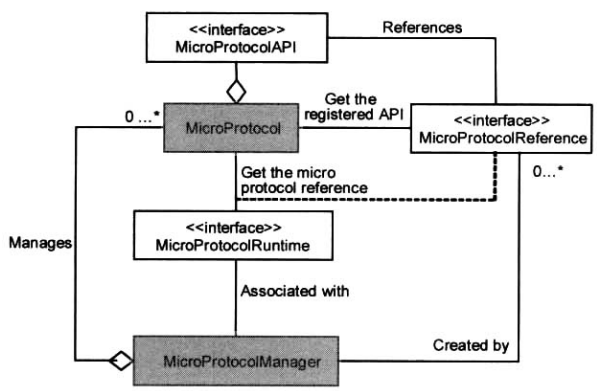
〈표 3〉 런타임 인터페이스 APIs

설 명	메 소 드
로컬 데이터 저장소 접근	public File getDataFile(String FileName) ; public Object getLocalData(Object Key) ; public void putLocalData(Object Key, Object Value) ;
참조 API 등록 및 해지	public void registerMicroProtocolAPI(MicroProtocolAPI API) throws IllegalStateException ; public void registerActiveServiceAPI(ActiveServiceAPI API) throws IllegalStateException ; public void unregisterMicroProtocolAPI( ) throws IllegalStateException ; public void unregisterActiveServiceAPI( ) throws IllegalStateException ;
이벤트 리스너 등록 및 해지	public void addVnetChannelListener(VnetChannelListener vcl) ; public void addMicroProtocolListener(MicroProtocolListener mpl) ; public void addMicroProtocolListener(MicroProtocolListener mpl, Filter f) ; public void addActiveServiceListener(ActiveServiceListener asl) ; public void addActiveServiceListener(ActiveServiceListener asl, Filter f) ; public void removeVnetChannelListener(VnetChannelListener vcl) ; public void removeMicroProtocolListener(MicroProtocolListener mpl) ; public void removeActiveServiceListener(ActiveServiceListener asl) ;
참조자 획득	public List getAllVnetChannelReference( ) ; public MicroProtocolReference getMicroProtocolReference(Identifier Id) ; public ActiveServiceReference getActiveServiceReference(Identifier Id) ;

4.2.2 마이크로 프로토콜 조합

프로토콜 조합은 기존 네트워크 프로토콜을 분해하여 기능 또는 알고리즘별로 개발하여 실행환경에 탑재한 마이크로 프로토콜을 바탕으로 새로운 네트워크 서비스를 제공하고자 할 경우 마이크로 프로토콜을 재조립함으로써 개발기간을 단축시키고 응용계층에서 원하는 네트워크 서비스를 제공하기 위한 방법이다. 기존의 프로토콜 개발 방법론은 인접 계층의 API를 이용하여 하부 프로토콜 계층의 기능을 추상화함으로써 프로토콜 개발을 간편하게 하고 계층간의 원활한 통신을 제공한 반면에 마이크로 프로토콜의 조합을 이용한 프로토콜 개발 방법론은 마이크로 프로토콜이 제공하는 API를 인접계층의 존재와 상관없이 자유롭게 호출함으로써 응용이 원하는 형태의 프로토콜 계층화가 가능하며 응용에서 요구하는 네트워크 서비스를 기능 또는 알고리즘에 따라 개발된 마이크로 프로토콜을 동적으로 조합함으로써 소프트웨어의 재사용성을 높일 수 있다.

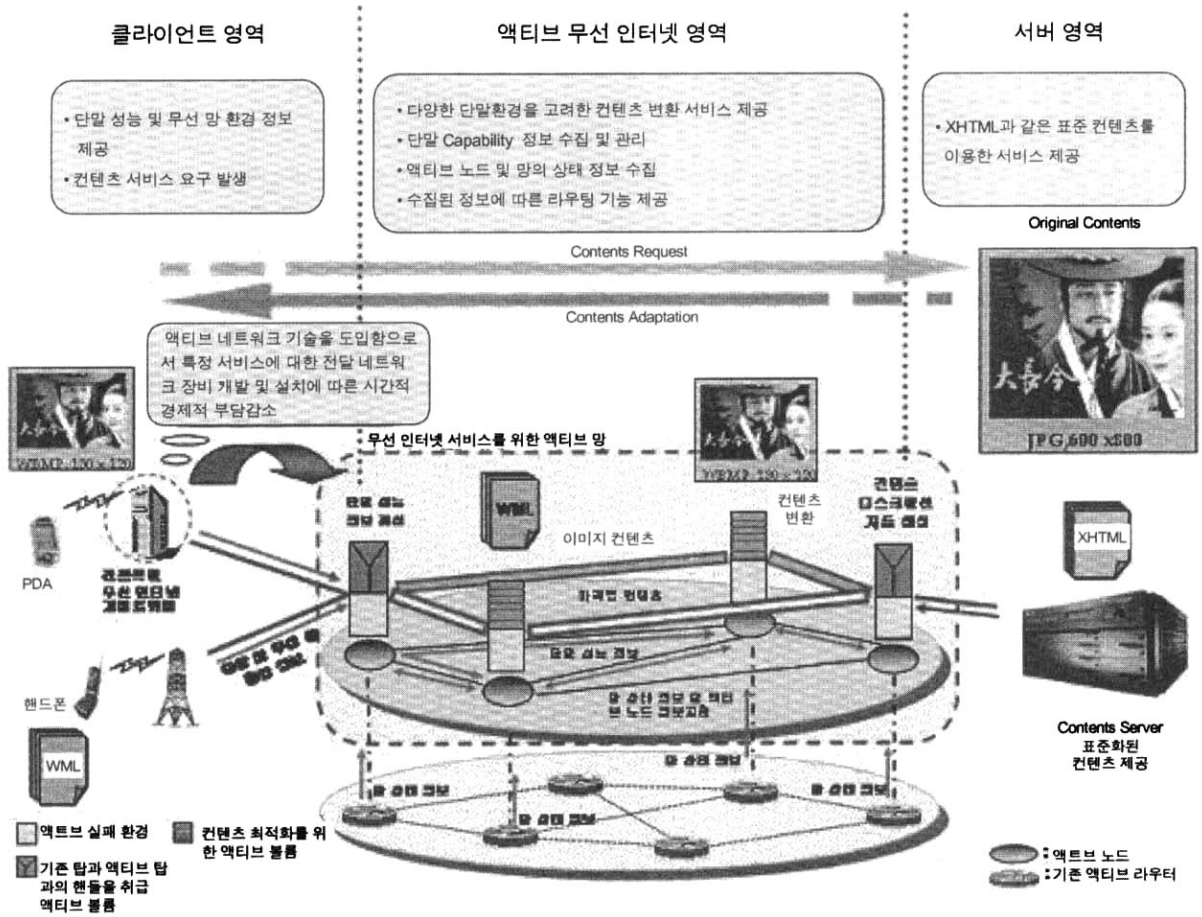
(그림 11)은 마이크로 프로토콜 조합모델에 대한 클래스 다이어그램을 기술한 것이다. 마이크로 프로토콜에 해당하는 MicroProtocol 클래스는 다른 마이크로 프로토콜과의 조합을 지원하기 위한 MicroProtocolReference 및 MicroProtocol-API 인터페이스를 가지며 인터페이스에 대한 실제 구현부는 감추어져 있다. 참조자 인터페이스는 해당 마이크로 프로토콜의 MicroProtocolAPI를 획득하기 위한 것으로 마이크로 프로토콜 관리자는 참조자를 얻을 수 있는 메소드를 런타임 인터페이스를 통하여 제공한다. 그리고 프로토콜 관리자는 개발된 모든 마이크로 프로토콜의 API를 레지스트리에 등록하여 통합적으로 관리하며 식별자를 통하여 레지스트리를 검색할 수 있다. 즉, 특정 마이크로 프로토콜은 자신의 네트워크 서비스를 구성하는데 필요한 다른 마이크로 프로토콜을 식별자를 이용하여 관리자로부터 참조자를 얻고 이것을 바탕으로 MicroProtocolAPI에 접근하여 응용에서 요구하는 네트워크 서비스를 구성한다. 액티브 서비스의 상호 참조를 통합 조합 기법 역시 마이크로 프로토콜의 그것과 동일한 방식으로 동작한다.



(그림 11) 마이크로 프로토콜 조합

4.3 실험환경 및 결과

현재 무선 인터넷에 대한 관심이 증가하면서 이동 단말의 보급률이 높아지고 무선 인터넷 서비스에 대한 요구가 지속적으로 증가하고 있으며, 이동 단말의 짧은 개발 주기와 각 단말간의 성능 차이가 커지면서 단말의 환경에 따라 독립적인 컨텐츠 서버를 구축하는 서비스 플랫폼은 시의 적절한 서비스를 제공하는데 많은 한계점을 노출한다. 그리고 다양한 이동 단말기의 성능 및 환경에 따라 동일한 컨텐츠 서비스를 제공하기 위해서는 단말에 맞는 컨텐츠 최적화 작업이 필요한데 WAP 게이트웨이를 두어서 서비스를 제공할 경우 사용자가 증가하면 게이트웨이에 부하가 집중되는 문제가 발생한다. 이러한 다양한 단말의 플랫폼 및 능력의 상이함과



(그림 12) 테스트 베드 환경

무선 망 사업자가 제공하는 인터넷 환경의 이질성으로 인하여 서비스 제공자는 각각에 대하여 모두 다른 컨텐츠 서버를 구축해야하는 어려움을 겪고 있다. 그러므로 본 절에서는 마이크로 프로토콜의 조합을 통하여 새로운 프로토콜을 개발하고 다양한 액티브 서비스 상호간의 참조를 통하여 기존의 수동적인 네트워크 노드에 지능을 부여하기 위한 CAFEs 실행환경과 단말을 능력을 기술하기 위한 Composite Capability/Preference Profile(CC/PP)를 이용하여 무선 웹 컨텐츠 전달 네트워크에 대한 실험환경 및 결과에 대하여 기술한다.

4.3.1 실험환경 구성

(그림 12)는 단말의 환경에 적합한 무선 웹 컨텐츠를 전달하기 위한 테스트 베드를 나타낸 것이다. 테스트 베드는 다양한 무선 단말이 존재하는 클라이언트 에지 네트워크와 컨텐츠 서버가 존재하는 서버 에지 네트워크가 양극에 존재한다. 중간 무선 웹 컨텐츠 전달 네트워크 부분에 CAFEs 실행환경이 탑재된 액티브 노드가 있다. 클라이언트 에지 네트워크에 존재하는 단말의 종류는 Wireless Application Protocol(WAP)을 사용하는 PDA와 HTTP를 사용하는 휴대단말로 구성되어 있다. 서버 에지 네트워크 구간에는 컨텐츠 서버에 해당하는 웹 서버가 존재한다. <표 4>은 무선 웹 컨텐츠 전달 네트워크를 구성하는 액티브 노드의 역할과 탑재되는 마이크로 프로토콜 및 액티브 서비스를 설명한 것이다.

4.3.2 동작과정

가. 웹 컨텐츠 요청절차

① 직접 또는 WAP 게이트웨이를 경유하여 AIN1의 WPAS에게 웹 컨텐츠 요청 전송

② AIN1의 WPAS는 RDTMP, VNRMP을 조합하여 AIN2의 WPAS로 요청을 포워딩

③ AIN2의 WPAS는 직접적으로 웹 컨텐츠 서버에 웹 컨텐츠 요청을 전송

나. 웹 컨텐츠 응답절차

① 웹 서버는 AIN2의 WPAS에 웹 컨텐츠 응답 전송

② AIN2의 WPAS는 RDTMP, DFCMP, SORMP을 조합하여 CTN2의 CTAS로 응답 포워딩

③ CTN2의 CTAS는 CC/PP-CAS를 통하여 단말의 능력을 확인하고 마크업 언어 및 이미지 크기 변환을 실행하고 결과를 ②와 동일한 프로토콜을 조합하여 CTN1의 CTAS로 전달

④ CTN1의 CTAS는 CC/PP-CAS를 참조하여 이미지 컬러 및 크기를 변환한 후 ②와 동일한 프로토콜을 조합하여 AIN1 노드의 WPAS로 변환된 컨텐츠 전달

⑤ AIN1의 WPAS는 최종 단말로 직접 또는 WAP 게이트웨이를 경유하여 컨텐츠 포워딩

〈표 4〉 마이크로 프로토콜 및 액티브 서비스

액티브 노드	
Active Interworking Node1 (AIN1)	클라이언트측 HTTP 프록시 단말의 프로파일을 기술한 CC/PP 정보 저장
Contents Transformation Node1 (CTN1)	단말이 지원하는 형태의 이미지 포맷 변환 임시 CC/PP 정보 캐쉬
Contents Transformation Node2 (CTN2)	단말에 적합한 이미지 크기 및 마크업 언어 변환 임시 CC/PP 정보 캐쉬
Active Interworking Node2 (AIN2)	서버측 HTTP 프록시
마이크로 프로토콜	
Virtual Network Routing Micro-Protocol (VNRMP)	가상 네트워크 채널에서 액티브 패킷을 라우팅하기 위한 마이크로 프로토콜
Service Oriented Routing Micro-Protocol (SORMP)	액티브 서비스를 기반으로 액티브 패킷을 라우팅하기 위한 마이크로 프로토콜
Reliable Datagram Transmission Micro-Protocol (RDTMP)	액티브 서비스 사이의 신뢰적인 데이터그램 전송을 위한 마이크로 프로토콜
Datagram Flow Control Micro-Protocol (DFCMP)	데이터그램 플로우 제어 마이크로 프로토콜
액티브 서비스	
Web Proxy Active-Service (WPAS)	웹 컨텐츠 요청 및 응답을 액티브 네트워크와 연동하기 위한 프록시 서비스
CC/PP Caching Active-Service (CC/PP-CAS)	단말의 CC/PP 정보 캐쉬를 위한 서비스
Contents Transformation Active-Service (CTAS)	단말의 환경에 맞게 컨텐츠를 변환하기 위한 서비스

4.3.3 실험결과

무선 웹 컨텐츠 전달 네트워크의 실험은 서로 다른 두 가지 최종 단말인 WAP 브라우저를 사용하는 PDA와 ME 브라우저 기반의 핸드폰을 사용하여 수행하였다. PDA는 Wireless Markup Language(WML)와 WBMP 흑백 이미지 포맷을 지원하며 핸드폰은 micro Hyper Text Markup Language(mHTML)와 GIF 컬러 이미지 포맷을 지원한다.

WAP 브라우저 기반의 PDA와 ME 브라우저를 사용하는 핸드폰 단말을 이용한 실험결과이다. 웹 서버에 XHTML과 JPG의 형태로 저장되어 있던 웹 컨텐츠가 최종 단말에 전달되는 과정에서 CAFEs 실행환경에 탑재된 다양한 마이크로 프로토콜 및 액티브 서비스들이 조합 및 참조를 통하여 단말의 형태에 적합한 마크업 언어 및 이미지의 크기, 컬러, 포맷을 변환하는 것을 확인할 수 있다.

5. 결론 및 향후연구

본 논문에서는 액티브 네트워크 개념을 도입하여 기존의 프로토콜을 기능별로 분할한 마이크로 프로토콜의 조합을 통해 새로운 프로토콜을 구성하고 액티브 서비스 상호참조를 이용하여 기존의 응용과 쉽게 호환함으로써 소프트웨어의 재사용성을 높일 수 있는 새로운 실행환경에 대하여 설명하였다. 또한 설명한 CAFEs 실행환경을 이용하여 현재 무선 인터넷에서 단말 및 네트워크의 이질성으로 인한 서비스 개발의 어려움을 해결하고자 노력하였다.

CAFEs 실행환경은 시스템의 분석 및 설계를 위하여 UML[14]을 사용함으로써 시스템의 모델링에 정형성을 부여하였다. 가상 네트워크를 위하여 설계된 가상 네트워크 채널은 다양한 프로토콜 계층을 기반으로 동작함으로써 실행환경의 네트워크 이질성을 극복하였다. 이를 바탕으로 프로토콜 및 서비스 모델을 설계하여 새로운 프로토콜 및 서비스의 개발에 대한 용이성을 확보하였다. 그리고 이벤트 위임 모델을 통하여 실행환경에 탑재되어 동작하는 마이크로 프로토콜 및 액티브 서비스의 연동방안을 정립하였다.

이러한 설계를 바탕으로 CAFEs 실행환경은 시스템의 결합도를 낮추고 응집력을 높이기 위하여 검증된 다양한 디자인 패턴[15]을 적용하여 구현함으로써 수정 및 유지보수 비용을 최소화하였고 플랫폼 독립적으로 개발함으로써 이식성을 높였다. VNET 시스템 관리자는 마이크로 프로토콜 관리자와 연동하여 마이크로 프로토콜에게 쉽게 네트워크 정보를 전달할 수 있다. 마이크로 프로토콜 및 액티브 서비스 관리자는 프로토콜 조합 및 서비스 상호참조에 대한 기법을 제공하고 마이크로 프로토콜 계층과 액티브 서비스 계층간의 접근을 용이하게 한다. 이벤트 엔진은 시스템 상태변화를 동적으로 감지할 수 있는 방안을 제공한다. 또한 개발된 CAFEs 실행환경을 사용하여 무선

인터넷에서 단말의 환경에 적합한 컨텐츠를 전달하기 위한 무선 웹 컨텐츠 전달 네트워크 테스트 베드를 구축하여 실험하였다. 이 실험에서는 사용자의 단말에 탑재된 웹 브라우저나 서비스 제공자의 웹 서버에 대한 수정 없이 단말에 적합한 무선 웹 컨텐츠 전달 서비스를 제공하기 위한 방법론을 제시함으로써 실행환경의 실용성을 확보하였다.

향후에는 실행환경에 존재하는 다양한 마이크로 프로토콜 및 액티브 서비스들 사이의 조합 및 참조에 대한 접근 제어 기술에 대한 연구가 이루어져야 한다. 또한 동적인 마이크로 프로토콜 및 액티브 서비스 설치 기법과 실행환경이 동작하는 액티브 노드의 원격관리 방법에 대한 연구가 필요하며 무선 웹 컨텐츠 전달 서비스 외에 다양한 적용분야에 대한 각종 연구가 이루어져야 한다.

참 고 문 헌

[1] Stephen F. Bush, Amit B. Kulkarni, Active Networks and Active Network Management, Kluwer Academic/Plenum Publishers, ISBN 0306465604, 2001.

[2] Tennenhouse, D. L., Wetherall, D. J., "Towards an Active Network Architecture," Computer Communication Review, Vol.26, No.2, April, 1996.

[3] S. Bhattacharjee, K. Calvert, Y. Chae, S. Merugu, M. Sanders, E. Zegura, "CANEs : an execution environment for composable services," Proceedings of DARPA Active Networks Conference and Exposition(DANCE'02), pp.255-272, May, 2002.

[4] Xiaodong Fu, Weisong Shi, Anatoly Akkerman, Vijay Karamcheti, "CANS : Composable, Adaptive Network Services Infrastructure," Proceedings of USENIX Symposium on Internet Technologies and Systems(USITS), March, 2001.

[5] Sriram Ramabhadran, Joseph Pasquale, "A Framework for Application-Specific Customization of Network Services," Proceedings of Data Compression Conference (DCC'97), pp.456, March, 1997.

[6] Patrick Tullmann, Mike Hibler, Jay Lepreau, "Janos : A Java-oriented OS for Active Network Nodes," Proceedings of DARPA Active Networks Conference and Exposition(DANCE'02), pp.117-129, May, 2002.

[7] D. Wetherall, J. Guttag, D. Tennenhouse, "ANTS : A Toolkit for Building and Dynamically Deploying Network Protocols," Open Architectures and Network Programming(IEEE OPENARCH'98), pp.117-129, April, 1998.

[8] Robert Barden, Bob Lindell, Steven Berson, Ted Faber, "The ASP EE : An Active Network Execution Environment," Proceedings of DARPA Active Networks Conference and Exposition(DANCE'02), pp.238-254, May, 2002.

[9] Javed I. Khan, Seung S. Yang, "A Framework for Building Complex Netcentric Systems on Active Network," Proceedings of DARPA Active Networks Conference and Exposition(DANCE'02), pp.409-426, May, 2002.

[10] Gray T. Wong, Matti A. Hiltunen, Richard D. Schlichting, "A Configurable and Extensible Transport Protocol," Proceedings of INFOCOM 2001, Vol.1, pp.319-328, April, 2001.

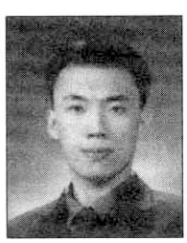
[11] G. Minden, E. Komp, S. Ganje, M. Kannan, S. Subramaniam, S. Tan, S. Vallabhaneni, J. Evans, "Composite Protocols for Innovative Active Services," Proceedings of DARPA Active Networks Conference and Exposition(DANCE'02), pp.157-164, May, 2002.

[12] S. Merugu, S. Bhattacharjee, Y. Chae, M. Sanders, K. Calvert, E. Zegura, "Bowman and CANEs : Implementation of an Active Network," Proceeding of the 37th Annual Allerton Conference on Communication, Control and Computing, Monticello, Illinois, September, 1999.

[13] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, Jonathan M. Smith, "A secure active network architecture : Realization in switchware," IEEE Network, Vol.12, No.3, pp.37-45, May/June, 1998,

[14] Martin Fowler, Kendall Scott, UML Distilled : A Brief Guide to the Standard Object Modeling Language, Addition-Wesley Publisher, ISBN 020165783X, 1999.

[15] Erich Gamma, John Vlissides, Ralph Johnson, Richard Helm, John M. Vlissides, Elements of Reusable Object Oriented Software, Addison-Wesley Publisher, ISBN 0201633612, 1994.



이 화 영

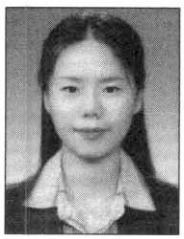
e-mail : hylee04@lge.com

2002년 경북대학교 컴퓨터과학과(이학사)

2004년 경북대학교 대학원 컴퓨터과학과 (이학석사)

2004년~현재 LG전자 홈넷사업팀 시스템 개발그룹 연구원

관심분야 : 컴퓨터 통신, 이동 컴퓨팅, 무선인터넷, 액티브 네트워크, 홈 네트워킹 등



강 보 영

e-mail : bykang@ccmc.knu.ac.kr

2003년 경북대학교 컴퓨터과학과 이학사

2003년~현재 경북대학교 대학원 컴퓨터과학과 석사과정

관심분야 : 컴퓨터 통신, 이동 컴퓨팅, 무선인터넷, 액티브 네트워크, 홈 네트워킹 등

임 경 식

e-mail : kslim@knu.ac.kr

1982년 경북대학교 전자공학과 공학사

1985년 한국과학기술원 대학원 전산학과 공학석사

1994년 University of Florida 전산학과 공학박사

1985년~1998년 한국전자통신연구원 책임연구원, 실장

1998년~2004년 경북대학교 컴퓨터과학과 조교수

1998년~현재 경북대학교 컴퓨터과학과 부교수

관심분야 : 이동 컴퓨팅, 무선인터넷, 홈 네트워킹, 컴퓨터통신 등