유비쿼터스 네트웍 환경에서 계층적 이벤트 서비스를 위한 다중 이벤트 서버 아키텍처

학생회원 신 준 헌*, 박 준 호**, 종신회원 강 순 주***, 최 준 용****

Multi Event Server Architecture for Hierarchical Event Service under Ubiquitous Network Environment

Jun-Heon Shin*, Jun-Ho Park** *Student Members*,
Soon-Ju Kang*** Jun-Yong Choi**** *A Lifelong Members*

요 약

유비쿼터스 네트웍은 다양한 프로토콜을 사용하는 많은 디바이스로 구성되며 상호간의 유연한 연동을 필요로 한다. 본 논문에서는 유비쿼터스 네트웍 환경에서 디바이스 상호간의 신뢰성 있는 통신을 지원하기 위한 계층적 이벤트 서비스와 이를 위한 다중 이벤트를 제안한다. 본 논문에서 제안된 이벤트 서비스는 다양한 통신 방법을 지원하기 위해 물리적인 위치 및 논리적인 위치에 따라 디바이스를 분류하였으며 분산 네트웍 환경에서 나타나는 이벤트 전송의 중복을 제거하여 네트웍 부하를 줄이며 이벤트 전송 과정에서 발생 가능한 오류에 대하여 신뢰성 있는 처리를 보장하도록 설계 구현되었다. 제안된 이벤트 서비스는 홈 네트웍 사례 연구를 통하여 직접 구현되었으며 성능을 평가하였다.

Key Words : Event Service, Home Network, Ubiquitous Network, Middleware, CORBA

ABSTRACT

Ubiquitous network consists of various devices using several heterogeneous protocols and requires seamless communication between devices. This paper proposes a new hierarchical event service and multiple event server architecture to implement the service under ubiquitous environment. The proposed event service classifies devices by logical and physical location to abstract the idiosyncrasy of used protocols, and reduces network load. Due to the design consideration, the proposed architecture removes transfer of duplicated event data effectively. As a result, we can guarantee the reliable event delivery. The prototype multiple event server architecture was implemented according to the proposed idea and evaluated its performance under a home network test bed.

Ⅰ. 서 론

유비쿼터스 네트웍[1]을 구성하는 각각의 디바이스들은 목적이나 제공하고자 하는 서비스에 따라 서로 다른 특징을 가질 뿐만 아니라 사용하는 네트웍 또한 다양하다. 디바이스가 제공하는 서비스에 따라 하부 네트웍 망은 크게 데이터 네트웍, 제어 네트웍, 멀티미디어 네트웍, 무선 네트웍 등으로 나눌 수 있다. 신뢰성 있는 데이터의 전송을 위해서는 TCP/IP와 같은 데이터 네트웍을 사용하며, 제어 정

* 경북대학교 전자전기컴퓨터학부 실시간 시스템 연구실 (setiang@hotmail.com),
** 경북대학교 전자전기컴퓨터학부 (zec@palgong.knu.ac.kr)
*** 경북대학교 전자전기컴퓨터학부 (sjkang@ee.knu.ac.kr), **** 경북대학교 전자전기컴퓨터학부 (jychoi@ee.knu.ac.kr)
논문번호 : 040125-0317, 접수일자 : 2004년 3월 1일
※본 연구는 한국과학재단 특정기초연구(R01-2003-000-10252-0)지원으로 수행되었음.

보의 실시간 전송을 위해서는 LonWorks[2] 등의 제어 네트워이 사용되고 있다. 또한, 여러 미디어 정보를 동시에 전달해야 하는 멀티미디어 정보전송 시 광대역 폭을 지닌 IEEE1394[3]와 같은 멀티미디어 네트워이 사용되고 있으며 이외에도 Bluetooth[4], wireless LAN[5]과 같은 무선 기반의 네트워이 사용되고 있다. 유비쿼터스 네트워의 디바이스들은 필요로 하는 전송률이나 제어 특성이 각기 다르기 때문에 모든 디바이스들을 하나의 네트워 프로토콜로 묶는다는 것은 각 디바이스들의 특성들을 활용할 수 없게 하여 현실적으로 불가능하다. 따라서 유비쿼터스 네트워은 특정 디바이스의 제어 및 관리의 특징을 상실하지 않으면서 다수의 프로토콜들 간에 유연한 연동이 가능한 형태를 가져야 하며, 이러한 기능을 제공하기 위한 미들웨어 설계에 있어서는 복합적인 하위 프로토콜 환경을 통합하고 상위의 응용 어플리케이션 계층에 단일의 인터페이스를 제공하는 기능이 필수적이다.

또한 유비쿼터스 네트워 환경에서 각기 다른 하부 프로토콜을 사용하는 디바이스들 사이의 연동은 필수적으로 일어난다. 예를 들어 데이터 네트워에서 동작하는 에이전트가 제어 네트워에서 동작하는 디바이스를 제어하거나 모니터링 하는 경우, 두 네트워 사이의 유연한 상호 통신이 필요하게 된다. 따라서 본 논문에서는 CORBA(Common Object Request Broker Architecture)[5]를 이용하여 하위 프로토콜을 추상화하고 상위 어플리케이션 사이의 상호 연동을 지원할 수 있는 이벤트 서비스 구조를 제안하고자 한다. CORBA는 하위 계층에 새로운 네트워 프로토콜을 쉽게 추가할 수 있는 확장성 있는 구조를 가지고 있으므로 다양한 하위 네트워의 추상화를 제공하여 분산된 객체들 간의 상호 연동을 제공하기에 적합하다. 또한 플랫폼과 언어에 독립적이고 다양한 서비스를 제공할 수 있어 이기종 간의 서버/클라이언트 구조 개발이 용이하다. 또한 CORBA는 분산 객체들 간의 비동기 통신[7]을 지원하기 위해서 이벤트 서비스(Event Service)[8]를 정의하고 있어 상위 객체들 간의 원활한 비동기 통신을 제공한다. 하지만 표준으로 제안된 이벤트 서비스는 보편적인 형태의 인터페이스를 가지기 때문에 유비쿼터스 네트워 환경에는 적합하지 않다. CORBA 이벤트 서비스는 단일 이벤트 서버를 정의하며 하나의 이벤트 서버를 통하여 모든 분산 객체가 이벤트 데이터를 송수신 하는 구조를 가진다. 따라서 디바이스 개수가 증가함에 따라 서버의 부하도 증가할 것이며, 단일 이벤트 서버의 오류는 전체 시스템의 오류를 유발할 수 있다. 또한 CORBA 이벤트 서버는 이벤트 필터링 기능을 정의하고 있지 않아 분산 객체들은 모든 이벤트를 받아야 하는 문제점을 가지고 있으며, 전송 과정에서 발생한 오류를 해결하는 방법을 제시하지 않고 있다. 따라서 본 논문에서는 CORBA를 유비쿼터스 환경의 기반 미들웨어로 사용하고 상위에서 디바이스들 간의 상호 메시지 전송을 위하여 확장된 계층적 이벤트 서비스 구조를 제안하고 이를 위한 다중 이벤트 서버 구조도 제안한다.

본 논문은 다음과 같이 구성된다. 2장에서 요구사항을 살펴보고, 3장에서는 제안된 이벤트 시스템의 전체적인 구조 및 세부 설계에 대해 설명한다. 4장에서는 제안된 이벤트 서비스를 이용한 사례 연구를 살펴보고, 5장에서는 제안된 이벤트 서비스의 성능을 평가한다. 6장에서 관련 연구를 살펴보고, 마지막으로 7장에서 결론을 맺도록 한다.

Ⅱ. 요구사항 분석

유비쿼터스 네트워은 다양한 종류의 디바이스들로 구성된다. 대표적인 유비쿼터스 네트워인 홈 네트워의 경우 전등과 같은 단순 가전기기와 인터넷에 연결된 냉장고, 에어컨과 같은 정보 가전기기들, 고속의 A/V 데이터 전송을 요구하는 디지털 캠코더 및 디지털 TV와 같은 멀티미디어 기기들이 존재한다. 이러한 디바이스들은 서로 다른 특징들을 가지고 있다. A/V 가전기기의 경우 데이터 전송을 위한 높은 대역폭을 필요로 하며 전구와 같은 단순 가전기기는 신뢰성 있는 제어 명령의 전송을 필요로 한다. 유비쿼터스 네트워과 같은 유동적인 환경에서 각각의 디바이스들은 상호 연동하기 위해 다른 종류의 디바이스들과 비동기 통신을 필요로 한다. 디바이스들 간의 비동기 통신을 위한 방법으로 분산 콜백(Distributed Callback)[9]이 있다. 분산 콜백은 단대단(peer-to-peer) 통신으로 자신이 전송의 주체가 되어 데이터를 특정 디바이스로 전달할 수 있다. 분산 콜백을 이용하여 상호 통신을 하기 위해서는 통신하고자 하는 다른 객체의 정보를 모두 알고 있어야 한다. 유비쿼터스 네트워 환경에서는 많은 디바이스가 존재하며, 네트워에 존재하는 다양한 디바이스 정보를 저장하기 위해서는 많은 리소스를 필요로 한다. 또한 디바이스의 유동적인 추가와 제거에 따른 디바이스 정보의 갱신 및 유지, 관리 또한

개별적인 디바이스에게 맡겨진다. 따라서 유비쿼터스 네트워 환경에서 분산 콜백을 이용한 통신은 전체 시스템의 확장성(scalability)을 제한하게 된다. 또 다른 문제점으로는 통신 과정에 발생한 오류는 전송의 주체가 되는 송신측 객체가 모두 처리해 주어야 하며 수신 객체의 수에 비례하여 송신 객체는 통신을 위해 많은 시간이 필요로 하게 된다. 그리고 전송 과정에서 오류 발생으로 송신 객체가 블록(block)이 되어 버린다면 데이터 전송의 신뢰성을 보장하지 못한다. 또한 통신의 대상이 되는 송신 및 수신 객체의 의존적인 관계(tightly coupled)로 인하여 개발 및 유지 보수에 어려움이 있다. 따라서 두 객체간의 비동기 통신 및 약한 의존 관계(loosely coupled connection)를 제공하기 위해 이벤트 서비스를 이용하는 것이 유비쿼터스 환경에 적합하다.

CORBA 명세서에서는 객체 사이의 비동기 통신을 위한 이벤트 서비스를 정의 하고 있다. 이벤트 서비스는 분산 콜백의 문제점으로 지적한 많은 양의 리소스와 확장성의 제한 그리고 통신 과정에서의 오류 처리와 디바이스 간의 밀접한 의존관계 등을 해소할 수 있다. 그러나 표준으로 제안된 CORBA 이벤트 서비스 역시 유비쿼터스 네트워 환경에 적용하기에는 몇 가지 문제점을 가지고 있다.

1. CORBA 이벤트 서비스의 문제점

(1) 단일 이벤트 서버

CORBA 명세서에서는 단일 이벤트 서버를 중심으로 다수의 이벤트 공급자 및 소비자를 고려하여 인터페이스를 정의하고 있어 이벤트 서버의 분산은 고려하지 않고 있다. CORBA에서 정의된 이벤트 전송 방식은 단일 이벤트 서버를 중심으로 모든 디바이스가 통신하는 중앙 집중식(centralized) 구조가 된다. 이 경우 디바이스의 리소스 관리 측면에서 용이하지만 디바이스 개수가 증가함에 따라 이벤트 서버에 네트워 트래픽이 집중될 것이며 단일 이벤트 서버 오류로 인하여 전체 서비스에 영향을 줄 것이다. 유비쿼터스 환경에서는 중앙 집중식 이벤트 서버로 네트워을 관리하는 것은 부적절하며 분산 이벤트 서버를 두어 관리 하는 것이 전체 시스템의 안정성을 보장해 줄 것이다. 따라서 하나의 이벤트 서버가 아닌 위치에 따른 분산 이벤트 서버가 필요하며 분산된 이벤트 서버들 사이의 통신을 위한 인터페이스 정의가 필요하다. 이벤트 서버들 사이의 통신은 일반적인 공급자 및 소비자 관계와 동일한 성격을 가진다. 이는 하나의 이벤트 서버 입장에서 다른 이벤트 서버는 이벤트 공급자로 자신은 이벤트 소비자로서 역할을 하기 때문이다.

(2) 중복 전송

유비쿼터스 네트워 환경에서 각 디바이스 및 어플리케이션은 역할에 따라 이벤트의 공급자, 소비자 또는 두 가지 역할을 동시에 수행할 수 있다. 예를 들어 홈 네트워 환경에서 온도 센서와 같은 디바이스는 공급자로써 동작하며 모니터링 디바이스는 소비자 역할을 할 것이다. 전구와 같은 디바이스는 상태정보를 송신하고 제어정보를 수신하는 공급자와 소비자 두 가지의 역할을 하게 된다. 이러한 경우 CORBA 명세서에 정의된 이벤트 서비스 구조에서 디바이스가 공급자이면서 소비자로서 동작할 때 자신이 생성한 데이터를 자신이 수신하는 경우가 발생한다. 또한 이벤트 서버 사이의 공급자 및 소비자 관계에서 이벤트의 무한 반복 전송이 일어날 수 있다. 따라서 이벤트 서버 사이의 이벤트 반복 전송은 하부 디바이스에게 영향을 미치며 네트워을 불안정하게 만든다.

(3) 불필요한 이벤트 전송과 그룹 이벤트 전송의 고려 미흡

CORBA 이벤트 서버 명세서는 이벤트 필터링 기능을 정의하고 있지 않다. 따라서 채널에 연결된 모든 소비자는 채널에 연결된 모든 공급자가 생성하는 이벤트 데이터를 수신하게 되며 네트워에 연결된 디바이스는 불필요한 데이터를 수신하게 된다. 또한 불필요한 이벤트의 전송으로 인해 전송량이 증가하여 전송 지연이 발생하게 된다. 따라서 소비자 입장에서는 자신이 필요로 하는 이벤트 데이터만 수신하는 구조가 필요하다. 이와 반대로 공급자 입장에서는 소비자를 지정하여 이벤트 데이터를 전송할 수 있는 구조가 필요하다. 유비쿼터스 환경에서 이벤트는 특정한 디바이스 그룹에 전송될 필요가 있다. 홈 네트워 환경을 예를 들면 "거실에서 안방의 모든 전등을 꺼라" 또는 외출시 "집안의 모든 가전기기의 전원을 꺼라" 등의 서비스는 공급자가 특정 소비자뿐만 아니라 다수의 소비자를 선택하여 전송하는 구조가 제공되어야 할 것이다.

(4) 이벤트 서버의 신뢰성 보장 미흡

이벤트 서비스를 수행하는 도중 이벤트 데이터 전달 과정에서 오류가 발생할 수 있으며 이벤트 서비스 자체가 블록 될 수 있다. 이러한 오류 상황은 유비쿼터스 네트워 전체의 이벤트 전달이 불가능하게 한다. 기본적으로 CORBA는 원격 객체간의 통신 방법이 동기통신(Synchronous communication

with reply)을 사용하기 때문에 발생할 수 있다. 따라서 이벤트 서버는 오류가 발생하더라도 하나의 이벤트 연결만 블록이 되며 나머지 이벤트는 지속적으로 전달할 수 있는 신뢰성 있는 구조를 가져야 할 것이다.

Ⅲ. 제안한 시스템의 세부설계 및 구현

1. 이벤트 서버

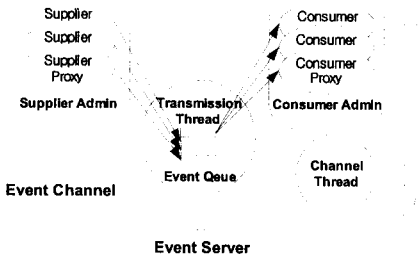


그림 1. 제안된 이벤트 서버 내부 구조

그림 1은 본 논문에서 제안하는 이벤트 서버의 내부 구조를 보여준다. 서버 내부에는 복수개의 채널이 존재할 수 있으며 각 채널에는 전송 쓰레드(Transmission Thread)와 채널 쓰레드(Channel Thread)가 동작한다. 그리고 프락시 객체 생성을 담당하는 공급자/소비자 관리자(Supplier/Consumer Admin) 객체로 구성된다. 이벤트 서버에 공급자와

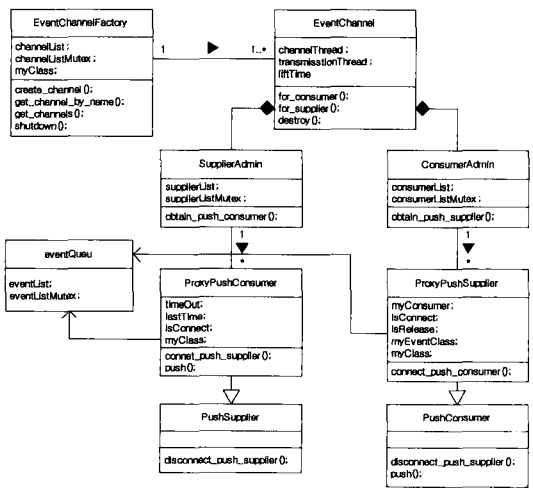


그림 2. 이벤트 서버 클래스 다이어그램

소비자가 연결되면 그 개수만큼 프락시 객체가 생성된다. 서버 내부의 컴포넌트들을 클래스 단위로 주요 기능 및 저장소 등의 관계를 나타내면 그림 2와 같다.

(1) 이벤트 채널 팩토리

이벤트 채널 팩토리 클래스는 기본적으로 이벤트 채널을 관리하기 위한 클래스이며 내부에 이벤트 채널 정보를 저장하기 위한 리스트와 리스트 접근을 위한 뮤텍스(mutex)를 가진다. 그리고 요청에 따라 이벤트 채널을 생성하거나, 생성된 이벤트 채널을 관리하는 오퍼레이션을 가진다.

(2) 이벤트 채널

이벤트 채널 클래스는 공급자와 소비자에게 역할에 따른 관리자 인터페이스(Admin Interface)를 제공한다. 이벤트 서버 내부에는 복수개의 채널 생성이 가능하며, 각각의 채널에는 두개의 쓰레드가 수행된다. 전송 쓰레드는 이벤트 큐를 확인하여 공급자 프락시 객체가 전송한 이벤트 데이터를 소비자 프락시 객체에 전달하는 기능을 수행한다. 채널 쓰레드는 소비자 프락시(ProxyPushConsumer) 클래스 내부의 timeout, lastTime, isConnect와 공급자 프락시(ProxyPushSupplier) 클래스 내부의 isRelease, isConnect를 확인하여 유효하지 않는 프락시 객체를 제거하는 기능을 수행한다. 그리고 이벤트 채널은 채널 내부에 프락시 객체가 존재 하지 않을 때 즉, 공급자와 소비자 존재하지 않을 때 채널 소멸을 결정하는 시간(liftTime)을 가진다.

(3) 공급자 관리자(Supplier Admin)

공급자 관리자 클래스는 이벤트 서버 내부에서 공급자와 통신을 담당하는 프락시 소비자 객체를 생성하는 기능을 제공한다. 공급자의 수에 따라 프락시 객체가 생성되며 이들을 관리하기 위하여 리스트 형태의 자료구조를 사용한다. 이벤트 채널 팩토리 클래스와 동일하게 리스트 접근을 위해서 뮤텍스를 사용한다.

(4) 소비자 관리자(Consumer Admin)

소비자 관리자 클래스는 소비자를 위한 클래스이며 공급자 관리자 클래스와 동일한 역할을 한다. 프락시 공급자 객체를 생성하는 기능을 제공하며 생성된 프락시 공급자 객체 관리를 위하여 리스트와 뮤텍스를 사용한다.

(5) 프락시 소비자

프락시 소비자 클래스는 공급자와 직접 통신을 담당하는 클래스로 푸쉬 공급자(PushSupplier) 클래스를 상속한다. 공급자의 연결 상태와 아이디를 저

장하며 프락시 객체 유효성을 검사하기 위한 lastTime, timeOut을 저장한다.

(6) 프락시 공급자

프락시 공급자 클래스는 소비자와 직접 통신을 담당하는 클래스로 푸쉬 소비자(PushConsumer) 클래스를 상속한다. 내부에 소비자의 연결 상태, 아이디, 그리고 전송 과정에서 발생한 예외를 저장하는 isRelease를 가진다. 이 외에도 소비자에게 이벤트 데이터를 전송하기 위한 소비자 객체 정보를 저장하며 소비자가 수신을 원하는 이벤트 데이터의 클래스(myEventClass)도 저장한다.

(7) 푸쉬 공급자 및 푸쉬 소비자

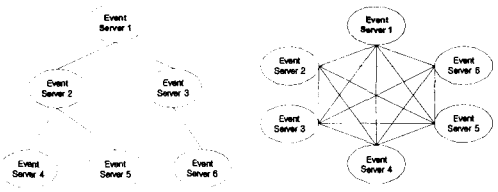
이벤트 채널을 이용하지 않고 공급자와 소비자가 직접 통신을 할 때 사용하는 클래스로 이벤트 데이터 전송과 서로간의 연결을 끊을 수 있는 오퍼레이션으로 이루어져 있다.

(8) 이벤트 큐

프락시 소비자가 공급자로부터 이벤트 데이터를 수신했을 때 이벤트 데이터를 이벤트 큐에 저장한다. 채널 내부의 전송 쓰레드는 이벤트 큐를 확인하여 프락시 공급자 클래스 내부의 소비자 객체 정보를 통하여 소비자에게 이벤트 데이터를 전달하게 된다. 이벤트 큐는 이벤트 서버 내에서 프락시 소비자/공급자 사이에 이벤트 전송을 위하여 데이터를 저장하는 공간이다. 이벤트 큐는 이벤트 데이터를 리스트 형태로 관리하며 접근을 위하여 뮤텍스를 사용한다.

2. 계층적 다중 이벤트 서버

본 논문에서는 유비쿼터스 환경을 위한 분산 다중 이벤트 서버 아키텍처를 제안한다. 이를 위해서 그림 3과 같이 이벤트 서버들 사이의 연결을 위한 인터페이스가 필요하다. 분산 이벤트 서버 환경에서 이벤트 서버들의 관계는 일반적인 공급자 및 소비자와 동일한 형태를 가진다.



(a)계층적 연결 (b)완전한 연결

그림 3. 다중 이벤트 서버

그림 3의 (a), (b)는 분산 다중 이벤트 서버를 구성하기 위한 두 가지 방법을 보여준다. 그림 3의 (a)는 계층적으로 구현된 이벤트 서버 연결 구조를 나타내며, (b)는 상호 연결된 서버 구조를 보여준다. 그림 3의 (a)의 경우 이벤트 서버 1이 공급자로 동작하고 서버 4,5가 소비자로 동작할 경우 이벤트 서버 2를 경유하여 전달된다. 이러한 경유 각 서버는 계층적 구조에서 상위 서버와 하위 서버의 연결 정보만 알고 있으면 되지만 하나의 서버의 오류는 하위 서버에 이벤트 전송을 할 수 없게 된다. 반면 그림 3의 (b)의 경우 각 이벤트 서버는 서버 개수만큼 연결 정보를 저장해야 한다. 그러나 특정 이벤트 서버의 오류로 다른 이벤트 서버들 사이의 전송에는 영향을 주지 않으며 이벤트 데이터의 전송 경로를 최소한으로 줄일 수 있는 장점이 있다. 따라서 본 논문에서는 그림 3의 (b)와 같이 서버들 사이에 완전히 연결된 서버 구조를 적용하였으며 각 이벤트 서버 사이의 연결을 위하여 그림 4와 같이 이벤트 서버 상위에 설정 서버(Configuration Server)를 정의 하였다.

설정 서버는 새로운 이벤트 서버가 연결 될 때 기존에 수행되고 있는 이벤트 서버에 새로운 이벤트 서버가 추가되었음을 알려준다. 또한 새로이 추가된 이벤트 서버에는 기존에 수행되는 이벤트 서버의 정보를 알려 줌으로서 상호간의 연결을 가능하게 해 준다. 이러한 연결 과정은 다음 그림 5와 같다.

이벤트 서버 1은 설정 서버에 자신의 정보를 등록한다. 이 등록 정보는 서버 2가 등록 되었을 때 서버 2에게 전달되어 서버 2가 서버 1에 소비자로 등록될 때 사용된다. 또한 서버 2가 등록되었을 때 설정 서버는 서버 1에게 전달하며, 서버 2의 정보를 이용하여 서버 1이 서버 2에 소비자로 등록한다. 이러한 방법으로 그림 3의 (b)와 같이 완전한 연결이 이루어진다.

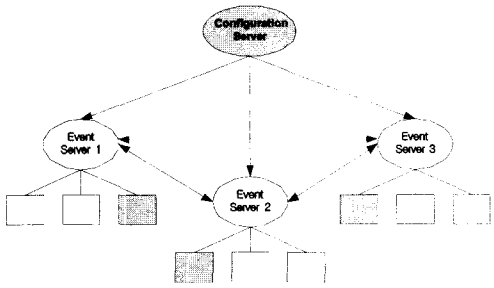


그림 4. 설정 서버와 이벤트 서버의 구성

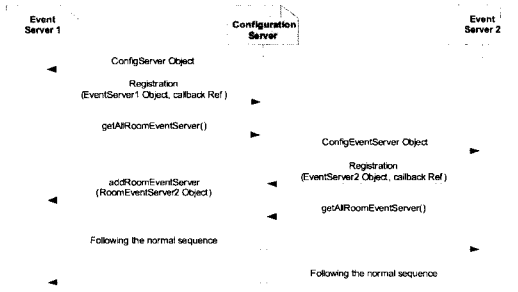


그림 5. 설정 서버와 이벤트 서버의 연결 과정

3. 디바이스 및 이벤트 데이터의 분류

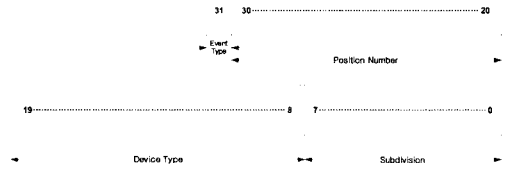


그림 6. 디바이스 및 이벤트 데이터 분류

본 논문에서는 디바이스에 유일한 클래스를 부여하여 이벤트 전송 시 공급자와 소비자를 명확히 구분하였다. 디바이스 클래스 구성은 그림 6과 같이 32비트 필드를 사용한다. 상위 20에서 30비트는 디바이스의 물리적인 위치 정보를 나타낸다. 11비트의 길이를 가지며 각각의 비트에 의미를 두어 총 11개의 위치로 구분이 가능하다. 계층적 홈 네트워 환경에서는 룸 단위로, 첫 번째 룸은 0000000001(2)로, 두 번째 룸은 0000000010(2)로 표현가능하다. 8에서 19비트는 디바이스의 타입을 구분 짓는 필드로 디바이스의 논리적인 성격을 나타낸다. 홈 네트워 환경에서 램프, A/V, 난방, 냉방, … 등으로 디바이스의 성격을 구분할 수 있다. 0에서 7비트는 동일한 성격(타입)을 가지는 여러 디바이스들을 세분화하기 위한 필드로서 동일한 룸에 램프종류의 디바이스가 3개 있다면 디바이스 클래스는 00000001(2), 00000010(2) 그리고 00000100(2)으로 사용된다. 표 1은 홈 네트워 환경에서 디바이스 구분을 위한 각각의 필드의 의미와 사용 예이다. 이렇게 분류된 디바이스가 발생하는 이벤트 데이터를 성격에 따라 제어 정보와 상태 정보 이벤트로 구분할 수 있으며 최상위 비트에 저장된다.

표 1에 정의된 이벤트는 미리 정해지는 것은 아니라 홈 네트워에 사용될 경우의 이벤트 형태에 대한 정의이다. 표 1에서 정의된 이벤트 타입은 소비자나 공급자가 프락시 객체를 생성할 때 이벤트 서버에 등록할 수 있는 이벤트 클래스의 예를 보여준다. 공급자가 프락시 객체에 이벤트를 전달할 때 등록된 이벤트 객체를 비교하여 소비자에게 전송할지를 결정한다. 즉 이벤트 공급자와 소비자는 유비쿼터스 네트워에서 사용할 임의의 포 1과 같은 이벤트 타입을 정의할 수 있다.

표 1. 디바이스의 분류

Name	Bit Range	Value	Meaning
Event Type	31 (1 bit)	0	Device status event
		1	Control event
Room Number	20~30 (11 bit)	00000000001 (2)	Bed room
		00000000010 (2)	Living room
		...	..
Room Number	8~19 (12 bit)	0x001	Light
		0x002	A/V
		0x003	Controller
			...
		0xC00	Reserved for Event Server
Subdivision	0~7 (8 bit)	0x01	Defend on Device Type
		0x02	
		...	

4. 불필요한 전송 제거

앞서 3.3절에서 설명한 바와 같이 이벤트 형태를 정의하여 각 디바이스마다 유일한 클래스를 구분하였다. 이는 이벤트 데이터 전송 시 공급자와 소비자를 명확히 정의하여 불필요한 전송을 줄일 수 있다. 또한 이벤트의 성격에 따라 상태정보 이벤트와 제어정보 이벤트로 구분하였다. 상태정보의 경우 이벤트 데이터를 발생시킨 공급자를 구분할 수 있기 때문에 관심 있는 소비자들에게 전송할 수 있다. 이와 반대로 제어정보의 경우 이벤트 데이터를 발생시킨 공급자보다 수신하는 소비자를 명확히 구분하여 이벤트 데이터를 전송한다. 이러한 전송 방법을 제공하기 위하여 소비자는 수신을 원하는 디바이스의 종류를 등록할 수 있어야 하며 공급자는 이벤트 데이터를 발생할 때 이벤트 데이터의 종류(상태 및 제어정보)에 따라 자신의 디바이스 클래스와 수신 소비자를 지정할 수 있어야 한다. 다음 코드는 공급자 및 소비자가 이벤트 서버에 등록할 때 사용하는 인

터페이스를 나타낸다.

```
interface ConsumerAdmin
{
    ProxyPushSupplier obtain_push_supplier(
                        in unsigned long eventclass,
                        in unsigned long myeventclass);
}
interface SupplierAdmin
{
    ProxyPushConsumer obtain_push_consumer(
                        in unsigned long timeout,
                        in unsigned long myeventclass);
}
```

서버 내부에 프락시 객체를 생성할 때 소비자는 32비트 두 개의 매개변수를 사용한다. 첫 번째 매개변수인 eventclass는 자신이 수신하고자 하는 디바이스의 클래스이다. 두 번째 매개 변수는 자신의 디바이스 클래스로 네트워 상에서 디바이스가 공급자이면서 소비자로써 동작할 때 자신 발생한 이벤트 데이터를 자신이 수신하지 않도록 하기 위하여 필요하다. 이러한 중복 전송 방지는 이벤트 서버들 사이에서도 동일하게 적용된다. 이벤트 서버도 유일한 클래스를 가지며 일반적인 디바이스와 동일하게 공급자/소비자 관계를 가지도록 설계되었다.

두 개의 매개변수는 이벤트 서버 내부 전송 쓰레드에서 프락시 객체 사이의 전송 시 사용된다. 제어 이벤트의 경우에는 이벤트 데이터와 함께 전송되는 클래스와 각각의 소비자 클래스를 비교하여 전송하며, 이와 다르게 상태정보 이벤트의 경우에는 소비자가 수신을 원하는 클래스와 비교하여 전송한다.

5. 그룹 이벤트 전송

유비쿼터스 환경에서는 개개의 디바이스 제어뿐만 아니라 논리적인 단위로 다수의 디바이스를 제어할 수 있는 방법이 필요하다. 홈 네트웍에서 예를 들어보면 '거실의 모든 디바이스를 끈다(power off).' 또는 '홈 내의 모든 전등을 끈다.' 식의 서비스가 필요하다. 본 논문에서 제안하는 이벤트 서비스의 경우 모든 디바이스는 32비트의 유일한 클래스를 가지고 있으며 제어 이벤트의 경우 수신 디바이스를 지정할 수 있다.

표 1의 분류를 바탕으로 그림 7의 첫 번째 예는 하나의 이벤트 서버 하부의 모든 디바이스를 제어하는 그림으로 공급자는 이벤트 데이터와 함께 0xC02FFFFF의 이벤트 클래스를 사용할 수 있다. 0xC02은 제어 이벤트이며 거실로 전송을 뜻하며 FFFFF는 모든 타입의 디바이스를 뜻한다. 동일한 방법으로 그림 7의 두 번째 예에서는 이벤트 클래스를 0xFFF001FF값으로 설정할 수 있다. 0xFFF는 제어 이벤트 이며 모든 룸을 뜻하고 001FF는 전구 종류의 모든 디바이스를 의미한다. 자신이 관심 있는 이벤트 클래스를 정의하여 전송함으로써 물리적 위치에 따른 디바이스 그룹, 또는 논리적인 디바이스의 그룹 제어가 가능하도록 하였다.

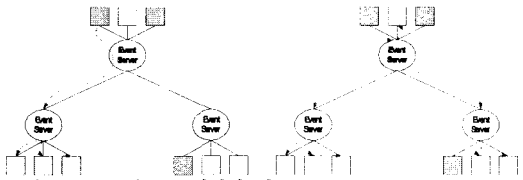


그림 7. 그룹 이벤트 데이터 전송

6. 이벤트 서버의 신뢰성 보장

```
interface PushConsumer
{
    void oneway push(in any data,
                     in unsigned long eventclass)
    raises(Disconnected);
}
```

다음 코드는 이벤트 데이터 전송 과정에서 발생할 수 있는 오류에 대하여 서버의 신뢰성을 보장하기 위해 전송을 위한 인터페이스를 정의하고 있다. 이벤트 전송 시 특정 이벤트 소비자의 블록 현상을 막기 위해 아래 코드와 같은 푸쉬 함수를 정의한다.

그림 1에서 전송 쓰레드는 채널 내부에 하나만 존재하며 여러 개의 소비자로 이벤트 데이터를 전송하게 된다. 이 경우 첫 번째 소비자로 이벤트 데이터 전송할 때 오류가 발생하면 전체 쓰레드가 블록되어 채널 내부에 존재하는 모든 소비자로 이벤트 데이터 전송이 불가능해 진다. 따라서 푸쉬함수를 단방향(oneway) 함수로 정의하여 블록 안되도록 하였다. 함수의 매개변수는 CORBA의 any타입의 이벤트 데이터와 32비트 evnetclass를 사용한다. eventclass는 3.4절에서 설명한 것과 같이 상태정보의 경우 이벤트 데이터를 발생시킨 디바이스의 아이디를 나타내고 제어 이벤트일 경우는 목적지 디바이스의 클래스를 나타낸다.

oneway 키워드는 CORBA의 원격호출 방식을 비동기통신(asynchronous communication without reply)으로 가능하게 해 준다. oneway 키워드를 사용함으로써 공급자가 이벤트 서버로 또는 이벤트

서버가 소비자로 이벤트 데이터를 전송 할 경우 이벤트 서버의 응답, 소비자의 응답을 기다리지 않고 자신의 일을 수행하도록 한다. 따라서 공급자는 이벤트 서버의 응답이 없어도 블록 되지 않으며 이벤트 서버 역시 소비자의 응답이 없어도 블록 되지 않도록 하였다.

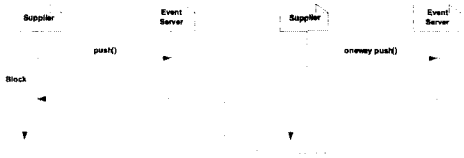


그림 8. 일반적인 푸쉬 함수와 oneway 푸쉬 함수

이외의 오류 상황(원격 객체의 오류)은 CORBA의 예외(exception)처리를 통하여 해결할 수 있다. 이벤트 서버 내부에는 공급자와 소비자의 수만큼 프락시 객체가 생성, 존재 하게 된다. 공급자의 경우 lifetime을 지정하여 시간 내에 이벤트 데이터 전송이 발생하지 않으면 서버내부에서 프락시 객체를 제거하였다. 그러나 본 논문에서 제안하는 이벤트 서비스의 경우 푸쉬 모델을 사용하기 때문에 소비자 프락시 객체의 유효성을 검사할 수 있는 방법은 없다. 따라서 이벤트 서버 내부에서 소비자에게 전송할 때 발생하는 예외 상황을 검사하여 유효하지 않는 프락시 객체를 제거하도록 하였다. oneway 푸쉬 함수의 사용과 예외 처리를 통하여 네트워을 구성하는 디바이스의 오류에 서버가 받지 않도록 하여 이벤트 서버의 신뢰성을 높였다.

Ⅳ. 사례 연구: 분산형 홈 네트워

본 논문에서 제안된 다중 이벤트 서버는 그림 9와 같이 분산형 홈 네트워을 실현하기 위해 고안되었다. 분산형 홈 네트워은 그림 9에서 보는바와 같이 홈 내부에 하나의 홈 서버가 존재하고 각 룸마다 룸 브릿지[10]라는 룸 서버가 존재한다. 홈 서버는 외부망에 대한 게이트웨이 역할과 룸 서버들에 대한 위치 정보만을 관리하며 실제적인 모든 가전제품에 대한 관리는 각 방에 설치된 룸 브릿지가 담당한다. 롬 브릿지는 하부 디바이스를 정보를 관리하며 다른 룸 브릿지 하부의 디바이스 정보도 관리한다. 이때 사용되는 프로토콜은 일반적인TCP/IP, A/V 데이터 전송을 위한 IEEE1394, 제어 네트워 프로토콜인 LonWorks가 사용된다. 연결된 디바이스들은 웹을 통한 모니터링 및 제어가 가능하다.

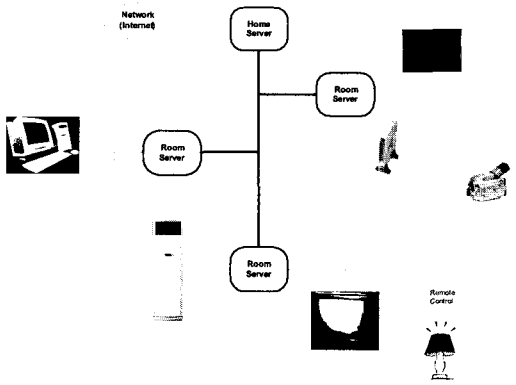
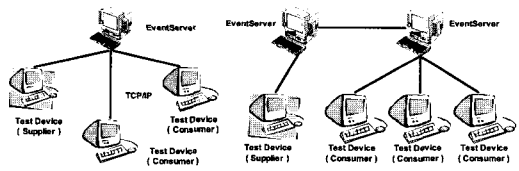


그림 9. 홈 네트워 하드웨어 구조

각 룸 서버들은 하나의 이벤트 서버를 탑재하고 있으며 다양한 프로토콜을 사용하는 디바이스들을 단일 인터페이스로 묶어 주는 역할을 한다.4장에서 설명한 바와 같이 하나의 룸 서버(이벤트 서버)에 연결된 디바이스들은 방 단위로 제어될 수 있으며 또한 홈 네트워 전체에서 논리적인 단위로 제어될 수 있다.

Ⅴ. 성능평가

프락시 객체 사이의 이벤트 전송 성능 평가를 위하여 그림 10의 (a)와 같이 네트워을 구성하였다. 그림 10의 각 이벤트 서버들은 인텔 팬티엄 프로세스 기반의 싱글 보드 컴퓨터를 이용하여 Linux 커널 2.4.18 상에서 구현되었으며 기반 미들웨어로 사용하는 CORBA는 GNOME 프로젝트로 진행되고 있는 C언어 기반의 ORBit CORBA ORB (ORBit-0.5.13)을 이용하였다. 제안된 분산 다중 이벤트 서버의 테스트를 위하여 하나의 이벤트 서버와 공급자로 구성하였으며 공급자는 10000us 단위로 이벤트를 발생시키도록 하여 1000개의 이벤트 데이터를 전송, 평균 전송시간을 측정 하였다.



(a) 서버가 하나 일 때 (b) 서버가 둘 일 때

그림 10. 성능평가를 위한 시스템 구성

분산 이벤트 서버 환경에서 디바이스가 발생한 이벤트 데이터를 다른 이벤트 서버 하부의 디바이스로 전달할 때 동일한 서버 내부에서 전달할 때보다 이벤트 서버를 하나 더 경유하게 된다. 그러나 제안 하는 구조는 완전한 연결 구조를 가지기 때문에 어떠한 경우에도 2개의 이벤트 서버만 경유하며 그 이상의 이벤트 서버를 경유하여 전송되지는 않는다. 따라서 제안하는 다중 이벤트 서버의 성능 평가는 2개의 이벤트 서버 사이에서 발생하는 전송 오버헤드를 측정함으로 가능하다. 이를 위하여 그림 10의 (b)와 같이 네트워크을 구성하였다.

위의 그림 11은 그림 10의 (a)와 같이 동일한 이벤트 서버에 소비자를 10개 연결하였을 때 각각의 소비자에게 전달되는 평균 전송시간을 보여준다. 제안된 이벤트 서버의 트랜스미션 쓰레드는 순차적으로 소비자에게 전송한 후 다음 이벤트 데이터를 역순으로 소비자에게 전송하기 때문에 전체 소비자의 평균 전송시간은 동일하다. 그러나 공급자와 소비자 프락시 객체 사이의 이벤트 데이터 전달 시간을 측정하기 위하여 순차적으로 소비자에게 이벤트 데이터를 전송하도록 하였다. 위의 결과를 보면 트랜스미션 쓰레드가 소비자와 공급자 프락시 객체 사이에서 이벤트 데이터를 전달할 때 평균적으로 200us 정도가 소요됨을 알 수 있다.

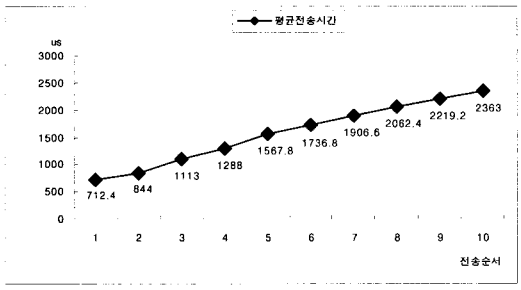


그림 11. 프락시 객체 전송 순서에 따른 평균 전송 시간

다음은 소비자의 개수를 증가시키며 평균 전송 시간을 측정한 결과로 이벤트 서버 사이에서 전송 오버헤드를 측정하기 위한 것이다.

그림 12에서 알 수 있듯이 하나의 이벤트 서버 내에서는 소비자 수가 증가에 따라 평균 전송 시간도 증가한다. 이것은 트랜스미션 쓰레드의 전송 횟수가 증가한 결과이다. 두 개의 서버를 거치는 경우도 소비자 개수가 증가함에 따라 평균 전송시간이 증가하였다. 그러나 서버 하나일 때와 비교해 보면 서버 사이의 전송 시 3ms 정도의 오버헤드가 발생함을 확인할 수 있다.

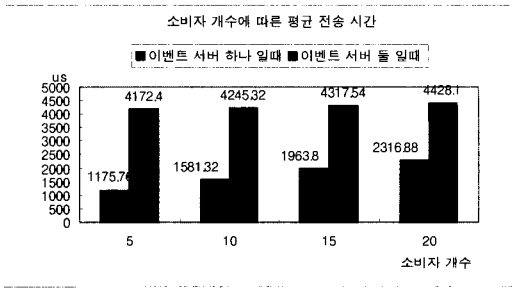


그림 12. 소비자 개수에 따른 전송 시간

Ⅵ. 관련 연구

CORBA 환경에서 이벤트 서비스를 구현한 예로써 OmniORB[13]와 ORBix[12]가 있다. 그림 13,14는 OmniORB와 ORBix의 이벤트 서비스의 내부 구조를 나타낸다.두 이벤트 서비스는 모두 CORBA의 표준 인터페이스를 따르고 있지만 내부 구현은 서로 다르다.

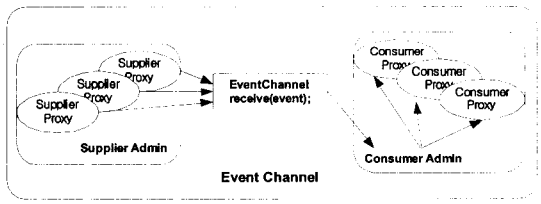


그림 13. OmniORB의 이벤트 서비스 구조

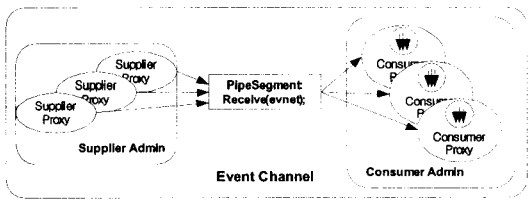


그림14. ORBix의 이벤트 서비스 구조

그림 13에서 보이는 바와 같이 프락시 객체는 공급자로부터 이벤트 채널 인터페이스의 receive 함수를 통해 이벤트를 이벤트 서버 내부로 전달하며 소비자 관리자(Consumer Admin)를 통해 이벤트를 소비자 프락시 객체에게 전달한다. ORBix의 경우 채널 내부의 receive함수를 통하여 각각의 소비자 프락시 객체에 직접적으로 전달한다. 즉 소비자 프락시 객체 내부에서 생성된 각 쓰레드가 이벤트 전송

을 담당하게 된다.

OmniORB의 경우 불필요한 내부 전송 단계를 많이 가지고 있어 특정 소비자에게 이벤트를 전송하기 위한 지연시간이 크며 하나의 쓰레드를 통해 이벤트를 전송하기 때문에 하나의 이벤트 전송 지연은 전체 이벤트 전송에 영향을 주게 된다. 반면 ORBix의 경우에는 소비자 프락시 객체 내부의 쓰레드가 이벤트 데이터를 전송함으로써 다른 소비자에게 영향을 미치지 않도록 설계되어 있다.
하지만 두 이벤트 서비스는 CORBA의 이벤트 서비스 명세를 따라 구현되어 유비쿼터스 환경에 적합하지 않다[11]. 본 논문에서는 이벤트 필터링, 불필요한 이벤트 전송 제거 등과 같은 기능을 제공하여 이벤트 서비스의 성능을 향상 시켰다. 본 논문에서 제안된 이벤트 서비스는 내부 전송 쓰레드가 프락시 객체 사이의 이벤트 전송을 담당하며 이벤트 필터링을 수행한다. 또한 oneway 푸쉬 함수를 이용하여 OmniORB와 같은 문제점은 발생하지 않는다.

Ⅶ. 결 론

본 논문에서는 유비쿼터스 환경에서 계층적 이벤트 서비스 구조를 제안하고 이를 위한 다중 이벤트 서버 구조를 설계 구현하였다. 제안된 소프트웨어 구조는 CORBA를 이용하여 하위 프로토콜을 추상화함으로써 다중 프로토콜을 지원하며, 단일의 이벤트 인터페이스를 제공하였다. CORBA 명세서의 정의된 이벤트 서비스를 확장하여 성능 향상 및 분산 환경을 지원하도록 구현하였다.

제안하는 분산 이벤트 서버 구조는 중앙 집중적 구조에서 발생할 수 있는 신뢰성 문제와 서버의 과부하를 제거하기 위하여 분산 이벤트 서버 인터페이스를 정의하고 있다. 또한 이벤트 형태를 정의하여 이벤트 공급자 및 소비자를 구분할 수 있도록 하였으며, 이벤트 자체의 구분 또한 가능하게 하였다. 이러한 분류를 통하여 중복된 이벤트 전송을 막았으며 원하는 이벤트만을 수신할 수 있도록 하였다. 그리고 특정 디바이스의 오류로 인한 전체 시스템의 오류를 제거하여 신뢰성 있는 전송구조를 가지도록 하였다. 구현된 분산 이벤트 서버는 대표적인 유비쿼트 환경인 홈 네트워에 적용하여 보았으며, 여러 테스트 방법을 통하여 제안한 시스템의 성능을 평가 하였다.

본 논문에서 제안한 분산 이벤트 서버는 이벤트 서버 사이의 통신을 제외하고는 특정한 이벤트 형태를 정의하고 있기 때문에 대표적인 유비쿼터스 환경인 홈네트워 외에도 센스 네트워과 제어 네트워에도 적용되어 사용될 수 있다.

참 고 문 헌

[1] Ken SAKAMURA, Ubiquitous Computing KAKUMEI, 2002.

[2] LonTalk Protocol Specification Version 3.0, 1994.

[3] IEEE1394 for Linux, http://www.linux1394.org.

[4] Bluetooth Resource Center, http://www.palowireless.com/bluetooth.

[5] Gerard O'Driscoll, The Essential Guide to Home Networking Technologies, Prentice Hall PTR, 2001.

[6] Object Management Group, The CommonObject Request Broker Architecture and Specification, Revision2.2: Object Mangement Group, 1995.

[7] R. Orfali, D. Harkey, and J. Edwards, The Essential Distrubuted Objects Survival Guide, John Wiley and Sons, 1996.

[8] Object Management Group, CORBAservices: Common Object Service Specification Event Service Specification: Object Management Group, 1995.

[9] HomePNA, http://www.homepna.org.

[10] Soon Ju Kang, Jun Ho Park, Sung Ho Park, "ROOM-BRIDGE: A Vertically Configurable Network Architecture and Real-Time Middleware for Interoprability between Ubiquitous Consumer Devices in Home," *Lecture Notes in Computer Science* 2218(p.232-251).

[11] T. Harrison, D. Levine, and D. Schmidt, "The Design and Performance of a Real-Time CORBA Event Service," *in Proceedings of the 1997 Conference on Object-Oriented Programming Systems, languages and Application(OOPSLA)*. Atlanta, Georgia, USA: ACM Press, 1997, pp.184-200.

[12] OmniORB CORBA, http://omniorb.sourceforge.net.

[13] Iona Technologies, "Orbix 3 Product Family," Iona Technologies, Whire Paper April 1999.

신 준 헌(Jun-Heon Shin) 학생회원

2002년 2월 : 경북대학교 전자공학과 졸업
2004년 2월 : 경북대학교 전자공학과 석사

<관심분야> 전자공학, 통신공학, 광통신 공학

박 준 호(Jun-Ho Park) 비회원

1998년 2월 : 경북대학교 전자공학과 졸업
2000년 2월 : 경북대학교 전자공학과 석사
2000년 3월~현재 : 경북대학교 전자공학과 박사과정

<관심분야> 홈 네트워, 미들웨어, 실시간 시스템

강 순 주(Soon-Ju Kang) 종신회원

1983년 2월 : 경북대학교 전자공학과 졸업
1985년 2월 : 한국과학기술원 전자계산학과 석사
1995년 2월 : 한국과학기술원 전자계산학과 박사

1985년 ~ 1996년 : 한국원자력연구소, 핵인공지능 연구실 선임연구원(과책), 전산정보실 선임연구원(실장)
1996년 ~ 현재 : 경북대학교 전자전기공학부 정보통신전공 부교수
2000년 ~ 2001년 : University of Pennsylvania. Dept. of CIS. 방문연구교수

<관심분야> 실시간 시스템, 임베디드 시스템, 지식기반시스템

최 준 용(Jun-Yong Choi) 비회원

1984년 2월 : 전남대학교 전산통계학과 졸업
1991년 2월 : 전남대학교 전산통계학과 이학석사
2001년 8월 : 전남대학교 전산통계학과 이학박사

1991년 5월 ~ 2001년 8월 : 목포대학교, 전남대학교 시간강사
1992년 2월 ~ 2001년 1월 : 광주시교육청 전산자료 계장
2001년 9월~현재 : 경북대학교 전자전기공학부 계약교수

<관심분야> 소프트웨어공학, 운영체제, 임베디드 시스템