韓國컴퓨터情報學會 論文誌
第16卷 第9號, 2011. 9.

2011-16-9-1-6

상황인식 미들웨어를 위한 트랜스듀서 인터페이스 프로토콜 설계

장 동 욱*, 손 석 원**, 한 광 록***, 선 복 근****

A Design of Transducer Interface Protocol for Context-aware Middleware

Dong-Wook Jang *, Surgwon Sohn**, Kwang-Rok Han***, Bok-Keun Sun****

요 약

사용자가 필요로 하는 서비스를 제공하기 위해서는, 일상생활 곳곳에 편재된 센서가 수집한 각종 환경 정보를 효율적으로 상호 공유하여 주변 상황을 인식하는 기술이 필요하다. 그러나 이를 위한 센서의 종류는 다양하고 각각의 센서는 고유의 특성과 서로 다른 방식으로 통신을 하기 때문에 센서의 활용에 제한이 있다. 이에 센서와 네트워크 계층과의 통신 프로토콜 표준화를 위한 IEEE 1451이 발표되었다. 그러나 IEEE 1451은 트랜스듀서(Transducer)의 표준화를 위한 프로토콜이므로 미들웨어에 접속되지 않는다. 본 논문에서는 XML을 이용하여 주변상황 정보를 얻기 위한 프로토콜을 정의함으로써 상황인식 미들웨어에 연결되는 트랜스듀서 인터페이스 및 응용 인터페이스 프로토콜을 제안한다. 그리고 서로 다른 센서와 응용 프로그램을 이용한 교량 건전성 감시 시스템과 철로 감시 시스템을 구현하고 제안한 인터페이스 프로토콜의 효용성을 확인하였다.

▸ Keyword : 트랜스듀서 인터페이스 프로토콜, 상황인식 미들웨어, 구조물 건전성 감시

Abstract

Context awareness technologies are based on efficient sharing of environment information of ubiquitous sensors in everyday life, and users require this awareness technologies to get quality of services. However, the application has been restricted due to its varieties of sensors and many different methods of communications. Therefore, IEEE 1451 standard has been published to

• 제1저자 : 장동욱 • 교신저자 : 손석원
• 투고일 : 2011. 05. 11, 심사일 : 2011. 06. 12, 게재확정일 : 2011. 07. 11.
* 호서대학교 컴퓨터공학과(Dept. of Computer Engineering, Hoseo University)
** 호서대학교 벤처전문대학원(Graduate School of Venture, Hoseo University)
*** 호서대학교 컴퓨터공학부(Dept. of Computer Engineering, Hoseo University)
**** 호서대학교 공학교육혁신센터(Engineering Education Research Center, Hoseo University)
※ 이 논문은 2010년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2010-0024242)

interface between sensors and network layer. But it does not connect to a middleware because IEEE 1451 is for transducer standards. This paper presents a transducer and application interface protocol which connect to the context-aware middleware by defining a protocol to obtain context information using XML. We have implemented a bridge health monitoring system and railroad monitoring system in which different sensors and users' application are used to prove the efficacy of proposed interface protocols.

▸ Keyword : Transducer Interface Protocol, Context-aware Middleware, Structural Health Monitoring

I. 서 론

최근 급격한 기술 발전추세에 따라 스마트폰과 스마트 TV 등과 같이 컴퓨터 기술이 최신의 통신기술(Bluetooth, Zigbee, Wifi, LTE, WIMAX)과 접목됨으로써 다양한 분야에서 디지털 융합(Digital Convergence)기술이 우리 삶을 혁명적으로 바꿔 놓고 있다. 제록스 팔로 알토(Xerox Palo Alto) 연구소의 마크 와이저(Mark Weiser)가 1988년 정의한 유비쿼터스 컴퓨팅[1]이 빠르게 현실화 되고 있다. 사용자가 필요로 하는 서비스를 제공하기 위해서는, 일상생활 곳곳에 편재된 센서 및 컴퓨터들이 수집한 각종 환경 정보를 효율적으로 상호 공유하여 주변 상황을 알아내는 상황 인식 기술이 필요하다[2]. 상황 인식 기술은 사용자를 중심으로 하는 주변 환경과 사용자간 혹은 사용자와 정보 기기간의 상호 관계를 지능적, 능동적으로 선택하여 지원해 줌으로써, 사용자로 하여금 정보 획득 및 실행을 보다 용이하도록 한다. 이러한 기술은 "일상 환경 속에 편재된 언제 어디서나 이용 가능한 컴퓨팅 환경"인 유비쿼터스 컴퓨팅 환경에 가장 중요한 핵심 기술 중에 하나이다[3]. 이러한 유비쿼터스 컴퓨팅 사회를 성공적으로 구축하기 위해서는 센서노드 하드웨어, 센서 네트워크, USN 미들웨어, 그리고 USN 응용 서비스 등의 USN 핵심기술의 개발이 반드시 필요하게 된다[4]. 특히 USN 응용시스템과 센서노드 중간 부분에 위치하여, 이 둘 간의 유연한 통합을 지원할 수 있는 USN의 인터페이스 기술이 중요하지만 국내외 많은 USN 응용시스템은 센서의 종류가 같지 않아 이를 지원하기 위해 미들웨어가 중복 개발되고 있어 유비쿼터스 저변확대에 저해가 되고 있다.

본 논문에서는 유비쿼터스 컴퓨팅에서 XML을 이용하여 센서의 데이터 포맷을 정의함으로써 이기종간의 USN 센서에서 동일한 방법으로 상황정보를 수집하고 처리할 수 있는 상황인식 시스템을 이용한 구조물 건전성 감시 시스템을 제안한다.

II. 관련 연구

유비쿼터스 시대에 접어들면서 주변 환경을 감지하고, 감지된 정보를 신호로 변환하여 여러 상황인식 시스템으로 연결된다. 이런 센서들은 여러 이벤트를 감지하며 산업, 차량, 가정, 의료, 농업 등의 모든 분야에서 다양하게 사용되고 있다. 하지만 기존의 센서들은 지금까지 주로 센싱 작업 그 자체에 관점을 두고 개발되어 왔다. 그러나 센서의 종류는 무한하고 각각의 센서는 고유의 특성과 서로 다른 연결 방식을 가지고 있기 때문에 사용자가 센서들을 일일이 확인한다는 것은 매우 어렵고, 높은 비용을 필요로 한다. 이를 해결하기 위해 센서들의 연결을 위한 일반적인 표준이 필요했고 IEEE 1451 Standard[5]가 제정되었다. IEEE 1451.1 에서는 NCAP(Network Capable Application Processor)[6]의 모델을 정의하며 스마트 센서 노드들의 시스템으로의 통합을 용이하게 한다. IEEE 1451.2에서는 TII(Transducer Independent Interface)를 이용하여 트랜스듀서와 NCAP간의 하드웨어 통신 채널에 대해 정의한다[7]. IEEE 1451.3은 분산 멀티 시스템을 위한 TEDS(Transducer Electronic Data Sheet) 포맷을 지원하고 버스 구조를 지원하기 위해 각 계층 별 통신 프로토콜을 정의하였다[8]. IEEE 1451.4는 혼합모드 통신 프로토콜(MMI)과 보안된 TEDS 형식을 지원하여 PnP의 장점을 부각시켰다[9]. IEEE 1451.5는 트랜스듀서와 NCAP 사이의 무선 인터페이스 연결에 대해 정의하였다[10]. 또한 IEEE 1451.1 표준에서의 NCAP 목적은 센서 노드와 네트워크 사이에 가교 역할을 하여 센서 네트워크 시스템 개발 및 통합을 용이하게 하는데 있다. 센서 노드와의 통신을 위해서는 센서에 대한 정보를 필요로 하게 되는데 이 정보는 IEEE 1451.4에서 정의한 템플릿 형식을 참조한다. 템플릿은 센서나 액추에이터(Actuator) 타입에 따라 필요한 항목들을 분류한 것으로 템플릿 ID에 따라 각기 다른 형식을 가진다.

이러한 IEEE 1451을 이용한 시스템 구현에 관한 연구가 진

행 되고 있다. UML(Unified Modeling Language)와 자바를 이용하여 IEEE 1451.0과 1451.5의 표준 구현[11], 모듈방식의 하드웨어와 소프트웨어의 구조에 대해 IEEE 1451.5 기반 고속 스마트 센서의 TIM (Transducer Interface Module) 개발[12], IEEE 1451.5 기반 무선 Ad-hoc 센서 네트워크 구현 [13], NCAP와 WTIM(Wireless Transducer Interface Module)의 구현[14]등이 그 예이다.

그러나 위의 IEEE 1451은 센서와 센서의 정보를 받아 처리하는 시스템의 표준화에 대해서 논의가 되어 있고, 이러한 센싱 정보를 받아 처리한 후, 이를 애플리케이션 계층에 전달하는 데이터에 대해서는 정의가 되어 있지 않다. 즉, 센서에 의해 수집된 상황정보를 수집하여 처리한 후 이를 사용자의 애플리케이션에 전달하기 위해서는 애플리케이션과의 별도의 인터페이스 프로토콜이 필요하다.

또한 개별 센서의 경우 트랜스듀서 채널 TEDS를 이용하여 센서의 정보를 지능화 하는 것은 매우 어려울 수 있다[15]. 각각의 센서에서 포함시켜야 하는 데이터시트 정보가 복잡할 경우, IEEE 1451.0의 트랜스듀스 채널 TEDS는 그 정보를 다 수용하지 못한다[15]. 또한 아직까지 IEEE 1451를 지원하지 못하는 센서의 경우 이를 해결하기 어렵다.

그러나 XML을 이용하여 이기종 RFID/센서 디바이스를 동적으로 관리함으로써 상황에 맞는 서비스를 제공할 수 있다[16]. 이에 본 논문에서는 XML을 이용하여 트랜스듀서 프로토콜과 애플리케이션 프로토콜을 일원화하여 간결하게 정의함으로써, 다양하고 복잡한 센서에 대해 통일된 프로토콜을 제공하여 상황인식 시스템의 플랫폼에서 독립시키며, 더 나아가 상황인식 시스템에서 애플리케이션과의 프로토콜도 트랜스듀서 프로토콜과 동일한 방법으로 센서의 상황 정보를 정의하도록 함으로써 USN 환경에서 다양한 응용 서비스에 보다 쉽고 빠르게 적용할 수 있는 미들웨어를 기반으로 구조물 건전성을 감시하는 시스템을 구현한다.

III. 인터페이스 프로토콜 설계

1. 인터페이스 프로토콜을 위한 시스템 구조

이기종간의 센서에서 수집된 정보를 토대로 상황을 인식하는 미들웨어와 응용 시스템은 그림 1과 같이 구성한다. 본 논문에서는 그림1과 같은 상황인식 응용시스템을 이용하여 서로 다른 센서와 애플리케이션을 사용하는 두 종류의 구조물 건전성 감시 시스템을 설계한다. 구조물 건전성 감시 시스템은 이기종 센서에서 상황 정보를 전송받아 이를 미들웨어로 전달하는 상황수집 에이전트(Wrapper Agent), 사용자로부터 또는 미들웨어로부터 명령을 받아 다양한 장치를 제어하는 장치제어 에이전트(Control Agent), 그리고 사용자가 상황정보를 모니터링 하고 제어하는 사용자 프로그램(Application)과 미들웨어로 구성한다.

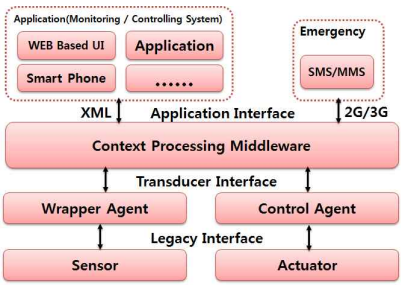

그림 1. 상황 인식 시스템의 구성도
Fig. 1. Architecture of context-aware system

상황 수집 에이전트는 다양한 센서나 장치에서 얻어진 상황정보를 미들웨어에서 요구하는 데이터 형식으로 변환하여 미들웨어로 전송하고, 미들웨어의 일부 명령을 받아 센서나 장치로 전송하는 중계기 역할을 담당한다. 장치 제어 에이전트는 장치를 제어하는 에이전트로서, 미들웨어에서 전달된 제어 명령어를 연결된 장치의 명령어로 변환하여 전달한다. 사용자 프로그램은 미들웨어에서 데이터를 받아 사용자에게 표시하고, 사용자의 명령어를 받아 미들웨어로 전달하는 역할을 담당한다.

2. 인터페이스 프로토콜을 위한 미들웨어

수집된 상황정보를 인식하고, 장치를 제어하며 사용자와의 인터페이스를 담당하는 상황인식 미들웨어의 구조는 그림2와 같다.

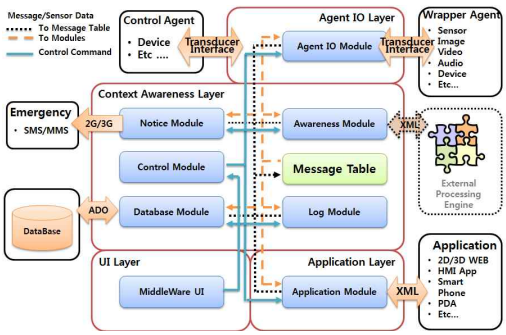

그림 2. 상황 인식 미들웨어의 구조
Fig. 2. Architecture of context-aware middleware

미들웨어는 에이전트와의 효율적이고 규격화된 정보 교환

을 위하여 XML로 정의된 프로토콜을 사용하고 있다. 상황 수집 에이전트를 통해 수집된 상황정보는 미들웨어의 메시지 데이터로 저장되어 각 모듈에서 처리된다. 사용자(WEB Based UI, 응용프로그램, 스마트폰 등)는 다양한 인터페이스를 이용하여 상황정보를 모니터링하고 제어 한다.

에이전트와 사용자 프로그램의 데이터를 처리하고 관리하는 미들웨어는 그림2와 같이 7개의 모듈과 1개의 메시지 테이블, 그리고 사용자 UI로 구성되어 있다.

다양한 상황인식을 위한 미들웨어는 크게 에이전트 IO 계층과 상황 인식 계층, 그리고 응용 계층으로 구분된다. 에이전트 IO 계층은 외부 에이전트 데이터의 입출력을 담당하는 계층으로서, 전달되는 모든 데이터는 미들웨어의 에이전트 IO 모듈을 통하여 메시지 형태로 변환된 후, 목적지의 모듈을 메시지에 표기한 후, 큐의 저장소로 이뤄진 메시지 테이블에 저장된다. 응용 계층은 사용자 프로그램과의 데이터 입출력을 담당하는 계층으로 응용 모듈이 외부 데이터의 요청을 받아 미들웨어의 메시지를 외부 사용자의 요청에 맞게 가공한 후 전송하는 역할을 담당한다. 미들웨어 메시지로 변환된 상황정보는 상황 인식 계층의 통지(Notice), 인식(Awareness), 데이터베이스(Database), 로그(Log) 모듈에서 본인에게 수신된 메시지를 메시지 테이블에서 가져와 임무를 수행하는 구조이다. 각각의 모듈별 요약은 표1과 같다.

표 1. 상황인식 미들웨어의 모듈
Table 1. Modules of context-aware middleware

Module	Running Based	dependence
Control	Thread	Independence
UI	Event	Depend on Control
Notice	Message Queue	Independence
Awareness	Message Queue	Independence
Database	Message Queue	Independence
Log	Message Queue	Independence
Application	Event Loop	Independence
Agent IO	Event Loop	Independence
MessageTable	Queue Storage	Depend on Modules

표1에 나열된 모듈의 역할을 간단히 기술 하면 다음과 같다.

제어 모듈(Control Module) : 각 모듈의 이상여부와 메시지의 이상여부를 감지하여, 이상여부 발생 시 정해진 규칙에 의해 조치함으로써 미들웨어의 전체적인 운영을 담당한다.

미들웨어 UI : 각 모듈별 현황과 메시지 처리 현황을 사용자가 모니터링 할 수 있도록 표시하는 역할을 담당한다.

통지 모듈(Notice Module) : 응급상황이 인지되었을 경우 이 모듈에 정의된 방법에 따라 사용자에게 통지하고, 사용자에 의해 정해진 규칙에 의해 상황에 대처하는 모듈이다.

인식 모듈(Awareness Module) : 상황정보가 저장된 메시지를 분석하고 현재 상황을 판단하는 모듈이다. 필요에 따라 외부 상황인식 엔진을 이용할 수 있으며, 이때는 상황인식 엔진과 커뮤니케이션을 담당하는 역할을 한다.

데이터베이스 모듈(Database Module) : 미들웨어는 상황정보를 인식하도록 설계되어 있으며, 상황정보 보관은 이미 효율성과 신뢰성이 확보된 외부의 전용 데이터베이스(Oracle, MSSQL, MySQL등)를 이용한다. 따라서 미들웨어는 외부 데이터베이스와의 연동이 필요하며, 데이터베이스 모듈이 이 역할을 담당한다. 이 모듈은 모든 메시지의 내용을 외부 데이터베이스에 맞게 변환 후 저장하고, 요청에 따라 데이터를 읽어오는 역할을 담당한다.

로그 모듈(Log Module) : 미들웨어의 모든 메시지의 흐름에 대한 기록과 함께 각각의 모듈별 메시지 처리 결과를 기록하는 모듈이다. 미들웨어의 신뢰성 확보를 위한 모듈이다.

응용 모듈(Application Module) : 사용자가 상황 정보에 대하여 모니터링 하기 위해서 다양한 플랫폼의 모니터링 소프트웨어가 요구되며, 이때 모니터링 소프트웨어와 통신하는 모듈이다. 다양한 플랫폼에 대응하기 위해 미들웨어 내부의 메시지를 XML로 변환하여 외부 플랫폼에 전달하고, 외부 플랫폼에서 수신된 XML을 내부 메시지에 맞게 변환하는 역할을 담당한다.

에이전트 IO 모듈(Agent IO Module) : 외부의 상황 수집 에이전트와 장치제어 에이전트의 데이터 송수신 역할과 함께, 에이전트에서 수신된 데이터와 미들웨어의 메시지 형태의 데이터를 상호 변환하는 역할을 담당한다. 또한 외부 에이전트들의 상태 여부를 정기적으로 확인하여 외부 에이전트의 상태를 전달하는 역할을 담당한다.

메시지 테이블(Message Table) : 에이전트와 사용자 프로그램으로 부터 데이터를 받아 객체 형태의 메시지로 변환 후 각각의 모듈이 수신할 수 있도록 구성된 메시지 큐 저장소이다. 메시지의 형태는 상황 데이터의 규격을 정의한 XML을 기반으로 생성된다.

표1과 같이 UI와 메시지 테이블을 제외한 각각의 모듈은 DLL(Dynamic Linking Library)형태로 완벽하게 독립적으로 구성되어 있어, 기능 추가 및 수정을 위해 해당 모듈만 수정할 수 있도록 설계되어 있다. 예를 들어 데이터베이스 시스템을 Oracle에서 MS-SQL로 변경하고자 한다면 데이터베이스 모듈만 수정하고, 응급상황 감지 시 사용자 휴대폰의 SMS 통지에서 해당상황의 영상 등을 담아 통지하기 원한다면 통지 모듈에서 SMS를 MMS로 수정하면 해당 기능을 다른 모듈의 수정 없이 즉시 적용할 수 있도록 구성되어 있다.

3. XML 기반의 인터페이스 프로토콜의 개발

상황인식 미들웨어는 에이전트와 사용자 응용 프로그램을 위하여 크게 센서와 액추에이터(Actuators), 그리고 트랜스듀서 인터페이스(Transducer Interface)를 사용한다. 이기종간의 서로 다른 데이터 전송 방법을 XML에 정의하여 센서 또는 액추에이터의 데이터 전송을 한다.

3.1 트랜스듀서 인터페이스(Transducer interface)

트랜스듀서는 센서와 액추에이터(Actuator)를 말한다. 효율적인 상황 인식 응용 시스템은 이기종간의 센서나 장치와의 데이터 송수신이 가능해야 한다. 에이전트는 센서와 같이 주로 데이터를 수집하는 래퍼 에이전트, 액추에이터의 상태 또는 센서의 값에 의해 액추에이터를 제어하는 제어 에이전트로 크게 나뉜다. 미들웨어와 에이전트와의 통신은 효율적이고 규격화된 정보 교환을 위해 새로 설계된 전용 프로토콜을 이용한다. 에이전트의 연결된 센서와의 데이터 송수신은 센서에 정의된 규약을 이용하여 상황 정보를 수집하고, 미들웨어로부터의 명령을 액추에이터에 전달한다.

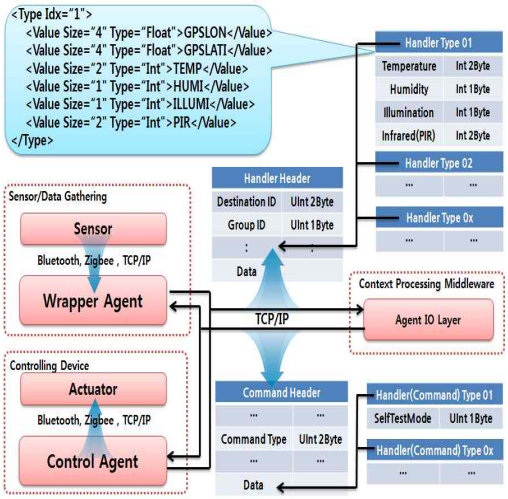

그림 3. 에이전트와 미들웨어의 통신 방법
Fig. 3. Communication method between agents and middleware

미들웨어와 에이전트의 통신은 그림3과 같이 트랜스듀서 인터페이스 프로토콜을 사용한다. 이 프로토콜은 헤더와 데이터를 포함하여 구성된다. 헤더는 데이터 종류에 관계없이 모든 패킷 앞에 붙어서 전달된다. 헤더는 표2와 같다.

표 2. 시스템 프로토콜 헤더
Table 2. System protocol header

Name	Size	Type
Destination ID	2 Byte	Unsigned Int
Group ID	1 Byte	Unsigned Int
Source ID	1 Byte	Unsigned Int
Data Type	2 Byte	Unsigned Int
Date : Year	2 Byte	Unsigned Int
Date : Month	1 Byte	Unsigned Int
Date : Day	1 Byte	Unsigned Int
Time : Hour	1 Byte	Unsigned Int
Time : Minute	1 Byte	Unsigned Int
Time : Second	1 Byte	Unsigned Int
Time : 10 MilliSec	1 Byte	Unsigned Int
Counter	2 Byte	Unsigned Int
Length (Size)	4 Byte	Unsigned Int
Data	Length - 20	Object

Destination ID : 데이터의 최종 목적지 ID이다. 데이터가 최종 목적지로 전송되기 위해 다른 에이전트를 이용해야 할 경우, 1바이트는 중계지 ID, 남은 1바이트는 최종 목적지 ID가 된다. 최종 목적지는 일반적으로 미들웨어이다.

Group/Source ID : 데이터를 송신하는 에이전트의 그룹과 그룹 내의 고유 ID이다. 에이전트는 그룹을 이용해 관리할 수 있다. 또한 출발지의 IP는 TCP/IP 내부 헤더의 정보를 이용한다.

Data Type : 헤더에 붙여 전송될 데이터의 구조를 표시하기 위해 사용한다. 각각의 데이터마다 고유의 인식표를 가지고 있으며, 이를 이용하여 동시에 다양한 정보를 송수신할 수 있다.

Counter : 에이전트에서 전송하는 데이터의 순번이다. 미들웨어는 이 순번을 이용하여 전송 순서를 확인한다.

Length : 헤더의 크기와 데이터의 크기를 포함한 실제 패킷의 전체 데이터 크기이다. 미들웨어에서는 이 값을 기준으로 데이터를 구분한다.

표2의 헤더에서 데이터 타입은 XML로 정의된 데이터의 종류를 의미한다. 데이터 타입은 2 바이트의 부호 없는 정수 크기이므로 최대 65536 종류의 데이터를 처리할 수 있다. 데이터 타입을 정의하기 위한 XML의 기본 구조는 그림4와 같다.

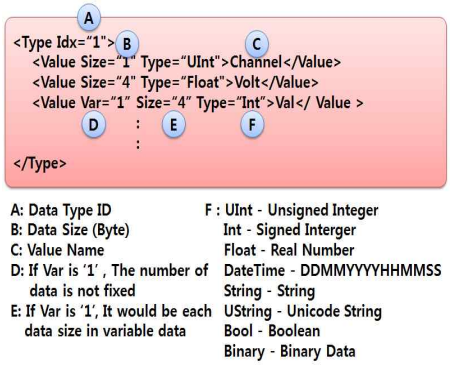

그림 4. 상황 정보 정의를 위한 XML의 구조
Fig. 4. XML structure for definition of context information

그림4에 의하면 새로 정의할 데이터 타입은 1번으로 정의한다. 1번 데이터 타입에는 1바이트 크기의 부호 없는 정수 데이터 'Channel'과 4바이트 크기의 실수 데이터 'Volt', 4바이트 크기의 정수 데이터 'Val'로 구성된다. 특히 'Val' 데이터는 Val값이 1이면 데이터가 한 개 이상 전달되는 가변 데이터이다. 그림4의 경우 4바이트 크기의 실수 데이터 'Volt' 뒤에 최소 한 개 이상의 4 바이트 크기의 정수 데이터 'Val'로 이뤄져 있음을 정의하고 있다. 'Val'의 개수는 전체 데이터 크기를 이용하여 계산한다.

위와 같은 방법으로 데이터 타입을 정의한다면 상황정보를 수집하기 위한 센서나 장치에서 사용하는 프로토콜과 관계없이 상황 인식 시스템이 모든 형태의 상황 정보 데이터를 수용할 수 있을 것이다.

3.2 애플리케이션 인터페이스

최근 급격히 증가하는 스마트폰과 웹서비스에서 데이터를 모니터링 하고 액추에이터를 제어하기 위해서는 각각의 서비스에 알맞은 프로토콜을 개발하거나, 제조사가 정의한 프로토콜을 이용해야만 했다. 더 나아가 구글의 안드로이드나 애플의 iOS와 같은 모바일 운영체제를 이용하는 애플리케이션 등에서는 데이터베이스에 직접 접근할 수 없기 때문에 이를 대체하는 최상의 방법으로 XML을 이용하고 있다.

미들웨어서도 그림5와 같이 기존의 미들웨어가 이용하였던 정의된 프로토콜과 제조사에서 정의한 프로토콜 외에도, 미들웨어에 마이크로 웹서버를 탑재하여 HTTP 프로토콜 기반의 XML 데이터를 활용한 데이터 전송 방법을 이용함으로써 플랫폼에 제약 없이 적은 비용으로 모니터링 및 제어 애플리케이션 개발이 가능하다.

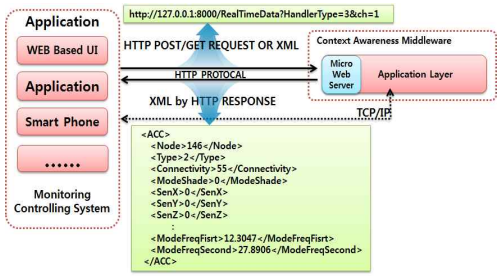

그림 5. 사용자 프로그램과 미들웨어의 통신 방법
Fig. 5. Communication method between applications and middleware

사용자 프로그램은 미들웨어에게 웹페이지를 요청하듯 표3과 같이 데이터를 애플리케이션에서 미들웨어로 요청한다.

표 3. 사용자 프로그램의 데이터 요청 예
Table 3. An example of data request for applications

http://127.0.0.1:8000/RealSensorData.xml?HandlerType=4&ch=2

표3에서는 127.0.0.1에서 실행되고 있는 미들웨어에게 4번 타입, 그리고 2번 채널의 실시간 데이터를 XML로 요청한다. 즉, 사용자 프로그램은 미들웨어로 HTTP의 GET 방식을 이용하여 필요한 인자를 전달하여 데이터를 요청하고, 필요하다면 POST 방식으로 XML 형식의 데이터를 전달하여 데이터를 요청한다.

표 4. 사용자 프로그램의 요청에 따른 미들웨어의 응답 예
Table 4. An example of middleware response for applications request

```
GET /RealSensorData.xml HTTP/1.1 200 OK
Accept: */*
Accept-Language: ko-KR
Accept-Encoding: gzip, deflate
Content-Type: application/xml
Accept-Charset: windows-949,utf-8;q=0.7,*;q=0.3
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C; InfoPath.3; Creative AutoUpdate v1.40.01)
Connection: keep-alive
Date: Wed, 20 Apr 2011 21:59:35 GMT
<?xml version="1.0" encoding="UTF-8"?>
<RealSensorData type="4" ch="2" TableNo="4">
    <Destination>0</Destination>
    <Group>0</Group>
    <Source>1</Source>
    <DateTime>2011-04-20 21:59:32</DateTime>
    <MilliSec>75</MilliSec>
    <Idx>17820</Idx>
    <Channel>1</Channel>
    <Value>1557.334</Value>
</RealSensorData>
```

사용자 프로그램의 요청을 받은 미들웨어는 표4과 같이 요청받은 실시간 데이터를 XML로 HTTP 헤더를 포함하여 전송함으로써 사용자 프로그램이 데이터를 받아 처리할 수 있도록 한다. 표4의 XML 데이터는 사용자 프로그램의 실시간 데이터 'RealSensorData.xml' 요청에 응답한 것으로 4번 타입의 2번 채널 데이터 "1557.334"를 전달하고 있다.

또한 HTTP는 요청을 받아 응답하기 때문에 100ms 이하의 실시간 처리에서는 HTTP 프로토콜을 활용하기에 적절하지 못하다. 따라서 위의 XML 데이터를 TCP/IP 프로토콜로 미들웨어에서 사용자 프로그램에 전송함으로써 HTTP의 한계를 보강하고 있다.

미들웨어의 상황정보 처리는 그림 6과 같이 처리된다.

센서 등에서 수집되는 상황 정보는 에이전트에 의해 미들웨어로 전송된 후, 미들웨어의 에이전트 IO 모듈을 통해 메시지로 변환되고 이 정보는 3개의 모듈을 목적지로 하여 메시지 테이블에 저장된다. 데이터베이스 모듈에서는 상황정보 메시지를 받아 데이터베이스에 보관하고, 처리 모듈에서는 상황정보의 이상 여부를 확인하며, 응용 모듈에서는 외부 사용자 프로그램에서 실시간 모니터링을 위한 XML로 변환된다. 이후 처리 모듈에서 상황정보의 이상 값이 인지되면 통지 모듈을 통하여 사용자에게 긴급 SMS를 발송하여 통보하고, 정해진 규칙에 따라 에이전트로 장치 제어를 위한 명령어를 전송한다.

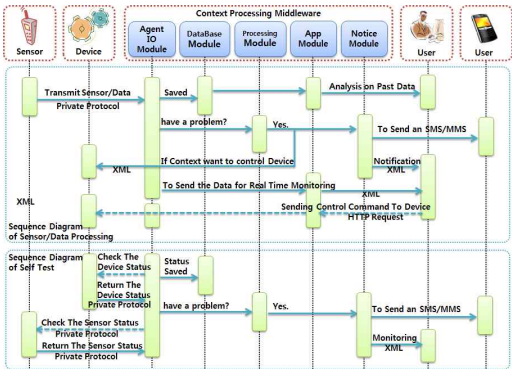

그림 6. 미들웨어의 상황정보 처리
Fig. 6. Context Information processing of middleware

사용자는 스마트폰과 웹사이트, PC 응용프로그램 등 다양한 플랫폼의 사용자 프로그램을 통하여 상황 정보를 모니터링하며, 직접 이를 이용하여 미들웨어를 거쳐 에이전트로 장치 제어 명령을 전달할 수 있다.

에이전트 IO 모듈은 주기적으로 에이전트와 사용자 프로그램의 상태를 확인하는 명령어를 전송한 후, 이들로부터 응답을 받아 상태를 능동적으로 확인하고, 처리 모듈이 이들 정보를 검토한다. 만약 에이전트에서 이상이 발견되면 사용자에게 통지하도록 한다.

IV 인터페이스 구현 및 실험

1. 상황인식 시스템의 인터페이스 정의

1.1. 시스템 환경

본 논문에서 제안한 이기종간의 센서와 장치를 효율적으로 상황인식 시스템에 적용하기 위하여 교량 감시 시스템의 모형과 철로 감시 시스템의 모형을 제작하고, 각각 센서를 부착하여 이기종의 두 시스템에서 효율적인 시스템 적용 여부를 실험하였다.

표 5. 실험에 사용된 시스템 환경
Table 5. System environment for experiment

항목	Spec
CPU	Intel Quad CPU Q6600
RAM	2 GB
OS	Windows XP SP3
DataBase	MySQL 5.5.11 for Windows
Middleware	Context Awareness Middleware

1.2 교량 감시 시스템의 인터페이스 정의

모형 교량에서는 Imote2의 3축 가속도센서 및 PZT 센서를 이용하여 교량의 변화에 따른 다량의 센서 데이터를 수집하여 실험하였다.

표 6. 모형 교량에 사용된 센서 종류와 개수
Table 6. Sensor type and quantities for a model bridge

Sensor	Sampling Rate	Channel
가속도 센서	250Hz	3 Ch
PZT 손상감지 센서	250Hz	3 Ch

표6에서 보듯 가속도 센서와 PZT 센서는 각각 초당 250개의 데이터를 수집한다. 실험에서는 가속도 센서 3개와 PZT 센서 3개를 사용하였으며 측정된 데이터를 미들웨어로 전송하는 에이전트를 개발하였다. 표6의 센서 중 가속도 센서의 데이터를 수신하기 위하여 데이터 타입 3번을 표7과 같이, 타입 4번을 표8과 같이 정의 하였다.

표 7. 가속도 센서를 위한 3번 데이터의 정의
Table 7. Definition of data type 3 for ACC sensor

```
<Type Idx="3">
    <Value Size="1" Type="UInt">Channel</Value>
    <Value Size="4" Type="Float">ValX </Value>
    <Value Size="4" Type="Float">ValY </Value>
    <Value Size="4" Type="Float">ValZ </Value>
</Type>
```

표7에 의하면 첫 1바이트에는 4개의 센서를 구분하기 위한 채널, 두 번째부터 네 번째 데이터는 4바이트의 실수 데이터로 구성되어 있다.

표 8. PZT 센서를 위한 4번 데이터의 정의
Table 8. Definition of data type 4 for PZT sensor

```
<Type Idx="4">
    <Value Size="1" Type="UInt">Channel</Value>
    <Value Size="1" Type="UInt">Cnt</Value>
    <Value Val="1" Size="4" Type="Float">Value</Value>
</Type>
```

표8에 의하면 첫 1바이트에는 4개의 센서를 구분하기 위한 채널, 두 번째 1바이트에는 전송할 데이터의 개수를 저장한 카운트(Cnt), 다음으로 카운트 개수만큼의 데이터를 실수 4바이트 크기로 각각 이어 붙여 전송한다.

1.3 철로 감시 시스템의 인터페이스 정의

모형 철로에서는 UBee430 센서를 이용하여 상황 정보를 수집하였고, Sollae System에서 생산한 ezTCP[17]라는 제어기를 이용하여 철로 전환 장치를 구현하였다.

표 9. 모형 철로에 사용된 센서 종류와 개수
Table 9. Sensor type and quantities for a model rail

Sensor	Sampling Rate	Channel
가속도 센서	5Hz	4 Ch
진동 센서	5Hz	6 Ch

표9와 같이 가속도와 진동센서를 이용하여 각각 초당 5개의 데이터를 수집하는 에이전트를 개발하였다. 총 센서는 10개를 이용하여 데이터를 수집하였다. 교량관리 시스템에 사용된 상황인식 미들웨어를 수정 없이 이용함으로써 이기종간의 데이터를 수용하는지 여부를 확인하였다. 표8의 센서중 가속도 센서의 데이터를 수신하기 위해 데이터 타입 14번을 표10과 같이 정의 하였다.

표 10. UBee430 가속도 센서를 위한 14번 데이터의 정의
Table 10. Definition of data type 14 for ACC sensor

```
<Type Idx="14">
    <Value Size="1" Type="UInt">Ex Temp</Value>
    <Value Size="1" Type="UInt">In Temp</Value>
    <Value Size="1" Type="UInt">Humidity</Value>
    <Value Size="4" Type="Float">Voltage</Value>
    <Value Size="2" Type="UInt">Illumination</Value>
    <Value Size="2" Type="UInt">Acceleration</Value>
</Type>
```

표10에 의하면 UBee430 가속도 센서에서 측정하는 6가지 측정값 외부온도, 내부온도, 습도, 배터리 전압, 조도, 가속도 값을 헤더 20 바이트와 데이터 11 바이트를 더한 모두 31바이트가 전송된다. 진동 센서도 가속도 센서와 유사한 방법으로 데이터 타입 15번을 정의하였다.

2. 구조물 감시 시스템의 실험

2.1 교량 건전성 감시 시스템

교량관리 시스템은 그림8과 같은 방법으로 모형을 만들어서 실험하였다.

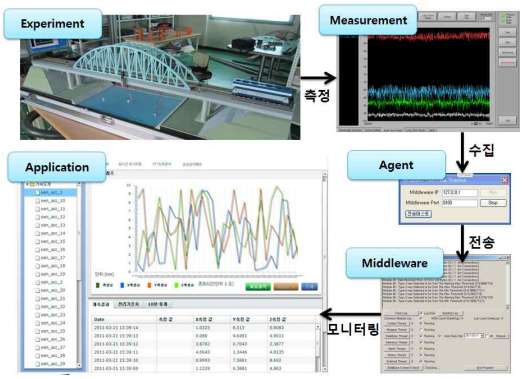

그림 7. 교량 건전성 감시 시스템의 실험
Fig. 7. An experiment of bridge health monitoring system

그림7의 모형교량에서 측정된 데이터는 6개의 센서에서 모두 1초당 1,500개의 센서 데이터가 에이전트에서 수집되었으며, 이는 센서 각각 1초간의 데이터를 모아 미들웨어로 전송되었다. 즉, 1초에 10개의 미들웨어 프로토콜에 의해 만들어진 패킷이 미들웨어로 전달되었으며, 각각의 패킷에는 250개와 100개의 데이터를 포함하여 전송되었다.

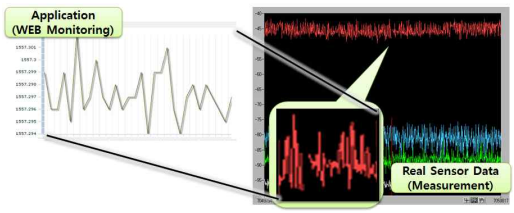

그림 8. 측정된 센서와 모니터링 값의 비교
Fig. 8. Comparison between real sensor data and web monitoring data

그림8의 왼쪽 화면은 웹기반의 실시간 모니터링 프로그램에서 HTTP 프로토콜로 전송받아 표현한 그래프이다. HTTP 프로토콜을 이용해서 초당 100회의 데이터를 요청하고 받을 수 없다. 따라서 웹 서버에서는 초당 2회로 평균값을 전송하고 클라이언트에서 수신하여 모니터링 한다. 실시간 감시라는 상황에서는 이것이 문제점으로 작용할 수가 있어서 이 점을 해결하기 위해 웹 클라이언트가 아닌 전용 수신 프로그램을 사용하였다. 이 때 수신 프로그램은 서버의 특정 포트에 직접 접속하여 소켓(Socket)으로 데이터를 실시간 전송 받아 표현할 수 있는데 그림 8의 우측 그래프가 그것이다.

2.2 철로 감시 시스템

철로관리 시스템은 그림9와 같은 방법으로 모형을 만들어서 실험하였다.

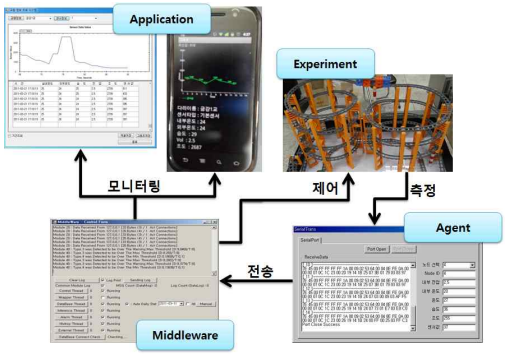

그림 9. 철로 감시 시스템의 실험
Fig. 9. An experiment of railroad monitoring system

모형 철로관리 시스템은 모두 10개의 UBee430 가속도와 진동센서를 이용하여 측정하였으며, 교량관리 시스템과는 달리 철로에 문제가 발생할 경우 이를 미들웨어가 스스로 감지하여 철로를 변경하는 시스템이 추가되었다.

UBee430 센서는 각각 초당 5개의 값을 전송하였다. 따라서 50개의 값이 실시간으로 에이전트를 통하여 미들웨어로 전송되었으며, 그림10은 PC용 모니터링 프로그램과 안드로이드 스마트폰용 모니터링 프로그램의 값이 일치하는 것을 확인할 수 있다. 모형 교량의 데이터보다 측정 개수가 현저히 적어 HTTP 만으로도 비교적 정확한 모양의 그래프를 확인할 수 있다.

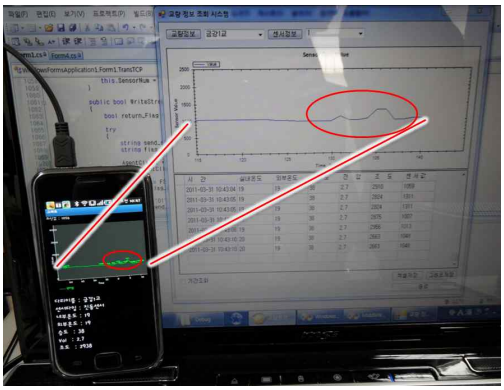

그림 10. 측정된 값의 스마트폰과 PC프로그램 비교
Fig. 10. Result view comparison between Smart-phone and PC application

그림11과 같이 센서의 값이 지정된 임계치보다 높을 경우 미들웨어에서는 통지 모듈을 통하여 사용자에게 SMS를 송신하여 통보하고, 에이전트 IO 모듈을 통하여 외부의 철로 변환 장치에게 철로를 A에서 B로 전환하라는 명령을 전달하여 철로 사고를 미연에 방지할 수 있다.

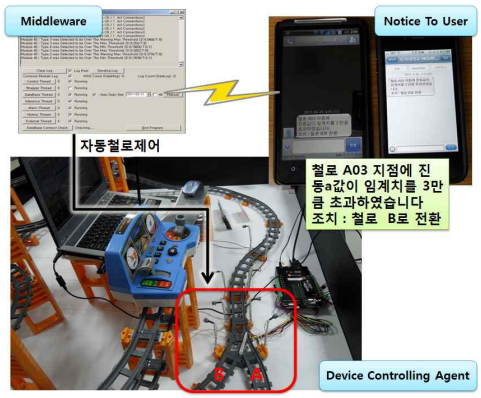

그림 11. 철로 감시 시스템의 응급상황 실험
Fig. 11. An experiment of emergency situation of railroad monitoring system

V. 결 론

본 논문에서는 상황인식 미들웨어에 여러 종류의 센서와 액추에이터를 접속시키기 위하여 비용이 많이 드는 하드웨어를 포함하는 표준화 설계를 따르지 않고 소프트웨어 에이전트

만을 구현함으로써 그 기능을 대신하는 트랜스듀서 인터페이스 에이전트를 구현하였다. 이 때 센서 또는 액추에이터가 서로 다르지만 이를 미들웨어와 규격화된 통신을 할 수 있는 트랜스듀서 인터페이스 프로토콜을 제안함으로써 미들웨어와 다양한 트랜스듀서간의 접속이 가능하였다. 이는 XML을 이용하여 트랜스듀서 인터페이스 에이전트의 데이터만을 정의함으로써 상황인식 미들웨어를 수정하지 않고 센서와 액추에이터를 사용할 수 있다는 것이다.

애플리케이션 인터페이스는 마이크로 웹서버를 미들웨어에 구현하고 XML 데이터를 지원함으로써 기존의 HTTP 브라우저와 데이터베이스 사용이 불가능한 스마트폰을 동시에 사용할 수 있게 하였다. 구현된 트랜스듀서 인터페이스 에이전트와 애플리케이션 인터페이스를 평가하기 위해서 센서 및 액추에이터가 많이 사용되는 교량 건전성 시스템과 철로 감시 시스템을 구현하고 적용시킴으로써 에이전트의 효용성을 입증하였다.

향후 연구과제로는 기존의 IEEE 1451 규약을 완전히 따르는 트랜스듀서 인터페이스 에이전트를 설계하는 것이다. 이것은 하드웨어의 사용을 없앰으로써 센서 및 액추에이터 인터페이스 개발에 비용 및 개발시간에 유리할 것으로 판단된다.

참고문헌

[1] M.Wesier. "Some Computer Science Issues In Ubiquitous Computing." Communication of the ACM, Vol. 36(7), pp 75-84, Jul. 1993.

[2] G. Banavar, A. Bernstein, "Issue and challenges in ubiquitous computing : Software infrastructure and desige challenges for ubiquitous computing applications," Communication of ACM, Dec. 2002.

[3] C. D. Kidd, R. Orr, G. D. Abowd, C. G. Atkeson, I. A. Essa, B. MacIntyre, E. Mynatt, T. E. Starner and W. Newstetter, "The Aware Home : A Living Laboratory for Ubiquitous Computing Research," Proc. of the 2nd Int'l. Workshop on Cooperative Buildings, Oct. 1999.

[4] M.S.Kim , Y.J.Lee, J.H.Park, "Trend of USN Middleware Technology", Trend analysis of Electronics and Telecommunications, Vol.22 No.3, pp 67-79, Jun. 2007.

[5] IEEE 1451 Standard working group, http://grouper.ieee.org/g roups/1451

[6] IEEE Inc., "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Network Capable Application Processor (NCAP) Information Model," IEEE Std 1451.1-1999, 2000.

[7] IEEE Inc., "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," IEEE Std 1451.2-1997, 1998.

[8] IEEE Inc., "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators-Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems," IEEE Std 1451.3-2003, 2004.

[9] Jones, Charles H., "IEEE 1451.4 Smart Transducer Template Description Language.", May 2004.

[10] IEEE Inc., "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," IEEE Std 1451.5-2007, pp.C1-236, Oct. 2007.

[11] Song, E.Y. and Kang Lee, "An implementation of the proposed IEEE 1451.0 and 1451.5 standards", Sensors Application Symposium, pp.72-77, Feb. 2006.

[12] Sweetser, D., Sweetser, V. and Nemeth-Johannes, J., "A modular approach to IEEE-1451.5 wireless sensor development", Sensors application Symposium, pp.82-87, Feb. 2006.

[13] Jungil Heo, Jaehwan Kim, Jungho Seo, Suyoung Lim, Junhong Ahn, Junsoo Ahn and Wooshik Kim, "A Study on the Implementation of a wireless Ad-hoc Sensor Network based on the IEEE 1451.5", Advanced Communication Technology, The 9th International Conference on, Volume 1, 12-14 pp.419-422, Feb. 2007.

[14] Se-moon Oh, Min-ha Keum, Dong-hyeok Kim, Jin-sang Kim, Won-kyung Cho, "Implementation of the Wireless Transducer Interface Module and NCAP architecture", The Journal of KOREA Information and Communications Society Vol.33 No.12, Dec. 2008.

[15] Jung-Hwan Lee, Dong-Jin Kim, Jeong-Do Kim, and Yu-Kyung Ham, "The Reference Model for Smart Web Sensor Based on IEEE 1451 and Web-service Using Gas Sensor", Journal of IEEK SC, Vol.45 No.6, Nov. 2008.

[16] Hyu-Chan Kim, Seok-Youn Koh, Wan-Ki Koh, Moon-Seok Yang, "Design and implementation of a dynamic management method for heterogeneous RFID/USN devices", Journal of The Korea Society of Computer and Information, V.14, No.7, pp.143-150, Jul. 2009.
[17] Sollae Systems, http://www.sollae.co.kr/

저 자 소 개

장 동 욱
2005 : 호서대학교 컴퓨터공학과 공학사.
2007 : 호서대학교 컴퓨터공학과 공학석사.
2011 : 호서대학교 컴퓨터공학과 공학박사
관심분야 : HCI, USN, Middleware
Email : coco@hcialb.net

손 석 원
1985 : 인하대학교 전자공학과 공학사.
1987 : 인하대학교 전자공학과 정보공학석사.
2007 : 인하대학교 컴퓨터정보공학박사
현 재 : 호서대학교 벤처전문대학원 부교수
관심분야: 무선센서네트워크, 이동통신, 제약만족최적화
Email : sohn@hoseo.edu

선 복 근
1999 : 호서대학교 컴퓨터공학과 공학사.
2001 : 호서대학교 컴퓨터응용기술학과 공학석사.
2006 : 호서대학교 컴퓨터공학과 공학박사
현 재: 호서대학교 공학교육혁신센터 교수
관심분야 : 정보검색, 임베디드시스템
Email : bksun@hoseo.edu

한 광 록
1989 : 인하대학교 정보공학박사.
현 재 : 호서대학교 공과대학 컴퓨터공학과 교수
관심분야 : 정보검색, HCI, e-Health, 시멘틱웹
Email : krhan@hoseo.edu