센서의 Transparency 를 지원하는 아키텍처 설계

정종태*, 정국상*, 최덕재**
전남대학교 전산학과
e-mail : {northstar, handeum}iat.chonnam.ac.kr*, dchoi@chonnam.ac.kr**

A Design of the Architecture for Sensors' Transparency

Jongtae Jeong, Kugsang Jeong, Deokjai Choi
Dept. of Computer Science, Chonnam National University

요 약

유비쿼터스 컴퓨팅 환경의 인프라는 센서, 미들웨어, 그리고 응용 프로그램으로 구성된다. 유비쿼터스 컴퓨팅 환경이 실현되기 위해서는 이 세 요소들이 상호 유기적으로 메시지를 전달해야 하며, 구성 요소를 구성하는 개체들에 대해서 존재 유무를 파악할 수 있어야 한다. 우리는 센서와 미들웨어간의 통신을 위한 컴포넌트를 쉽게 만들기 위하여 망 관리에서 사용되었던 SNMP 를 이용하였다. 하지만 SNMP 를 적용한 구조에서는 동적으로 변화는 센서에서 무방비한 입장이었다. 다시 말해서 동적으로 변하는 센서나 혹은 그 속성값들이 변할 때, 그 시스템의 전문가가 직접 통신 컴포넌트를 수정해야 했다. 또한 센서들이 접하게 될 플랫폼들이 여러 종류가 있기 때문에 센서를 만드는 벤더들 또한 여러 버전의 API 를 제공해야 한다. 따라서 본 아키텍처는 센서나 그 속성값들이 변할 때 사람에 의해서 조작하지 않고 시스템 스스로 설정하며 그 센서들을 사용할 수 있도록 하고, 벤더들의 수고를 덜어 줄 수 있는 아키텍처를 제안한다. 이러한 메커니즘은 기존 망 관리에서 사용했던 AgentX 에서 착안하였다. AgentX 프로토콜의 메커니즘을 이용함으로써 사용자의 입장에서 센서에 대한 Transparency 를 보장 받을 수 있으며, 부수적으로 센서를 만드는 벤더들은 여러 버전의 API 를 만들어야 할 수고를 덜게 되었다.

1. 서론

최근 몇 년 사이에 유비쿼터스[1][2]라는 용어가 대두되면서 사회적으로 많은 변화가 일어나고 있다. 유비쿼터스 컴퓨팅 환경은 인간 생활에 있어서 최적인 환경을 구축하기 위하여 환경이 인간의 상태와 주변 환경의 요소에 따라서 변화하게 된다. 이러한 유비쿼터스 컴퓨팅 환경을 구축하기 위한 인프라는 크게 센서, 미들웨어, 그리고 응용 프로그램 부분으로 구성되어 있다. 이렇게 구성된 인프라는 두 가지 핵심 기능이 가지고 있어야 한다. 그 첫째로 세 구성 요소들이 메시지 전달을 중심으로 서로 유기적으로 동작해야 하고, 둘째는 각 인프라 속에서 구성 요소들의 존재를 파악하고 있어야 한다. 즉 센서가 센싱한 데이터를 미들웨어로 보내고 미들웨어는 그 데이터를 수집, 가공하여 응용 프로그램에게 전달하는 기능을 가져야 한다. 또한 인프라를 구성하는 요소들은 자신이 관리하는 개체들을 파악하고 있어야 한다. 즉, 미들웨어는 도처에 있는 다수의 센서를 포함하고 새로운 센서가 추가되거나 기존에 있는 센서가 제거되었을 경우에 사용자 혹은 전문가의 개입 없이 미들웨어에서 이를 수용할 수 있어야 한다. 따라서 본 논문은 사용자나 전문가의 영향을 받지 않고 센서를 추가, 제거할 수 있도록 하는 메커니즘을 논하고자 한다.

현재까지 미들웨어에 대한 연구는 Dey 의 Context-Aware 미들웨어[3], CALAIS[4], Gaia[5], 그리고 Oxygen 프로젝트[6] 등 많이 진행되어 왔다. 이 미들웨어들은 센서와 통신하기 위한 독자적인 컴포넌트를 가지고 있다. 하지만 이러한 미들웨어에서도 센서를 추가할 때는 개발자가 미들웨어에서 제공하는 API 와 벤더에

서 제공하는 센서 API 를 참고하여 수정한다. 그 예로 Dey 의 Context-Aware 미들웨어에서 통신을 담당하는 컴포넌트인 Widget[3]을 들 수 있다.

여러 해 동안 망 관리를 위하여 여러 기법들이 사용되었고 표준화도 되었다. 그 중에서 SNMP[7]는 망 관리에 있어서 단순하면서 조작하기 쉬워서 널리 사용되고 있다. 이런 망 관리 기법을 유비쿼터스 컴퓨팅을 지원하는 미들웨어에 적용하면 현존하는 유비쿼터스 컴퓨팅을 지원하는 미들웨어보다 여러 장점이 있다. 첫째, SNMP 가 가지고 있는 연산자 중에서 Get, Set, Trap 연산자는 센서로부터 값을 가져올 때, 센서에 특정 값을 지정할 때, 그리고 센싱된 값을 통해 이벤트 처리할 때 사용하기에 적합하다. 둘째, SNMP 는 여러 기간 동안 사용되었기 때문에 관련 툴이 많이 존재한다. 따라서 쉽게 에이전트 개발이 가능하다는 것이다. 하지만 SNMP 를 적용한 센서와 미들웨어간의 통신 컴포넌트[8]에서도 다음과 같은 문제점이 발생된다. 이종의 센서가 추가되었을 경우 에이전트와 MIB 의 모양이 변하게 된다. 다시 말하면 센서와 연결되어 있는 에이전트는 그 센서에 해당되는 MIB 을 가지고 있을 것이다. 그런데 그 센서를 표현하는 속성값이 바뀌었을 경우 MIB 이 변경되어야 한다. 이에 함께 MIB 에 연결되어 있는 에이전트 또한 변경이 될 수 밖에 없다. 그 결과 에이전트를 만드는 개발자는 센서 추가될 경우 발생되는 에이전트와 MIB 의 변화에 즉시 대응해야 하고, 에이전트와 MIB 수정하는 과정에서 일시적으로 에이전트의 기능이 중지된다.

이러한 문제점을 해결하기 위하여 AgentX(Agent eXtensibility)[9]를 적용한다. AgentX 는 기존의 에이전트 부분을 마스터 에이전트와 서브 에이전트로 나눈다. 따라서 AgentX 의 기본적인 구조는 매니저, 마스터 에이전트, 서브 에이전트, 그리고 MIB 이 된다. 두 에이전트 중에서 센서와 직접 연결되어 있는 부분은 서브 에이전트가 된다. 따라서 센서가 추가되고 제거될 경우 상위 레벨, 즉 마스터 에이전트와 매니저에게는 아무런 영향을 주지 않게 된다. 그리고 각 센서들은 서브 에이전트에게 값을 전달할 경우에 특정 플랫폼에 자신의 API 을 구애 받게 된다. 따라서 벤더들은 특정 플랫폼이라는 문제를 극복하기 위하여 여러 플랫폼을 지원할 수 있는 여러 버전의 API 를 제공하기 된다. 이 경우 에이전트를 개발하는 자는 각 센서들의 API 를 알고 있어야 한다. 하지만 센서와 직접 연결되어 있는 서브 에이전트에 값을 전달하기 위해서 센서와 통신할 때 특정 플랫폼에 구속된 API 를 가진다. 이러한 문제 해결은 센서와 직접 연결되어 있는 서브 에이전트를 벤더에서 자신의 제품에 적합하게 개발하면 된다. 또한 미들웨어를 개발하는 개발자는 매니저와 마스터 에이전트를 서브 에이전트와 별개로 개발하면 된다. 그리하여 각 컴포넌트의 연결은 매니저와 마스터 에이전트간은 SNMP 프로토콜을 이용하며 마스터 에이전트와 서브 에이전트간은 AgentX 프로토콜을 사용하면 된다.

자유로운 센서의 추가와 삭제로 인하여 유비쿼터스 컴퓨팅 환경을 조성하는 과정에서 센서 연결에 대해서 유연성을 가질 수 있으며, 매니저 입장에서 보면 마스터 에이전트와만 통신을 하므로 서브 에이전트와 연결되어 있는 센서가 추가되었는지 제거되었는지를 알지 않아도 됨으로써 Transparency 효과가 있다. 또한 망 관리 기법을 적용하였기 때문에 다수의 센서들을 관리하게 될 때 보다 쉽게 접근할 수 있을 것이다.

본 논문에서는 센서 연결에 유연성을 위하여 망 관리 기법 중 AgenX 를 적용한 시스템 구조를 소개하고자 한다. 2 장에서는 시스템을 구성하는 요소들을 논하고, 3 장에서는 본 시스템이 동작되는 메커니즘을 설명한다. 마지막으로 4 장에서는 결론을 맺으면 본 논문을 마친다.

2. 시스템 구조도

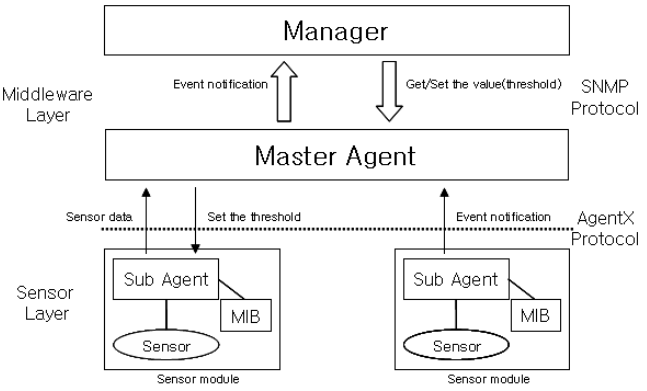

그림 1 시스템 구조

본 시스템은 유비쿼터스 미들웨어에서 센서 혹은 그 이하 하부 구조와 통신하는 컴포넌트에 중점을 두었기 때문에 매니저를 제외하고 미들웨어를 구성하고 있는 여타의 컴포넌트에 대해서는 언급하지 않는다.

본 시스템의 구조는 망 관리에서 적용되는 SNMP 에 그 뿌리를 두고 있다. 이는 매니저, 에이전트, 그리고 MIB 으로 구성된 기본적인 컴포넌트를 가지고 있다. 이 구조에서는 서론에서 지적했듯이 동적으로 변하는 관리 대상 객체에 효율적으로 대응하지 못한다. 따라서 AgentX 가 사용되는 마스터/서브 에이전트를 적용하였다. 우선 본 논문에서 제안한 시스템은 두 개의 계층-미들웨어와 센서-으로 나눈다. 그림 2 에서 보는 것과 같이 본 시스템은 2 개의 계층과 5 개의 컴포넌트들로 구성되어 있다. 그 첫째로 미들웨어 계층은 마스터 에이전트에서 보낸 데이터를 수집하거나 센서들을 조작, 관리 역할을 하는 매니저와 센서 계층에서 보내준 데이터를 1 차적으로 받은 마스터 에이전트가 위치한다. 그리고 센서 계층은 주변 환경에서 직접 데이터를 센싱하는 역할을 하며, 서브 에이전트, 센서, 그리고 센서에서 수집된 데이터를 저장하게 될 MIB 으로 이루어져 있다. 이는 센서 모듈이라는 이름으로 묶는다.

각 계층별로 통신하는 방법은 매니저와 마스터 에이전트간에서는 SNMP 프로토콜을 사용하며, 마스터 에이전트와 서브 에이전트간에서는 AgentX 프로토콜을 사용한다.

다음은 각 구성 요소들이 하는 역할을 설명한다.

2.1 컴포넌트 구성 요소

(1) 매니저(Manager)

매니저는 각종 센서로부터 응용 프로그램까지 전달되기 전에 센서에서 센싱된 데이터가 수집되는 컴포넌트이다. 또한 매니저는 각 에이전트들을 감시하고 조정할 수 있게 하는 인터페이스를 가지고 있다. 이 두 가지 역할을 자세히 살펴보면, 첫째로 매니저는 사용자 프로그램에서 요청한 정보를 구하기 위하여 Get 연산자를 사용하여 마스터 에이전트에게 SNMP Get Request 메시지를 보낸다. 그럼 마스터 에이전트는 해당 값을 Get Response 메시지를 통해 매니저에게 전달한다. 그리고 또한 그리고 수집된 데이터를 기초로 하여 응용 프로그램이 원하는 정보로 전환이 가능하다. 이는 응용 프로그램에서 원하는 값을 얻을 때는 Get 연산자를 사용하여 마스터 에이전트에게 데이터를 요청하게 되고 이벤트를 설정할 때에는 센서에 대해서 Set 연산자를 사용하여 값에 대한 조건을 생성한다.

(2) 마스터 에이전트(Master Agent)

마스터 에이전트는 서브 에이전트와 연결되어 있기 때문에 서브 에이전트에서 전달되는 값을 전달 받는다. 서브 에이전트에서 받은 데이터를 매니저에게 전달한다. 또한 자신에 연결되어 있는 서브 에이전트에 대한 목록을 가지고 있다. 서브 에이전트가 추가될 경우나 혹은 제거되었을 경우 마스터 에이전트는 서브 에이전트를 자동으로 추가하고 제거한다. 따라서 마스터 에이전트는 현존하는 서브 에이전트의 리스트와 매니저가 원하는 정보를 제공하기 위해서 서브 에이전트와 연결되어 있는 MIB 리전(region)를 가진다.

(3) 서브 에이전트(Sub Agent)

센서 혹은 컴퓨팅 장치에서 센싱된 데이터를 1 차적으로 받는다. 서브 에이전트는 Get 연산자를 통해 센싱된 값을 가져 올 수 있으며, 사용자가 설정한 값이 센싱된 데이터와 비교하여 그 범위를 벗어날 경우 마스터 에이전트에게 이벤트가 발생하였다고 통보할 수 있다. 그리고 서브 에이전트는 마스터 에이전트와 세션 연결이 이루어지면 주기적으로 마스터 에이전트에게 PING 메시지를 보내면서 마스터 에이전트가 존재를 파악할 수 있다.

(4) MIB(Management Information Base)

센서에 대한 정보를 관리하는 방법은 객체(Object)로 정보를 관리한다. 각 정보를 하나의 객체로 하여 객체들의 계층적 구조로 센싱한 정보(상황 정보)를 저장하고 검색할 수 있도록 하는데, 이런 객체들의 모임을 관리 정보 베이스(Management Information Base: MIB)라고 한다. 매니저는 에이전트의 MIB 객체의 값을 검색함으로써 센서에 대한 감시 기능을 수행하는 것이다. 또한 MIB 은 에이전트에게 특정 동작을 하게 할 수 있으며, 특정 변수들의 값을 변경하거나 에이전트의 구성 설정도 변경시킬 수 있다.

(5) 센서(Sensor)

센서는 주변 환경에 널리 퍼져 있으며, 주변 환경에 대한 데이터를 생성한다. 그 예로 환경 정보를 센싱하는 센서로써는 온도 센서, 습도 센서, 압력 센서 등이 있고, 적외선 혹은 RF 를 사용하여 사물이나 사람의 위치를 파악하는 위치 센서도 있다.

2.2 센서 모듈

센서 모듈은 그림 2 에서 보시다시피 센서 계층에 주가 된다. 이는 센서, 서브 에이전트, 그리고 MIB 으로 구성되어 있다. 이는 하드웨어 즉 센서를 제조하는 벤더를 위한 것이다. 왜냐 하면 센서를 만드는 벤더들은 그 센서가 사용될 만한 여러 플랫폼들을 고려해 통신에 관련된 API 를 제작해야 한다. 이에 동반하여 미들웨어 속에 있는 통신 컴포넌트도 센서의 API 에 독립적이지 못하다. 하지만 센서 모듈을 사용하면 이 문제를 해결할 수 있다. 벤더가 센서를 만들 때 이 모듈을 삽입하여 제작하면 된다. 서브 에이전트와 마스터 에이전트는 표준화된 통신 프로토콜 AgentX 를 사용함으로써 사용자의 조작 없이도 센서와 미들웨어간의 통신이 이루어진다. 결과적으로 센서 제조사와 미들웨어의 개발자들의 수고를 덜어주는 효과를 얻을 수 있다.

3. 시스템 구동 메커니즘

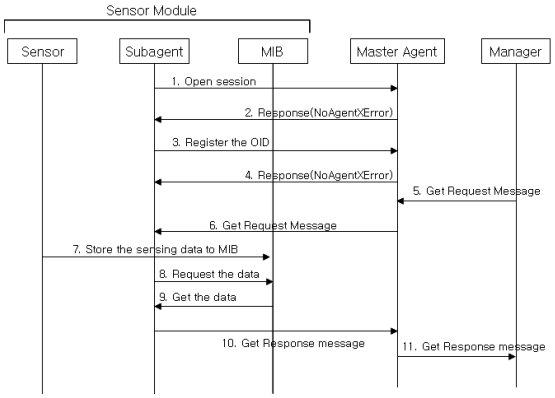

그림 2 세션 연결과 데이터 얻기

그림 3 은 센서 모듈이 마스터 에이전트에 연결되는 과정과 매니저에 의해서 요청된 데이터를 얻는 과정을 보여주고 있다. 서브 에이전트는 마스터 에이전트에게 세션 연결을 요구하게 되고 마스터 에이전트가 받아 드리면 서브 에이전트에 연결되어 있는 MIB 의 OID 를 마스터 에이전트에 등록한다. 이렇게 등록된 MIB 은 마스터 에이전트가 가지고 있는 MIB 리전(Region)의 한 요소가 된다. 응용 프로그램에서 요청받은 데이터에 대해서 매니저는 마스터 에이전트에게 Get Request 메시지를 보낸다. 마스터 에이전트는 해당 데이터가 있는 MIB 를 검색하여 해당 서브 에이전트에게 Get Request 메시지를 보낸다. Get Request 메시지를 받은 서브 에이전트는 자신에 연결된 MIB 에서 데

이터를 찾아서 마스터 에이전트에게 Get Response 메시지를 통해 데이터를 전달하고 마스터 에이전트도 또한 같은 방법으로 매니저에게 보낸다.

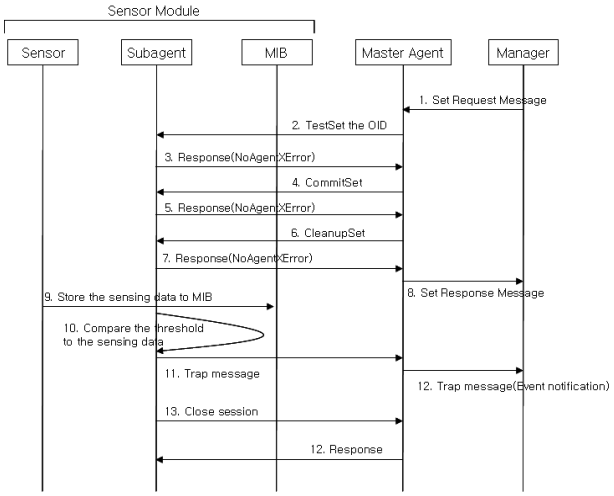

그림 3 Set 메시지를 이용하여 데이터 수정, Trap 메시지를 이용한 이벤트 통보, 그리고 세션 종료

그림 4 은 매니저에서 데이터를 수정하고 그 수정된 값에 대해서 이벤트가 발생하면 매니저에게 이벤트 내용을 통보하는 과정과 센서 모듈의 센싱 능력이 저하되어 제거할 경우에 서브 에이전트와 마스터 에이전트 사이에 연결되어 있는 세션을 종료하는 과정을 볼 수 있다. 매니저에서 특정 MIB 에 대한 값을 수정을 할 때의 과정은 3 단계로 이루어진다. 우선 해당 MIB 이 존재하는지를 알아보기 위해서 마스터 에이전트가 서브 에이전트에게 Test 메시지(TestSet PDU)를 보낸다. 서브 에이전트는 찾고자 하는 MIB 의 존재를 알려 준다. 그리고 마스터 에이전트는 값을 CommitSet 메시지를 통해 보내고 응답을 받는다. 마지막으로 마스터 에이전트는 값에 대한 변경을 승인하는 과정(CleanupSet)를 거치며 원하는 값을 설정한다.

이렇게 설정된 값은 서브 에이전트에서 이벤트를 발생시킬 때의 조건 값이 된다. 매니저가 설정한 값을 기준하여 센싱된 값과 비교한다. 그 결과 조건식에 부합될 때 이벤트를 발생하다. 이는 Trap 메시지를 통해 매니저에게 전달된다.

마지막으로 수명이 다한 센서는 서브 에이전트에서 마스터 에이전트에게 세션을 종료할 것을 요청하고 마스터 에이전트에서 응답을 하면 세션이 종료가 된다.

4. 결론

센서의 동적인 변화에 의해 MIB 과 에이전트의 수정이 불가피하고 벤더들이 여러 플랫폼에 맞는 센서의 API 를 지원해야 하는 문제가 발생하였다. 이러한 문제점을 해결하기 위하여 AgentX 프로토콜 메커니즘을 적용하였다. 기존 SNMP 의 에이전트가 마스터/서브 구조를 취하여 마스터 에이전트와 서브 에이전트로 나누어진다. 센서는 서브 에이전트와 연결이 되고 서브 에이전트는 마스터 에이전트에 연결이 된다. 이처럼 서브 에이전트와 센서 그리고 MIB 를 센서 모듈로 만들어 센서에 삽입하면 센서의 수나 종류가 변하더라도 그 센서에 해당되는 에이전트와 MIB 은 변하지 않아도 되며, 표준화된 프로토콜을 사용함으로써 벤더들은 미들웨어와 통신하기 위하여 여러 API 를 제공하지 않아도 된다. 이처럼 에이전트에 마스터/서브 구조를 사용함으로써 매니저는 센서에 대한 Transparency 를 보장 받을 수 있으며, 망 관리 기법을 사용하였기 때문에 센서 관리에 있어서 보다 쉬운 접근이 가능하다.

참고문헌

[1] Mark Weiser, “Hot Topics: Ubiquitous Computing,” IEEE Computer, 1993

[2] Mark Weiser, “The Computer for the 21st Century,” Scientific American, Vol. 265, No. 3, pp.94~104, Sep. 1991

[3] Anind K. Dey, "Providing Architectural Support for Building context-Aware Application," Georgia Institute of Technology, Nov. 2000

[4] Giles J. Nelson, "Context-Aware and Location System," Ph.D Thesis, Clare College, University of Cambridge, Jan. 1998

[5] Manuel Roman, “An Application Framework for Active Space Application,” University of Illinois, 2003

[6] Oxygen project, http://oxygen.lcs.mit.edu

[7] William Stallings, *The Practical Guide to Network-Management Standards*, pp.73, ADDISON-WESLEY PUBLISHING COMPANY, Oct. 1993

[8] 정종태, 정국상, 최덕재, “센서와 미들웨어간의 통신을 위한 아키텍처 설계 및 구현,” 한국컴퓨터종합학술대회(KCC) 2005 논문집, Vol. 32, No. 1(A), pp.865~867, 2005. 7

[9] IETF, Network Working Group, RFC2741