특집 10

WPAN 기반의 센서네트워크 소프트웨어 플랫폼 기술동향

목 차

김태환 · 남영진 · 김희철
(대구대학교)

1. 서 론

무선 센서 네트워크(WSN, wireless sensor network)는 많은 수의 센서노드로 구성되며, 각 센서노드는 임베디드 소프트웨어가 탑재되어 구동되는 임베디드 시스템이며, 통신 측면에서는 MAC 및 라우팅(Routing) 등을 지원하는 네트워크 프로토콜 스택이 탑재된 통신노드로서의 역할을 갖는다. 현재, 무선 센서 네트워크는 환경 감시, 생태 조사, 교통 정보, 농업 생산, 건축물 관리, 생산물 유통 등 다양한 응용 분야에서 실용화 단계에 접어들고 있으며, 네트워크 규모도 점차 증가하는 추세에 있다. 이에 따라, 그 어플리케이션의 복잡성도 증가하고 있으며, 효율적인 관리와 유지보수에 대한 관심도 높아지고 있다.

무선 센서 네트워크는 기존 모바일 애드혹 네트워크(MANET, mobile ad-hoc network)와 구별되는 고유 특성으로, 자원 제약성, 열악한 구동환경, 센서 노드의 하드웨어 및 소프트웨어 이질성(Heterogeneity)을 지니고 있다. 이에 따라, 기존 임베디드 시스템 및 MANET에서 사용되는 풍부한 기술들을 직접 활용하는 것이 어렵다. 먼저, 센서노드 상에서 구동되는 운영체제(이하 센서 운영체제)의 경우, 자원의 제약으로 인하여 일반 임베디드 운영체제와는 구별되는 기능 및 성능 요구사항을 갖는다. 특히, 응용 프로그래밍의 용이성, 높은 가용성, 동적 재구성의 편의성, 에너지 및 컴퓨팅 자원의 효율성 등의 요구들을 만족시킬 수 있어야 한다. 센서 운영체제와 센서노드 어플리케이션(이하 센서 어플리케이션) 사이에 위치하는 센서 미들웨어는 위치인식, 시간동기화, 센서 데이터 가공 등과 같은 센서 네트워크에 특화된 서비스를 제공할 수 있어야 한다. 한편, 센서노드에 탑재되는 MAC 및 라우팅을 위한 네트워크 프로토콜 스택들은 제한된 자원 및 극히 낮은 통신 신뢰성 등의 제약 사항들을 효과적으로 극복하여 시스템 자원 및 무선 데이터 통신의 효율성을 극대화시킬 수 있어야 한다. 한편, 센서 운영체제, 미들웨어 및 네트워크 프로토콜 등과 같은 핵심 요소뿐만 아니라, 센서노드에 탑재되어 구동되는 소프트웨어의 오류수정 또는 개량 등을 위한 센서 노드 소프트웨어 갱신, 유지보수 및 네트워크 관리와 등에 있어서도

경량, 에너지 절감 등과 같은 센서 네트워크 환경의 고유 특성이 그 설계에 고려되어야 한다.

위와 같은 센서 네트워크의 기존 기술과 구별되는 고유한 특성에 따라, WSN을 위한 운영체제, 미들웨어, 네트워크 프로토콜, 네트워크 관리 등 전 기술 영역 걸쳐 다양한 연구개발이 매우 활발하게 진행되어 왔다. 이러한 연구개발들은 센서 네트워크의 실용화에 크게 기여를 했으며, 실제로 많은 결과들이 센서 네트워크 구축에 활용되고 있다. 이러한 결과로서, 현재 무선 센서 네트워크는 독립적인 분산 컴퓨팅의 분야로 발전하고 있으며, 사물의 인터넷화(Internet of Thing)를 통한 유비쿼터스 시대를 견인하는 핵심기술로 정착하고 있다. 이러한 기술의 진화로 인하여 센서 네트워크를 위한 안정적 기능과 성능을 제공하고 다양한 어플리케이션 개발 및 유지보수의 효율성을 제고할 수 있는 통합 플랫폼이 요구되고 있다.

센서 소프트웨어 플랫폼은 센서노드 상에서 구동하는 센서 운영체제와 미들웨어 등의 시스템 소프트웨어와 센서 어플리케이션 개발자를 위한 다양한 개발도구로 구성된다. 이 센서 운영체제와 미들웨어는 다양한 응용분야의 어플리케이션을 신속하게 개발할 수 있고, 이들을 효율적으로 구동시킬 수 있도록 프로그래밍의 용이성과 높은 성능을 제공할 수 있어야 한다. 또한 서로 다른 사양의 센서노드와 싱크노드로 구성된 다양한 네트워크의 하드웨어 상에서 구동할 수 있도록 높은 이식성을 지니고 있어야 한다. 한편, 해당 시스템 개발도구들은 어플리케이션의 테스트와 디버깅 효율을 높이고, 유지보수 및 관리의 효율성을 제공할 수 있어야 한다. 즉, 응용 프로그램 개발의 용이성, 에너지 및 컴퓨팅 지원의 효율성, 높은 가용성, 동적 재구성의 편의성, 효과적인 개발도구 등의 조건들을 구비할 때, 해당 센서 소프트웨어 플랫폼은 다양한 응용 분야의 센서 네트워크 구축에 활용할 수 있는 범용 플랫폼으로서 발전할 수 있다.

본고에서는 센서 소프트웨어 플랫폼의 필요성, 구성요소, 기술개발 동향을 살펴본다. 먼저, 센서 소프트웨어 플랫폼의 필요성을 이해하기 위한 센서 네트워크의 특징을 간략하게 소개한다. 이후, 센서 소프트웨어 플랫폼의 구성요소인 센서 운영체제, 센서 미들웨어, 시스템 개발 및 네트워크 관리 기술들의 특징을 살펴본 후, 국내외 대표적인 센서 소프트웨어 플랫폼의 사례를 간략하게 소개한다.

2. 센서 소프트웨어 플랫폼 구성 요소

본 절에서는 센서 네트워크의 특징, 센서 운영체제 기술 동향, 센서 미들웨어 기술 동향, 센서 네트워크 개발 및 관리도구의 기술동향을 살펴본다. 네트워크 프로토콜 이슈는 플랫폼의 중요한 항목이지만 본 고 이외에서 독립적으로 많이 다루어지고 있으므로 지면의 제한으로 본고에서는 다루지 않는다[1].

2.1 센서 운영체제

지난 수년간 국내외 대학 및 연구소에서 다양한 설계 모델을 채택한 센서 운영체제들이 많이 발표되고 있다. 이러한 센서 운영체제 설계공간(Design space)은 수행제어 모델, 소프트웨어 아키텍처 모델, 프로그래밍 모델 등 다양한 요소들을 포함한다. 수행제어 모델의 유형은 프로그램의 구조와 그 수행 방식에 따라 이벤트(Event0 모델과 스레드(Thread) 모델로 구분한다. 초기 센서 운영체제는 이벤트 모델을 기반으로 설계되었으며, 이후 기존 컴퓨터의 수행제어 방식으로 사용되는 방식인 스레드 모델을 기반으로 많은 센서 운영체제가 설계되고 있다. 이 수행제어 모델은 프로그래밍의 용이성, 에너지 및 자원의 효율성에 크게 영향을 준다. 소프트웨어 구조는 단선적(Monolithic) 구조와 모듈화(Modular)구조로 구분된다. 소프트웨어 아키텍처 모델은 운

영체제의 동적 재구성 능력, 에너지 효율적인 리프로그래밍(Re-programming), 응용 프로그램의 운영체제 독립적 개발 및 탑재, 컴퓨팅 모델의 확장 등에 영향을 미친다. 또한 프로그래밍 모델은 프로그래밍 용이성을 결정짓는 요소로서 유한상태머신(FSM, finite state machine) 기반 모델, 컴포넌트 기반모델, 가상머신 기반 모델, 전통적인 API 기반 모델로 구분할 수 있다. 아래에서는 기존 센서 운영체제를 수행제어 유형인 이벤트 모델과 스레드 모델로 구분하여 간략하게 살펴본다.

2.1.1 이벤트 모델 기반 센서 운영체제

이벤트 기반 모델은 센서 네트워크 연구 초기에 주로 채택되어 센서 운영체제 설계에 활용되었다. 초기 센서노드의 MCU 자원은 최근에 사용되는 MCU들과 비교하여 상대적으로 매우 작았다. 즉, TinyOS가 개발된 초기(2000년)에 사용된 센서노드의 MCU는 ATMEL 90LS8535 모델로서 4 MHz 클록, 8 Kbytes 프로그램 메모리, 512 bytes의 데이터 메모리를 지니고 있었다. 이런 상황에서의 센서 운영체제 설계 모델로 초경량 운영체제를 구현할 수 있는 장점을 지닌 이벤트 기반 모델이 최상의 선택일 수밖에 없었다. 그 대신, 프로그램 구조가 기존에 사용하는 C-프로그램 수행과 같은 친숙한 방식이 아닌 유한상태머신(Finite state machine)과 같은 형태로 매우 어려워지는 단점을 감수해야 했다. 이를 해결하기 위하여 새로운 가상머신을 도입하거나, 새로운 언어를 사용하는 경우도 있다. TinyOS의 Mate 가상머신, NesC 언어가 그 대표적인 사례이다.

이벤트 기반 모델에서 이벤트(Event)는 비동기적 센서 트리거(Trigger)와 통신 패킷 도착 등으로 인해 발생하는 외부 인터럽트 등을 나타낸다. 프로그램은 일종의 서비스 루틴으로 볼 수 있는 독립적인 이벤트 핸들러(Event Handler)의 집합으로 구성된다. 일반적으로 이러한 서비스 루틴은 일종의 컴포넌트이며, 그 이름은 컴포넌트, 모듈, 메시지 핸들러 등 센서 운영체제마다 서로 달리 부른다. 프로그램의 구조는 해당 프로그램을 구성하는 컴포넌트들을 서로 직접 연결(Wiring)하는 방식의 구조 또는 독립적으로 존재하지만 식별자를 사용하여 간접적(Indirect) 연결 구조를 갖는다. 예를 들면 TinyOS의 경우에는 운영체제와 응용 소프트웨어 구별 없이 전체적으로 직접 연결된 컴포넌트 조합으로 표현된다.

프로그램 수행과정을 살펴보면, 먼저 신규 이벤트가 발생할 경우에 그 이벤트는 이벤트 큐(Event Queue)에 입력되며, 스케줄러는 선입선출(FIFO) 등의 지정된 스케줄링(Scheduling) 정책에 의거하여 이벤트 큐로부터 이벤트를 하나 꺼내서 해당 이벤트 핸들러를 수행시킨 후 그 수행이 끝나면 위 과정을 반복하는 방식으로 수행된다. 일반적으로 초기 이벤트 모델은 메모리에 컨텍스트 하나만 유지하여 메모리 사용량을 절감하기위해 이벤트 핸들러가 수행 도중에 타 이벤트 핸들러를 수행시키기 위해 교체(Preemption)하지 않고 종료 시까지 수행하는 방식(Run-to-completion)을 사용하였다. 물론 이러한 초기 기본 이벤트 모델이 이벤트 수행 시 새로운 소프트웨어적 이벤트를 발생시키거나, 긴 수행시간을 갖는 핸들러가 다른 이벤트의 수행을 지연시켜 내부 버퍼 오버플로우를 초래하는 경우 등을 (Bounded Buffer Problem) 방지하기 위하여 스케줄링 구조 변경 등의 개선이 지속적으로 이루어지고 있다.

이벤트 모델을 채택한 센서 운영체제로는 대표적인 센서 운영체제인 TinyOS, 스웨덴 컴퓨터 공학 연구소에서 개발한 Contiki, 미국 UCLA의 SOS가 있다. 가장 최초의 센서 운영체제이자, 가장 널리 사용되는 TinyOS는 컴포넌트 기반의 프로그램 구조를 가지며, 제한된 메모리 공간의

효율적 이용 및 에너지 효율적인 자원관리 등의 특징을 갖는다[2]. Contiki는 코드 블록(Code Block)을 기반으로 하는 프로그램 구조를 도입하여, 응용 소프트웨어 또는 서비스를 하나의 코드 블록으로 구성하고 있다. 각 코드 블록은 하위 디바이스 레벨의 서비스 핸들러(Poll Handler)와 여러 개의 이벤트 핸들러로 구성된다. 이벤트는 비동기적 이벤트와 코드 블록 간 호출에 해당하는 동기적 이벤트로 구성된다. 한편, 이벤트 스케줄링에서 우선순위를 지원하지 않으나, 이벤트 도착 시에 신속한 처리를 위하여 하위의 폴링 플래그(Polling Flag)와 상위의 이벤트 큐 기반의 2 단계 스케줄링 계층 구조를 지니고 있다[3]. SOS는 이벤트 모델의 스케줄링 단점을 보완하기 위하여 우선순위(Priority) 기반의 다중 큐(Multi-level Queue)를 채택하고 있다. 이벤트 대신 메시지 개념을 사용하며, 메시지 서비스 루틴들의 집합인 모듈 기반의 소프트웨어 구조를 사용하여 에너지 절감에 효과적인 모듈 단위의 동적 프로그램 갱신(Re-programming)을 지원한다. SOS의 모듈은 컴파일러에서 지원하는 위치-독립(Position-Independence) 코드 생성을 전제로 하고 있기 때문에, AVR MCU의 경우, 데이터 메모리가 4 Kbytes로 그 크기가 제한되어 프로그램 작성에 제한을 받고, MSP430 MCU의 경우는 이를 지원하지 않아 SOS 운영체제 자체를 구동시킬 수 없는 단점을 지니고 있다[4].

2.1.2 스레드 모델 기반 센서 운영체제

기존 컴퓨터의 일반적 구동 환경인 스레드 모델은 이벤트 모델보다 프로그램의 용이성과 스케줄링의 유연성이 우수하며 복잡한 센서 응용 소프트웨어를 구현하는데 장점을 지니고 있다. 이러한 장점에도 불구하고 초기 MCU의 자원 제약으로 인해 스레드 모델은 이벤트 모델보다 상대적으로 늦게 센서 운영체제 설계에 적용되었다. 하지만, 8-Bit MCU의 자원의 크기가 8 MHz의 클록, 64 Kbytes ~ 128 Kbytes의 프로그램 메모리, 4 Kbytes ~ 12 Kbytes의 데이터 메모리 등으로 초기 8-Bit MCU 보다 비교적 커지면서 이 모델의 적용이 검토되기 시작했다. 가장 먼저 멀티 스레드 모델을 적용한 콜로라도 대학의 Mantis 센서 운영체제는 4 Kbytes의 데이터 메모리와 128 Kbytes의 프로그램 메모리를 갖는 8-Bit Atmega 128L MCU를 탑재한 센서노드 상에서 기본 12개의 스레드를 무리 없이 지원하며, 스레드 교체 시 발생하는 오버헤드(Context Switching Overhead)가 약 120 개의 기계 명령어(Machine Instruction) 수행에 해당하며 그 시간은 약 60 μsec 정도로 기본 스레드 수행 주기(Time Quantum) 10 msec의 1% 이하에 불과하여 그리 큰 오버헤드가 되지 않는다는 점을 검증해 보였다. 이후 카네기 멜론 대학의 경우 8개의 태스크를 지원하는 Nano-RK 센서 운영체제에선 약 45 μsec로 더 작은 값을 발표하였다. 이후, 국내외에서 스레드 모델 기반 센서 운영체제가 많이 개발되었으며, 전통적인 멀티 스레딩(Multi-threading)의 동적 스레드 생성에서 벗어나 여러 개의 정적 태스크를 적재하여 지원하는 멀티 태스킹(Multi-tasking)으로 발전하고 있다. 나아가, 여러 개의 독립 프로그램을 동시에 수행시키며, 각 프로그램 하위에 스레드 생성을 지원하는 완전한 멀티 프로그래밍(Multi- programming)으로 점차 발전되고 있다.

스레드 모델을 채택한 대표적인 센서 운영체제로는 Mantis, Nano-RK, SenWeaver를 들을 수 있다. Mantis는 앞에서 언급한 바와 같이 최초의 스레드 모델 기반 센서 운영체제로서 그 개발 동기가 8-Bit MCU 기반 센서노드에서도 스레드 모델의 경량화 구현이 가능하고 여러 개의 스레드 지원도 큰 오버헤드 없이 가능하다는 점을 입증하는데 있었다. 따라서 새로운 이론적 기법보다는 일반적인 멀티 스레딩 컴퓨팅의 원

리 구현에 초점을 두고 있으며, 스케줄링의 경우, 5개로 구성된 우선순위(Priority)를 기반으로 주기적으로 태스크 교체를 할 수 있는 시분할-선점형(Time-sliced Preemptive) 스케줄링 방식을 채택하였다[5]. Nano-RK는 카네기 멜론 대학에서 개발한 센서 운영체제로서 센싱 주기와 서비스 시간이 서로 다른 여러 개의 센서를 다루는 태스크(Multi-modal Sensing Task)의 경우에, 수작업으로 코딩을 하여 제어하는 것이 매우 어려우며, 실제로 가능하지 않은 경우도 있다는 관찰에 의거하여 고정적 우선순위(Fixed Priority)를 갖는 주기적 태스크(Periodic Task)를 채택한 경성 실시간(Hard Realtime) 운영체제이다[6]. SenWeaver는 대규모 센서 네트워크 환경에 초점을 둔 센서 운영체제이다. 기존 스레스 모델의 단일 코드 구조(Monolithic)의 문제점을 해결하기 위하여 운영체제를 비롯한 전체 노드 소프트웨어를 세만틱 모듈 구조(Semantic Modular Architecture)로 설계였으며, 이를 기반으로 에너지 효율성이 높은 모듈 단위 동적 소프트웨어 갱신(Reprogramming)을 지원한다. 또한, 이 모듈 구조를 기반으로, 독립적으로 개발한 여러 개의 응용 소프트웨어를 특별한 변경 없이 탑재시킨 후 동시에 구동시키는 기존 컴퓨터상의 멀티프로그래밍(Multi-programming) 모델을 지원하고 있다. 한편, SenWeaver 운영체제는 API 함수, 기능(Functionality), 모듈, 소프트웨어 단계의 계층적 재구성 기능을 제공하여 코드 크기를 조정할 수 있도록 하였으며, MCU 종류 및 하드웨어의 사양에 상관없이 운영체제에 대한 코드를 자동 생성할 수 있는 기법을 통해 다양한 센서노드에서 구동될 수 있는 장점을 지니고 있다[7].

2.2 센서노드 미들웨어

일반적으로, 미들웨어는 운영체제와 응용 프로그램 사이에 위치하여 응용 프로그램들이 공통적으로 활용할 수 있는 서비스를 제공하는 소프트웨어를 지칭한다. 센서 네트워크 환경에서는 성능 확보를 위해서 응용 프로그램이 디바이스를 직접 액세스할 필요로 인해 센서 운영체제의 구현에서 디바이스 드라이버와 커널 등의 계층적 구분이 매우 모호하고, 또한 시간동기화 등 미들웨어에 해당되는 서비스들이 대부분 운영체제 내에 구현되고 있으므로 센서 네트워크의 미들웨어의 기술을 정확하게 정의하는 것은 용이하지 않다. 하지만, 센서 네트워크의 운영체제, 네트워크 스택 등 시스템 소프트웨어와 어플리케이션에 속하지 않는 공통 서비스를 미들웨어(이후, 센서 미들웨어) 기술로 간주할 수 있다. 이러한 주요 기술들로서 시간 동기화, 위치인식, 센서데이터 처리 기술을 간략히 소개한다.

2.2.1 시간 동기화 기술

노드와 노드간의 상호 협력이 필요하며 그 협력 동작이 상호 지정된 시간에 일어나야 하는 경우, 글로벌 시간을 서로 유지해야 이러한 협력과 정보의 사용이 가능하다. 센서노드는 MCU 내의 타이머를 기반으로 한 로컬 시간을 지니고 있으며, 주어진 기본 시간(Base Time)을 서로 맞춰 놓으면 로컬 시간을 활용하여 글로벌 시간을 산정할 수 있다. 하지만, 단순히 초기 기본 시간을 맞춰 놓았다 할지라도 센서노드의 클록 오차 및 네트워크 환경적 요인 등의 이유 때문에 정확한 글로벌 시간을 유지하는 것은 매우 어렵다. 이러한 문제점을 해결하기 위하여 다양한 시간 동기화(Time Synchronization) 프로토콜이 제안되고 있다. Receiver-to-receiver 프로토콜로서 정확도는 높지 않지만 비교적 경량 구현이 가능한 RBS(Reference Broadcast Synchronization), pairwise sender-to-receiver 프로토콜로서 높은 정확도는 유지하지만 오버헤드가 큰 TPSN (Timing-sync Protocol for Sensor Networks), 타임스템프(Time-stamp)를 메시지에 부착, 보

정을 하는 방식을 사용하는 MNTP(Manifold Node Time synchronization Protocol) 등이 대표적인 시간 동기화 기법들이다. 최근, 기존 동기 기법의 신뢰성, 정확성, 보안에 대한 취약성을 분석하고 개선하는 방안을 제시하는 등 안전하고 견고한 프로토콜에 대한 연구는 계속되고 있다.

2.2.2 위치인식 기술

센서 네트워크에서 노드의 위치 정보는 센서 네트워크의 유용성을 보다 높일 수 있는 중요한 정보이다. 일반적으로 위치인식은 크게 3단계로 구성된다. 첫 번째는 거리 측정 등의 물리적 데이터를 수집하여 최단거리 알고리즘의 수행을 통하여 물리데이터를 가공하는 처리를 수행하는 데이터 수집 단계이며, 두 번째는 이 수집된 데이터를 기반으로 원시(Primitive) 위치 정보를 산출하는 단계, 세 번째는 정제화(Refinement)를 통해 위치를 보정하는 단계로 이루어진다. 대부분의 위치 인식 기법들 있어서 입력 데이터는 크게 거리 정보, 노드 연결성(Connectivity), 클러스터 정보 등으로 구성되며 초기 위치를 계산하는 단계와 정제화 단계에는 MDS(Estimation, Multidimensional Scaling), SDP(Semi-Definite Programming), MLE(Maximum Likelihood Estimator) 등 서로 다른 다양한 알고리즘들이 적용되고 있다. 센서 네트워크의 환경, 즉 통신 비용, 전력소모량, 컴퓨팅 요구량 등 여러 성능 결정 요소들을 고려하여 이러한 알고리즘에 대한 개선이 진행되고 있다.

2.2.3 센서 데이터 처리

센서 네트워크의 응용에서는 주로 모니터링, 추적 및 협업 기능이 대부분의 응용 로직을 구성하므로 현실적으로 통신량을 감소시킬 수 있는 기술이 요구된다. 즉, 이러한 통신 요구량을 줄이기 위한 데이터 처리 기술이 필수적으로 미들웨어 서비스로 구현되어야만 해당 응용이 가능하다. 이러한 데이터 처리를 위한 접근 방식은 크게 통신 횟수를 감소시키는 기법, 전체 통신량을 감소시키는 기법, 복잡한 계산을 수반하는 로컬 데이터 처리를 위한 효율적인 알고리즘 등으로 구분 할 수 있다. 첫 번째, 통신 횟수를 감소시키는 기법으로는 주로 다수의 자식 센서노드로부터 전송되는 동종 데이터를 처리하는 그룹화(Aggregation)를 들 수 있다. 두 번째, 전체 통신량을 감소시키는 기법으로는 에너지 효율적인 무선 전송을 위한 데이터 압축 기술을 포함한다. 마지막으로 복잡한 계산을 수반하는 로컬 처리의 효율적인 알고리즘으로는 멀티 센서로부터 관심 이벤트/상황 추출을 위한 데이터 융합(Data Fusion) 기술을 포함한다. 이러한 알고리즘들은 이미 기존 컴퓨터들을 위해 많은 연구개발이 이루어진 분야이지만, 이들을 센서 네트워크에 접목시키는 것은 센서 네트워크의 특성, 즉 네트워크 크기와 토폴로지의 동적변화, 에너지 및 자원제약 등을 충분히 고려해야 하므로 매우 복잡하고 어려운 작업이다.

2.3 센서 네트워크 개발환경 및 네트워크 관리

센서 어플리케이션은 센서노드에서 구동하는 프로그램으로 일종의 임베디드 S/W로 분류될 수 있다. 따라서 센서 어플리케이션의 개발에는 임베디드 SW개발에서 사용하는 것처럼 MCU 제조사 등에서 제공하는 컴파일러, 공용 프리웨어 및 간단한 퓨징(Fuzing) 도구, 에뮬레이터(In-Circuit Emulator) 등의 툴셋으로 구성된 소규모 크로스 개발환경을 사용하고 있다. 이 경우, 대부분의 개발자들은 각자 자신이 선택한 개별적인 도구를 이용해 프로그램 편집 및 컴파일, 이미지 적재 및 디버깅을 수행하는 방식을 사용한다.

센서 어플리케이션은 센서 네트워크상의 타 노드와의 통신을 수반하므로 일반적으로 단일시스템 상에서 동작하는 임베디드 소프트웨어보다

그 개발이 더 복잡하고 어렵다. 특히, 대규모 센서 네트워크상에서 동작하는 복잡한 로직을 갖는 센서 어플리케이션의 개발에 있어 위 소규모 툴셋 기반의 개발 방식은 불편함을 야기할 뿐만 아니라 개발 시간을 증가시킬 수 있으며, 궁극적으로 소프트웨어 개발의 생산성 저하를 초래한다. 이에 따라, 보다 높은 생산성을 제공하는 개발환경의 요구된다.

아직, 전 세계적으로 보편적인 센서 운영체제가 사용되고 있지 않은 상황에서 센서 네트워크 개발환경 개발의 특징은 해당 센서노드 소프트웨어 플랫폼에 높은 의존성을 갖는다는 점이이다. 예를 들면, 어플리케이션 에뮬레이션, 센서노드 소프트웨어 모듈의 커스토마이징 등의 기능을 지원하는 개발도구는 해당 센서 노드 소프트웨어 아키텍처와 센서 운영체제에 의존적일 수 밖에 없다. 효율적인 개발환경을 위해서는 센서 운영체제와 함께 해당 개발도구들을 함께 개발해야 하는 부담이 수반된다. 따라서 대표적인 센서 운영체제들은 이를 위한 다양한 도구들을 함께 제공하고 있다.

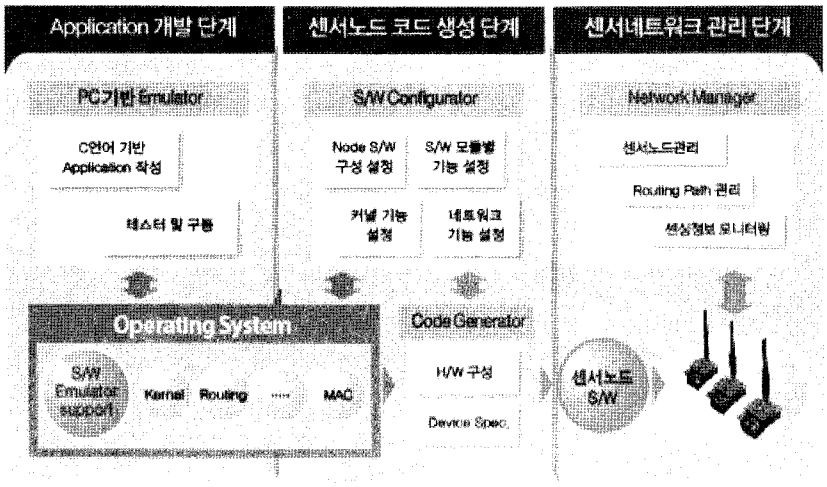
한편, 개발된 센서노드 어플리케이션을 각 센서노드에 탑재시켜 센서 네트워크를 구축, 설치한 후에는 소프트웨어 유지보수 및 에너지 모니터링과 같은 노드 레벨의 관리와 토폴로지 및 라우팅 방식 변경, 트래픽 모니터링, 장애 대처 등의 네트워크 레벨의 관리가 지속적으로 요구된다. 이러한 목적을 수행하기 위한 센서 네트워크 관리도구도 필요하다. 나아가, 센서 네트워크의 규모가 확대됨에 따라 기존 유선 네트워크 관리에서 필요한 네트워크 관리 시스템(NMS, network management system)과 같은 도구의 필요성이 증가하고 있다. 그러나 무선 센서 네트워크의 관리에 대한 이론적 체계가 정립되지 않아 현재는 업체별, 또는 연구기관별로 자체 관리 도구를 개발, 시험 테스트에 활용하고 있다. 특히, 이러한 관리 도구는 높은 효율성을 위하여 그래픽 기반의 기능 지원이 요구된다. 이러한 요구에 준해서 기존 센서 네트워크 운영체제를 개발하는 대학 또는 연구기관에서는 네트워크 관리 프레임워크와 이를 구현한 도구의 개발이 매우 활발하게 진행되고 있다.

3. 센서 소프트웨어 플랫폼 사례

본 절에서는 이러한 센서노드 소프트웨어 아키텍처, 센서 운영체제, 각종 개발도구 등의 제반 구성 요소를 구비한 SenWeaver 센서 소프트웨어 플랫폼을 살펴봄으로써 향후 범용 센서 소프트웨어 플랫폼 필요성 및 그 발전 전망을 조망해 본다.

SenWeaver 플랫폼은 대구대학교 유비쿼터스 신기술연구센터(UTRC)에서 개발한 범용 센서 소프트웨어 플랫폼이다. SenWever 플랫폼은 대규모 센서 네트워크를 위한 성능 요구인 프로그래밍의 용이성(Programmability), 다양한 하드웨어에의 가용성(Retargetability), 동적 재구성의 용이성(Dynamic Reconfigurability), 에너지와 컴퓨팅 자원의 효율성(Energy & Resource Efficiency)을 만족시키기 위한 목적으로 개발되었다. 이를 위해 모듈 단위 동적 갱신 등의 유연한 지원이 가능한 계층적 소프트웨어 아키텍처를 설계하고, 이를 기반으로 하는 멀티프로그래밍 모델을 지원하는 센서 운영체제 등을 포함하고 있다.

효과적인 센서 어플리케이션의 개발, 배치 및 유지보수를 위해서 SenWeaver 플랫폼은 (그림 1)과 같이 센서 응용 소프트웨어 개발 단계부터 센서 응용 소프트웨어와 센서 운영체제 커널의 컴포넌트 조합을 위한 센서노드 실행코드 코드 생성 및 실행이미지 적재 단계, 네트워크 관리 단계에 이르기까지 전 주기를 걸친 개발환경을 제공하고 있다. 즉, SenWeaver 플랫폼은 프로그래밍 개발 단계에서 어플리케이션의 구동을 사전 검증하기 위한 PC 기반 에뮬레이터, 센서노



(그림 1) 센서 소프트웨어 플랫폼 구성도

드 코드 생성단계에서 주어진 응용 소프트웨어의 성격과 하드웨어 자원의 크기에 최적화된 코드 생성을 위한 소프트웨어 설정 기능을 제공하는 S/W Configurator, MCU 종류나 하드웨어의 스펙에 상관없이 포팅에 필요한 해당 코드를 수작업 없이 자동으로 생성시킬 수 있는 기능을 통합 제공하는 Automatic Code Generator, 네트워크 관리 단계에서 사용하는 그래픽 기반 Network Manager를 제공하고 있다. 아래에서 그 특징을 간략하게 살펴본다.

3.1 SenWeaver Configurator

Configurator는 주어진 응용 소프트웨어의 성격과 하드웨어 자원의 크기에 최적화된 코드 생성을 위한 노드 소프트웨어 설정을 위한 도구이다. SenWeaver 플랫폼에는 운영체제, 미들웨어, 네트워크 스택의 세부 구성요소들을 비롯하여 노드에 탑재되는 응용 소프트웨어 모두가 일종의 소프트웨어 컴포넌트인 모듈 구조로 추상화되어 있으며, 이 모듈 레벨에서 모든 모듈에 대하여 균일하게 모듈 코드 및 런타임 버퍼 크기를 조정할 수 있다. 모듈은 여러 개의 기능(Functionality)으로 구성되며, 각 기능은 함수들의 집합으로 구성되는 계층적 구조를 가지며 코드 크기는 기능 레벨뿐만 아니라 함수 레벨로 선택을 할 수 있어 초소형 코드부터 전체 모듈에 이르기까지의 코드 크기를 자유롭게 조절할 수 있는 특징을 갖는다. 한편, SenWeaver Configurator는 플랫폼에서 제공하는 모든 모듈들과 응용 소프트웨어들 중에서 센서노드에 탑재할 모듈들을 선택한 후 각 모듈의 실행코드 위치와 필요한 데이터 메모리의 위치를 배치하는 기능을 제공하여 필요한 모듈들만 센서노드에 탑재시킬 수 있다. 아울러 여러 개의 응용 소프트웨어를 독립적으로 개발하여 하나의 센서노드에서 동시에 실행시킬 수 있는 다중프로그램 환경을 지원할 수 있도록 설계되었다.

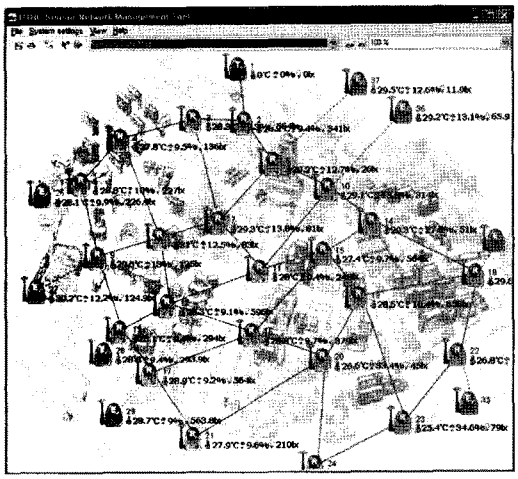
3.2 SenWeaver Automatic code generator

타 도구에서는 지원하지 않는 독창적인 기능으로 센서노드 하드웨어에 상관없이 코드를 자동으로 생성시키는 기능을 제공한다. 응용 개발자들은 MCU를 비롯한 탑재된 센서, 사용 I/O 포트 등의 측면에서 다양한 이질적인 사양을 갖는 각 센서노드에 대하여 SenWeaver 운영체제의 소스 코드를 변경하지 않고 자동으로 이식을

위한 해당 코드를 생성할 수 있다. 이를 위해서는 해당 센서노드를 구성하는 디바이스 컴포넌트, 컴포넌트간 연결 등의 하드웨어 사양을 담은 센서노드 플랫폼을 프로파일을 생성시키고, 앞에서 설명한 소프트웨어 모듈 프로파일을 등록하여, 필요한 모듈들을 선택한 후 코드 크기 조정을 위한 모듈별 설정을 수행하는 절차를 거치면 된다. 아울러 SenWeaver Configurator는 생성된 실행코드를 해당 센서노드에 퓨징하는 기능도 제공하며, 퓨징을 위한 타깃 노드에의 연결 방식으로는 Bootloader, JTAG, ISP 방식을 지원하고 있다.

3.3 SenWeaver Network Manager

SenWeaver Network Manager는 센서 네트워크 관리를 위한 도구로서 소규모 센서 네트워크 관리에 필요한 기능들을 주로 제공하고 있지만 향후 대규모 센서 네트워크를 위한 기능 확장 요구에 대비하여 오픈 아키텍처로 설계되었다. 또한 다양한 위치에서 원격 관리가 가능하도록 (그림 2)와 같이 웹기반 인터페이스를 제공하고 있다. 관리 기능은 노드 레벨과 네트워크 레벨의 계층 구조를 지니고 있는데, 노드 레벨의 기능으로는 노드 소프트웨어의 구성 관리 기능, 노드의 에너지 및 로그 조회 기능, 노드의 구동환경 변경에 따른 노드의 소프트웨어 모듈 교체 (Re-programming) 기능, 노드의 특정 소프트웨어 변경 기능 등을 제공하고 있다. 한편, 네트워크 레벨 관리 기능은 에너지 맵, 이웃 노드 (Neighbor node), 토폴로지 맵, 트래픽 표시 기능을 지니고 있다. 이러한 노드와 네트워크 관리 기능은 사용자 편의를 위해 그래픽으로 가시화하여 보여주며, 화면상에서 노드 또는 링크를 선택하여 해당 정보를 볼 수 있는 포인트 & 클릭 (Point-and-click) 방식을 채택하고 있다.



(그림 2) 관리자 시스템 구현 사례

4. 결 론

센서 소프트웨어 플랫폼은 센서노드 상에서 구동하는 센서 운영체제와 미들웨어 등의 시스템 소프트웨어와 센서 어플리케이션 개발자를 위한 다양한 개발도구로 구성된다. 이 센서 운영체제와 미들웨어는 다양한 응용분야의 어플리케이션을 신속하게 개발할 수 있고, 이들을 효율적으로 구동시킬 수 있도록 프로그래밍의 용이성과 높은 성능을 제공할 수 있어야 한다. 또한 서로 다른 사양의 센서노드, 싱크노드를 갖는 다양한 네트워크의 하드웨어 상에서 구동할 수 있도록 높은 이식성을 지니고 있어야 한다. 한편, 해당 시스템 개발도구들은 어플리케이션의 테스트와 디버깅 효율을 높이고, 센서 네트워크 유지보수 및 관리의 효율성을 제공할 수 있어야 한다. 이러한 조건들을 구비할 때, 이 센서 소프트웨어 플랫폼은 다양한 응용분야의 센서 네트워크 구축에 활용할 수 있는 범용 플랫폼으로서 발전할 수 있다. 무선 센서 네트워크의 기술이 지속적으로 발전하고 있는 가운데, 범용 센서 소프트웨어 플랫폼의 개발 및 보급은 다양한 무선 센서 네트워크의 응용 영역의 확장과 실용화의 첩경으로서 매우 중요하다.

참고문헌

[1] Yick, J. Mukherjee, and Ghosal, D., "Wireless sensor network survey", Computer Networks, Vol. 52, Issue. 12, pp. 2292-2330, Aug. 2008.

[2] E. Trumpler and R. Han, "A systematic framework for evolving TinyOS", in IEEE Workshop on Embedded Networked Sensors, pp. 61-65, May 2006.

[3] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors", Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, pp. 455-462, 2004.

[4] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes", Proceedings of the 3rd international conference on Mobile systems, applications, and services, pp. 163-176, 2005.

[5] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han, "MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms", ACM/Kluwer Mobile Networks & Applications (MONET), Vol. 10, No. 4, pp. 563-579, Aug. 2005.

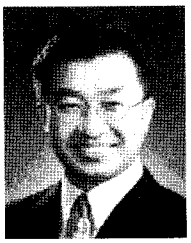
[6] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-RK: an Energy-aware Resource-centric RTOS for Sensor Networks", Proceedings of the 26th IEEE International Real-Time Systems Symposium, pp. 256-265, 2005.

[7] T.-H. Kim, H.-C. Kim, "Software Architecture for Highly Reconfigurable Sensor Operating Systems", Journal of IEMEK, Vol. 2, No. 4, pp. 242-250, Dec. 2007.

저자약력

김 태 환

2002년 대구대학교 전자공학과(학사)
2004년 대구대학교 정보통신공학과(석사)
2007년 대구대학교 정보통신공학과(박사)
2007년~현재 유비쿼터스 신기술 연구센터 선임연구원
관심분야 : 센서 소프트웨어 플랫폼, 센서 운영체제, 무선 센서네트워크, 임베디드 시스템
이 메 일 : tkim@utrc.re.kr



남 영 진

1992년 경북대학교 전자공학과(학사)
1994년 포항공과대학교 전자전기공학과(석사)
2004년 포항공과대학교 컴퓨터공학과(박사)
1994년~1998년 한국전자통신연구원 컴퓨터연구단
2004년~현재 대구대학교 컴퓨터IT공학부 조교수
관심분야 : 무선네트워크, 임베디드 S/W, 네트워크 스토리지
이 메 일 : yjnam@daegu.ac.kr

김 희 철

1983년 연세대학교 전자공학과(학사)
1990년 University of Southern California 컴퓨터과학(석사)
1995년 University of Southern California 컴퓨터과학(박사)
1983년~1988년 삼성전자 책임연구원
1997년~현재 대구대학교 정보통신공학부 교수
2006년~현재 유비쿼터스 신기술 연구센터 센터장
관심분야 : 센서 미들웨어, 운영체제, 무선 센서 네트워크
이 메 일 : hckim@daegu.ac.kr