센서 독립성을 지원하는 센서 노드 플랫폼 구조

은성배* · 소선섭**

1. 서 론

유비쿼터스 센서 네트워크(USN)는 물리 공간의 빛, 소리, 온도, 움직임 같은 물리적 데이터를 센서 노드가 감지하고 측정하여 무선 네트워크를 통해 사용자에게 전달하는 시스템이다. 센서노드의 핵심 요소로는 센서, MCU, RF 통신 모듈, 전력부를 들 수 있다. 대부분의 센서노드는 기본 개념면에서나 저전력 구현 면에서 1~2 개 정도의 센서와 RF 정도만을 갖는 단순한 구조를 갖는다.

문제는 센서의 종류가 매우 많고 특성이 서로 다르다는 것이다. 예를 들어, 온도 센서는, 측정값의 범위에 따라, 상온에서 동작하는 반도체 온도 센서, 수온을 재는 막대형 온도 센서, 1,000도 이상의 매우 뜨거운 온도를 재는 온도 센서 등으로 다양하다. 센서의 출력도 전압, 전류, 주파수 등으로 다양하며 증폭이 필요한 센서, ADC에 바로 붙일 수 있는 센서 등, 다양하다. 또한 센서는 운영환경 속에 장착되어야 하는데 예를 들어 수온을 측정하는 센서 노드는 센서 자체가 물속에 설치되어야 한다. 하지만 MCU, RF 통신모듈, 전력부는 방수케이스에 넣어 보호해야 한다. 따라서 센서를 센서노드에서 가능한 분리하여 설계, 구현하여야 한다.

센서 노드 개발에는 센서 HW 구현 및 FW 프로그래밍, MCU 프로그래밍 및 ADC 기능 구현, 무선 통신 프로그래밍, 저전력 기능 등이 구현되어야 한다. 현재, USN 응용 개발자는 센서노드에 요구되는 기능들을 직접, 통합, 구현하는데 이 점이 USN 개발을 어렵게 한다. 예를 들어, 대기 환경 모니터링 응용을 개발할 때, 응용 개발자는 센서와 MCU, RF 모듈등을 선정, HW를 조립한다. SW 개발은 Tiny-OS나 Nano-Q+ 등을 활용하여 개발하거나 운영체제 도움없이 FW를 직접 개발한다. 이때 Tiny-OS나 Nano-Q+ 등의 개발 플랫폼들을 참고할 수 있으나 개발자가 많은 부분을 수정해야 한다.

PC나 Linux[1]등에서는 응용 개발자가 PC 플랫폼 HW, 잘 개발된 운영체제, 디바이스 개발자가 제공하는 드라이버를 활용하여 목표 응용만을 개발하면 된다. 운영체제는 표준화된 API를 제공하며 응용 개발자는 이 API에 익숙한 상태이다. 디바이스가 변경되더라도 이 API는 변하지 않는다. 새로운 디바이스를 채택할 때에도 디바이스 드라이버를 디바이스 공급자가 제공하므로 응용이 변경될 필요가 없다. USN 개발에서도 응용 개

※ 교신저자(Corresponding Author) : 은성배, 주소 : 대전광역시 대덕구 오정동 133번지 (306-791), 전화 : 042)629-7928, E-mail : sbeun@hnu.kr
* 한남대학교 정보통신공학부 교수
** 공주대학교 컴퓨터공학부 교수
(E-mail : triples@kongju.ac.kr)

발자가 잘 개발된 센서노드 플랫폼과 운영체제, 그리고 디바이스 드라이버 위에서 응용을 개발한다면 효율을 극대화할 수 있을 것이다.

하지만 기존의 Tiny-OS[2-4], SOS[5], MANTIS [6], Nano-Q+[7] 등의 센서노드 운영체제들은 디바이스와 운영체제를 동적으로 연결하는 기능을 제공하지 못한다. 첫째로, 기존 운영체제의 HW 플랫폼들이 센서 연결을 위한 표준화된 인터페이스를 제공하지 못한다. 센서마다 사용전압이 다른데 이를 지원하지도 못하며 센서 인터페이스도 SPI나 I2C 등과 같은 표준 인터페이스를 제공하지 않는다. 둘째로, 응용 개발자에게 확정된 API를 제공하지 못한다. USN 응용은 센서의 활용이 매우 다양한데 기존 운영체제들은 센서를 추상화하지 못함으로써 센서마다 별도의 API가 필요하다[8]. 셋째로, 디바이스 개발자에게 디바이스 드라이버를 별도로 개발하고 접속할 수 있는 기능을 제공하지 못한다. 디바이스 개발자가 센서를 공급할 때 이를 위한 디바이스 드라이버를 미리 만들어서 제공하려 해도 기존 운영체제가 리눅스처럼 이를 접속할 수 있는 기능을 제공하지 못한다.

Tiny-OS를 예로 들면, 대기 환경 모니터링 응용개발자가 CO2 센서를 Tiny-OS 플랫폼 HW에 부착해야 하는데 그 플랫폼에서는 5V 전원을 공급하지 못하며 인터페이스도 적절하지 못하다. 두 번째는 Tiny-OS가 그 센서에 대한 API를 지원하지 않을 때 응용 개발자는 그 센서의 디바이스 드라이버를 직접 개발해야 한다. 세 번째는 Tiny-OS 운영체제와 응용프로그램이 확실히 분리되지 않음으로써 개발자가 Tiny-OS 내부를 알아야만 프로그램이 가능하다는 점이다. 이러한 문제들이 USN 응용개발을 어렵게 한다.

본 논문에서는 표준화된 센서노드 플랫폼 기반의 USN 응용개발 방법을 제시한다. 응용 개발자는 플랫폼이 제공하는 확정된 API 위에서 응용프로그램을 개발하면 된다. 또한 센서 디바이스 공급자는 표준화된 HW, SW 인터페이스에 맞게 디바이스 드라이버를 개발, 공급하면 된다. 이를 위하여 센서노드 플랫폼이 가져야 할 기본 적인 구조를 제시한다. HW가 갖추어야 할 기본 구조를 제시하고 센서 디바이스를 접근할 때 사용할 수 있는 표준화된 API를 제시하며 디바이스 드라이버가 OS와 접속되는 방안을 제시한다.

2장에서는 배경으로서 기존의 USN 개발방식을 제시하며 기존의 센서노드 OS의 기능을 분석한다. 또한, IEEE1451 표준의 한계를 지적한다. 3장에서 센서노드 플랫폼 기반 개발방법을 제시하고 플랫폼의 구조를 제시하며 기술적 해결방안을 제시한다. 4장에서는 디바이스를 동적으로 다운로딩하는 시스템 구조에 대하여 설명한다. 5장에서 결론을 기술한다.

2. 배 경

2.1 기존의 USN 개발 방법

기존의 USN 응용 개발 방법을 그림 1에서 도식한다. 그림에서 응용 개발자는 사용자의 응용 개발 요구를 받아서 센서와 MCU, RF 모듈 등을 선정, HW를 조립한다. SW 개발은 Tiny-OS [2-4]나 Nano-Q+[7] 등을 활용하여 개발하거나 운영체제 도움 없이 FW를 직접 개발한다. 이때 Tiny-OS나 Nano-Q+ 등의 개발 플랫폼들을 참고할 수 있으나 개발자가 많은 부분을 수정해야 한다.

그 이유는 크게 3가지이다. 첫째는 Tiny-OS나 Nano-Q+ 등의 기존 HW 플랫폼이 USN 응용 요구를 만족하지 못한다는 것이다. 환경 모니터링 응용들은 센서가 운영환경 내에 장착되어야 하는

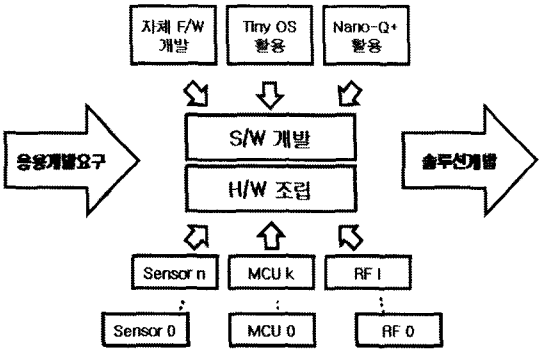


그림 1. 기존의 개발방법

데 기존 플랫폼들은 센서를 MCU 보드와 분리하기 어렵다. 예를 들어 수질 측정 센서의 경우 물속에 장착되어야 하고 플랫폼의 MCU 보드는 방수처리가 되어야 하는데 이런 점을 플랫폼에서 지원하지 못한다. 둘째는 Tiny-OS나 Nano-Q+ 등의 센서노드 운영체제가 응용프로그래머에게 다종다양한 센서들의 디바이스 드라이버를 제공하지 못한다는 점이다. 결국, 응용 개발자가 센서 디바이스 드라이버를 직접 작성해야 한다. 셋째로 센서 생산자가 기존 플랫폼에 장착할 수 있는 디바이스 및 드라이버를 제공한다면 응용 개발자의 개발 부담을 크게 덜 수 있으나 기존 플랫폼들은 표준화된 센서 HW 인터페이스를 제공하지도 못하며 또한 센서 생산자가 디바이스 드라이버를 만들 수 있는 구조도 제공하지 못한다.

이러한 개발 구조에서는 응용 개발자가 HW, SW, 디바이스 드라이버 등을 모두 제작해야 한다. 응용 개발자가 자체 보유한 플랫폼은 응용에 종속적이므로 수시로 제작되고 수정돼야 하므로 규모의 경제를 이루지 못한다. 개발 가격이 비싸지고 이 때문에 응용 개발 요구가 늘어나기 어렵다. 응용 개발이 많아질수록 개발 가격이 떨어지고 그것이 응용 개발 요구를 확대하는 선순환 구조를 이루기가 어렵다.

2.2 기존 센서노드 운영체제

2.2.1 Tiny-OS

기존 센서노드 운영체제로서 가장 먼저 개발된 Tiny-OS[2-4]는 이벤트 발생 중심의 상태 변화 방식을 채택한 센서 네트워크용 운영체제로써, 동시적인 프로세싱 및 제한된 하드웨어 메모리 공간에서의 효율적인 성능을 지원해주는 운영체제이다.

Tiny-OS의 센서 디바이스는 컴포넌트 기반 구조이고 이벤트 기반 멀티태스킹을 지원하며 NesC라는 프로그래밍 언어를 사용한 점이 특징이다. 그러나 HW 플랫폼이 센서를 장착할 수 있는 인터페이스나 전원 등을 제공하지 못하며 API도 센서 추상화를 제공하지 못한다. 센서 디바이스 드라이버를 접속하는 인터페이스도 제공하지 못한다.

2.2.2 SOS

SOS[5]는 메시지 패싱, 동적 메모리 할당, 모듈의 자율적인 적재와 제거를 지원하는 공동 커널(common kernel)을 지원하며, 동적 재구성이 특징이다. 이것은 새로운 모듈 업데이트가 필요할 때 무선 네트워크를 통하여 동적으로 변경할 수 있는 구조를 제공한다. 동적 업데이트가 센서 디바이스 드라이버 수준이 아니고 응용 프로그램 수준이며 센서 디바이스와 운영체제를 연결할 HW, SW 인터페이스를 제공하지 못한다.

2.2.3 MANTIS

MANTIS[6]는 레이어 기반의 운영체제이며, 하드웨어를 추상화시키는 디바이스 드라이버의 특징을 가진다. 그러나 이미 디바이스의 순서가 커널에 고정되어 있어 새로운 디바이스의 추가가 쉽지 않다는 단점이 있다. 또한, 마찬가지로 센서와 OS를 연결하기 위한 인터페이스를 제공하지

는 못한다.

2.2.4 Nano-Q+

Nano-Q+[7] 운영체제는 한국전자통신연구원(ETRI)에서 개발된 센서 네트워크를 위한 초소형 운영제제로써 다음과 같은 특징을 가진다. 에너지 소모를 최소화하고자 저전력 슬립모드를 제공하고, 제한된 메모리 사용을 최소화하도록 멀티스레드 간의 스택을 공유한다. C 기반의 프로그램을 지원하며 멀티 스레드 스케줄러 방식 등의 특징을 가진다. 마찬가지로 HW 플랫폼이 갖추어야 할 센서 인터페이스가 고려되지 않았고 센서를 위한 표준 APIfmf 제공하지 못한다. 또한 디바이스 드라이버가 OS와 연결될 인터페이스도 제공하지 않는다.

본 논문에서는 잘 정의된 API들과 통일된 디바이스 관리 기법을 제공함으로써 기존 센서노드 운영체제의 문제점을 해결하려 한다. 센서 디바이스 관리 기법은 많은 디바이스에 상관없이 프로그래머에게 통일된 API를 제공한다. 이러한 API를 이용하여 센서 디바이스 드라이버 제작자는 손쉽게 드라이버를 제작할 수 있다. 게다가 동적 디바이스 드라이버 추가방식을 채택하여 센서 디바이스 드라이버의 장탈착이 용이하다.

2.3 IEEE1451

IEEE1451[9]은 미국의 NIST[10]에서 주도하고 있는 표준으로서 1451.0~1451.6, 총 7개의 표준으로 나뉘어 진행 중이다. 이 표준은 트랜스듀서의 다양성을 극복하는데 초점을 맞추고 있다.

센서나 트랜스듀서(그림 2) 제조사들은 TIM (Transducer Interface Module)이라는 부분까지 표준에 맞춰 개발하고, 네트워크 연결 모듈 개발자들은 NCAP(Network Capable Application Processor)부분을 담당하면 표준에 정의한 인터페이스 및 트랜스듀서의 특성을 기술하는 TEDS (Transducer Electronic Data Sheet)에 의해 트랜스듀서의 투명성을 지원받는다. NCAP은 1451.0와 1451.1과 관련 있는데 1451.0[11]는 TIM에 연결된 물리적인 트랜스듀서들의 특성을 파악할 수 있는 방법을 제공하며 1451.1[12]은 네트워크를 통하여 트랜스듀서들을 발견하고 접근할 수 있는 객체지향적인 접근 표준을 제공한다.

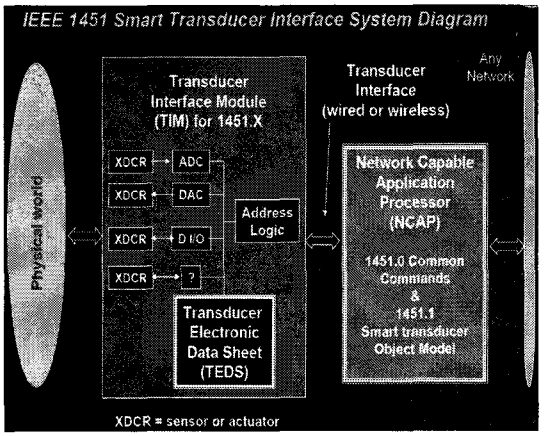


그림 2. IEEE 1451 스마트 트랜스듀서 구조

IEEE1451에서 정의하는 트랜스듀서와 NCAP과의 인터페이스는 센서노드 플랫폼을 개발할 때 많이 참고할 수 있으나 1451표준에서는 전원이 별도로 공급되고 저전력을 위한 제어도 어려우므로 1451 표준이 센서노드 플랫폼을 대체한다고 볼 수 없다.

3. 센서노드 플랫폼

본 논문에 이 장에서는 센서노드 플랫폼 기반의 개발 방법을 제시하고 플랫폼의 구조를 제시하며 HW, OS, 디바이스 인터페이스의 구조를 기술한다.

3.1 센서노드 플랫폼 기반 개발 방법

센서노드 플랫폼은 응용 개발자에게는 표준화된 API를 제공하며 센서디바이스 공급자에게는 표준화된 HW 인터페이스와 디바이스 드라이버 인터페이스를 제공한다. 이를 기반으로 그림 3에서 처럼 응용 개발자는 다양한 센서에 대한 처리를 고심하지 않고 익숙한 플랫폼과 API 위에서 응용을 개발할 수 있다. 센서 디바이스 공급자는 센서를 공급하면서 디바이스 드라이버도 함께 공급할 수 있다.

본 논문에 각 주체는 자신의 역할에 충실하면 되며 이를 통하여 선순환 효과를 기대할 수 있다. 플랫폼 공급자는 플랫폼의 성능을 높이기 위하여 최선을 다한다. 플랫폼이 응용에 독립적이므로 매출이 신장되며 이는 규모의 경제를 가능하게 한다. 따라서 플랫폼 가격이 낮아지면서 성능은 높아진다. 센서 공급자가 센서의 성능을 개선해서 공급하면 기존 응용에 동적으로 반영될 수 있으므로 센서의 개발이 촉진된다. 응용 개발자는 저가격의 플랫폼을 기반으로 센서 및 센서 드라이버와 익숙한 API 상에서 응용을 개발하므로 개발 비용을 낮출 수 있다. 이는 사용자로 하여금 더 많은 USN 응용 개발이 가능하게 하는 요소로서 USN 응용 개발 활성화를 위한 선순환 구조가 확립된다.

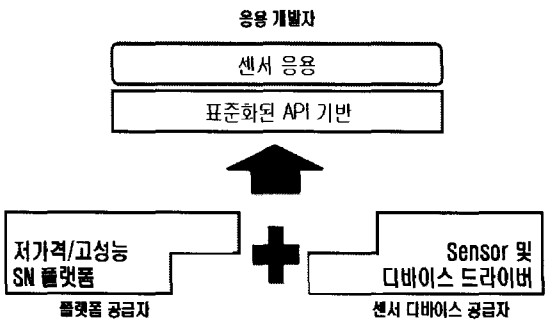


그림 3. 센서노드 플랫폼 기반 개발 방법

3.2 센서노드 플랫폼 구조도

센서노드 플랫폼 구조도를 그림 4에서 보여준다. 플랫폼은 크게 3계층이다. 제일 하위 계층은 HW 계층이며 2번째는 OS 이며 제일 위에 응용 프로그래머를 위한 API 계층이 존재한다. 각 계층에서 특히 고려되어야 할 부분이 색깔이 칠해진 4부분이다.

첫 번째로 API는 기존 운영체제의 API가 지원하지 못하는 센서 추상화를 지원하여야 하며 이것이 API에 반영되어야 한다. 둘째로 OS 계층에서 디바이스 드라이버 인터페이스가 지원되어야 한다. 센서 공급자가 디바이스 드라이버를 만들면 그 드라이버가 드라이버 인터페이스를 통하여 OS와 양방향으로 통신할 수 체계가 지원되어야 한다. 세 번째는 OS 계층의 디바이스 드라이버 자체인데 센서 공급자가 디바이스 드라이버를 작성할 수 있는 디바이스 드라이버 개발 체계를 제공해 주어야 한다. 끝으로 HW 계층인데 센서가 하드웨어적으로 연결될 수 있는 표준화된 인터페이스가 제공되어야 한다.

이들은 최종적으로 표준화되어야 하는데 USN 응용의 상용화가 막 시작하려 하는 2008년부터 추진하는 것이 필수적이다. 이 표준 플랫폼에 힘입어 USN 응용 개발이 규모의 경제를 실현하고 저가격으로 공급될 수 있다.

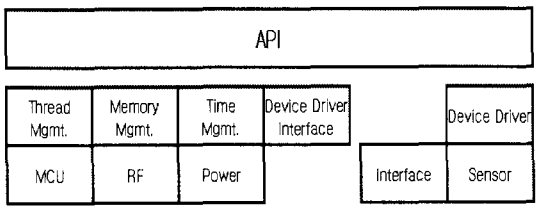


그림 4. 센서노드 플랫폼 구조도

3.3 표준화된 API

API는 프로세스 관련, RF 관련, 저전력 관련 등 다양하나 본 논문에서는 그 중에서 센서 디바이스를 관련 부분만을 제시한다.

3.3.1 센서 디바이스 종류

Linux에서 디바이스를 3종류로 분류한 것과 같이 본 논문에서는 센서, 이벤트 센서, 구동기의 3종류로 분류한다. 센서와 이벤트 센서는 입력 디바이스이며 구동기는 출력 디바이스이다. 센서는 read() 함수 호출시 블록없이 바로 값을 리턴한다는 점에서 이벤트 센서와 다르다. 이벤트 센서의 경우 사용자가 정의한 handler가 이벤트 발생시 호출되며 이때 read()함수를 호출하여 센서 값을 얻는다.

온도 센서나 조도 센서 등은 블록없이 센서값을 읽어 갈 수 있으므로 센서 디바이스에 해당되며 초음파 리시버는 동작이 시작된 후 특정 시간이 지나야 초음파를 수신하므로 이벤트 센서에 속한다. 초음파 발신기는 정해진 시간만큼 초음파를 발신하므로 구동기에 속한다.

3.3.2 디바이스 API

응용 프로그램이 사용하는 센서 디바이스 API는 모두 6가지이다.

1) int fd = open("device name")

디바이스 이름을 이용하여 open()함수를 호출하면 TDT에 등록된 이름과 비교, 일치하는 디바이스의 _open() 드라이버를호출하고 인덱스를 fd로 리턴한다. fd는 이후 API에서 디바이스 대신 사용된다. _open() 드라이버 함수는 보통 그 트랜스듀서의 전원을 켜는 역할을 수행한다.

2) void close(fd);

close() 함수가 호출되면 그 fd에 해당되는 _close() 드라이버가 호출된다. _close() 드라이버는 보통 그 트랜스듀서의 전원을 끄는 일을 한다.

3) int num = read(fd, struct sensor_type *, sizeof(struct sensor_type *));

read() 함수는 센서 디바이스로부터 데이터를 읽어오는 역할을 한다. 이때 데이터 사이즈는 디바이스 드라이버에서 제공한 구조체의 크기만큼을 읽는다. 실제 읽힌 데이터 크기를 num 값으로 리턴한다.

4) int num = write(fd, struct actuator_type *, sizeof(struct actuator_type *));

write() 함수는 구동기에 데이터를 보내는 역할을 수행한다. 이때 데이터 사이즈는 디바이스 드라이버에서 제공한 구조체의 크기만큼을 쓴다. 실제 써진 데이터 크기를 num 값으로 리턴한다.

5) int err = register_handler(fd, void * func_pt);

이벤트가 발생할 때 호출될 함수의 포인터를 등록한다. 오류가 있으면 그 결과를 리턴한다.

6) int err = ioctl(fd, int operation);

디바이스를 제어하는 명령어를 전달할 때 사용한다. 명령어의 종류는 디바이스에 따라 다르다. 예를 들어, 초음파 발신기의 경우, 발신 시작 및 발신 중지를 이 함수를 이용하여 구현할 수 있다. 디바이스 제공자는 자신의 매뉴얼에서 명령어와 동작을 정확히 기술해야 한다. 디바이스 API를 이용하여 응용프로그램을 작성하는 예를 Yang [8]에서 기술하였다.

3.4 디바이스 드라이버 인터페이스

그림 5는 트랜스듀서 디바이스 드라이버 인터페이스 구조를 보여준다. 그림의 자료구조는 운영체제 내에서 응용 프로그램의 API호출과 센서 디바이스 드라이버를 연결한다. _insert_TD() 함수는 디바이스 드라이버의 초기화 때 드라이버를 등록하는 함수이다. 그림에서 3개의 트랜스듀서

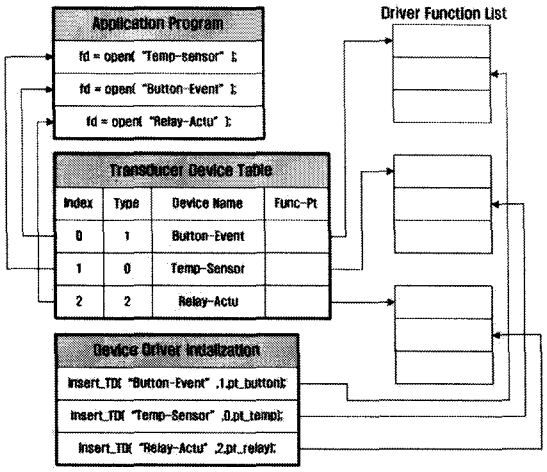


그림 5. 드라이버 인터페이스 자료구조

가 등록하는 것을 볼 수 있다. Transducer Device Table(TDT)은 각 디바이스의 이름과 디바이스 타입, 드라이버 함수 표의 포인터를 저장한다. 응용프로그램이 디바이스를 open()하면 TDT의 인덱스를 리턴하고 함수 호출 시에 이 인덱스를 활용한다.

3.5 HW 인터페이스

센서와 플랫폼과의 HW 인터페이스를 설계할 때 크게 3가지를 고려하여야 한다. 첫째는 센서모듈과 메인모듈을 분리하여 설계하여야 하는데 메인모듈이 방수, 방진되어야 하며 센서모듈은 운영환경내에서 동작되도록 개방되어야 한다. 둘째는 전원에 대한 고려인데 메인 모듈이 주로 3V 구동되는데 반하여 센서모듈은 3V, 5V, 9V, 24V 등으로 다양하다는 것이다. 게다가 센서가 상시 동작하지 않도록 MCU가 동적여부를 결정할 수 있어야 한다. 셋째는 통신방식인데 표준화된 인터페이스로서 SPI, I2C, 직렬통신(RS232, RS495)을 제시한다.

그림 6은 센서노드 플랫폼의 전원 관리의 예를

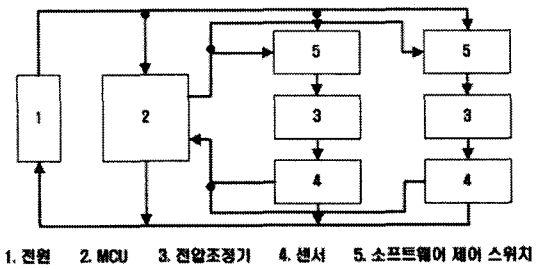


그림 6. 센서노드의 전원관리

보여준다. 그림에서 1과 2는 메인모듈에 속하며 오른 쪽의 4번 모듈 들이 센서모듈이다. 각 센서모듈은 메인 모듈로부터 전원을 공급받으며 센서요구 전압에 맞도록 3번의 전압조정기를 거친다. 저전력을 위하여 센서가 동작하지 않을 때에는 5번의 소프트웨어 제어 스위치에 의해 전원을 차단한다.

4. 센서 디바이스 관리체계

이 장에서는 센서노드 플랫폼 기반의 개발 방법의 부수 효과인 센서 디바이스 관리 체계를 기술한다. 센서 디바이스 드라이버가 OS와 분리되어 개발되고 동적으로 연결될 수 있음으로써 센서 디바이스 드라이버의 부분 다운로딩이 가능하다.

그림 7은 센서노드가 설치된 후 자동으로 센서 디바이스 드라이버가 서버로부터 다운로딩되는 절차를 보여준다. 1. 센서노드가 USN에 장착되면 요구되는 센서 정보가 2, 센서네트워크를 통하여 게이트웨이에 전달되고 3. 인터넷을 통하여 서버에 전달된다. 서버에서는 4. 적절한 드라이버를 선택하여, 5. 인터넷을 통하여 게이트웨이까지 전달하고 6. USN을 통하여 센서노드에 전달 설치된다.

하나의 센서노드가 여러 개의 센서 디바이스를 장착할 수 있으며 각 드라이버가 OS 및 응용과 분리되어 다운로딩 된다. 그림 7의 오른편 그림에서 드라이버와 OS가 분리되어 다운로딩된 것을

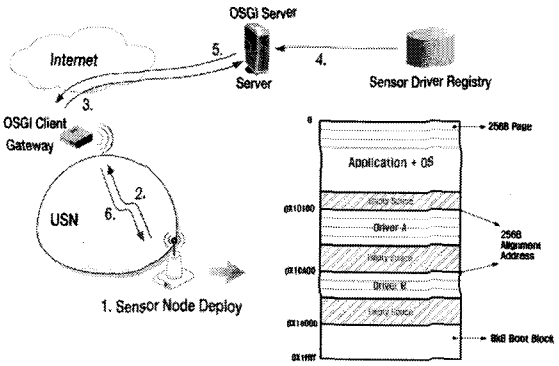


그림 7. 센서 드라이버 다운로딩 체계

보여준다.

원격 무선 업데이트 기능은 응용 프로그램의 요구에 따라 부트 로더가 실행되고 서버로부터 응용 프로그램을 다운로딩하여 어플리케이션 섹션을 업데이트 하는 것이다. 그러나 응용 프로그램 전체를 다운받는 것보다는 디바이스 드라이버 부분만을 다운받는 것이 효율적이다. 그러기 위해서는 디바이스 부분은 응용프로그램과 분리되어 플래시 메모리에 저장되어야 한다.

디바이스 드라이버가 등록되면 실제 구동함수의 시작 포인터가 디바이스 드라이버 인터페이스의 TDT에 저장되는데 구동함수의 시작포인터가 드라이버의 다운로딩된 실제 플래시 메모리 주소를 지정하면 된다.

예를 들어, 디바이스 드라이버 1은 10,A00h번지부터 시작되고 심볼테이블에서 xxx_open()은 10h번지부터 참조된다면 실제 디바이스 드라이버 1의 open()은 10,A10h번지가 된다.

5. 결 론

본 논문에서는 USN 개발을 활성화하기 위하여 PC 개발에서와 같은 기술 개발 분담 모델을 제시하였다. 이 모델의 각 주체가 자신의 역할에 충실할 때 규모의 경제가 실현되고 응용 개발 비용이 낮아지며 이를 통하여 응용 개발 요구가 늘어나는 선순환 효과를 기대할 수 있다.

이를 위하여 센서노드 플랫폼의 구조를 제시하고 기술 개발 요소를 명시하였다. 첫 번째로 API가 센서 추상화를 지원하여야 하며 이것이 API에 반영되어야 한다. 둘째로 OS 계층에서 디바이스 드라이버 인터페이스가 지원되어야 한다. 세 번째는 OS 공급자가 디바이스 드라이버를 작성할 수 있는 디바이스 드라이버 개발 체계가 제공되어야 한다. 끝으로 센서가 하드웨어적으로 연결될 수 있는 표준화된 인터페이스가 제공되어야 한다.

참 고 문 헌

[1] A. Rubini, *Linux Device Drivers*, O'Reilly & Associates, Inc., 1998.

[2] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer, and D. Culler, "The emergence of networking abstractions and techniques in tinyos," *Proc. of the First USENIX/ACM Symposium on Networked Systems Design and Implementation(NSDI 2004)*, 2004.

[3] T. Schmid, H. Dubois-Ferriere, and M. Vetterli, "Sensorscope: experiences with a wireless building monitoring sensor network," *Proc. of Workshop on Real-World Wireless Sensor Networks*, 2005

[4] P. Volgyesi and A. Ledeczi, "Component-based development of networked embedded applications," *Proc. of 28th Euromicro Conference*, 2002.

[5] C. C. Han, R. Kumar, R. Shea, E. Kohler, and M.B. Srivastava, "A dynamic operating system for sensor nodes," *Proc. of MobiSys*, pp. 163-176, 2005.

[6] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J.

Rose, A. Sheth, B. Shucker, J. Deng, and R. Han, "MANTIS: System Support For MultimodAl NeTworks of In-situ Sensors," *Proc. of 2nd ACM International Workshop on Wireless Sensor Networks and Applications*, pp. 50-59, 2003.

[7] S. Park, J. Kim, K. Lee, K. Shin, and D. Kim, "Embedded Sensor Networked Operating System," *Proc. of 9th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*, 2006.

[8] Manseok Yang, Sun Sup So, Steve Eun, Brian Kim, Jinchun Kim, "Sensos: A Sensor Node Operating System with a Device Management Scheme for Sensor Nodes," *International Conference on Information Technology (ITNG'07)*, pp. 134-139, 2007.

[9] Institute of Electrical and Electronics Engineers, Inc., "IEEE Standard for Smart Transducer Interface for Sensors and Actuators - Network Capable Application Processor (NCAP) Information Model," *Mixed-Mobile Communication Working Group of the Technical Committee on Sensor Technology TC-9 of the IEEE Instrumentation and Measurement Society*, June 1999.

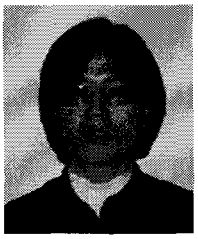
[10] http://ieee1451.nist.gov/.

[11] Draft Standard for a Smart Transducer interface for Sensors and Actuators: Common Functions, Communication Protocols, and Transducer Electronic Data Sheet(TEDS) Format, *TC-9 of the IEEE Instrumentation and Measurement Society*, Dec. 2005.

[12] T. Brooks and K. Lee, "IEEE1451 smart wireless machinery monitoring and naval vessels," *Proc. of 13th International Ship Control Systems Symposium*, Orlando Florida, Apr. 2003.

은 성 배

- 1985년 서울대학교 전산학과 학사
- 1987년 KAIST 전산학과 석사
- 1995년 KAIST 전산학과 박사
- 현재, 한남대학교 정보통신공학과 교수
- 관심분야 : 유비쿼터스 센서 네트워크, 임베디드 시스템



소 선 섭

- 1986년 이화여자대학교 전산학과 학사
- 1988년 KAIST 전산학과 석사
- 1998년 KAIST 전산학과 박사
- 현재, 공주대학교 컴퓨터공학부 부교수
- 관심분야 : 유비쿼터스 센서 네트워크, 소프트웨어 엔지니어링