용이한 USN 응용 개발을 위한 센서추상화 지원 센서노드 운영체제

(A Sensor Node Operating System Supporting Sensor Abstractions for Ease Development of USN Applications)

은 성 배 [†] 소 선 섭 [††] 김 병 호 [†††]
(Seongbae Eun) (Sun Sup So) (Byeong Ho Kim)

요 약 기존의 센서노드 운영체제들은 응용프로그램에 대한 센서추상화를 지원하지 못한다. 따라서 응용이 센서를 위한 하드웨어, 디바이스 드라이버 등을 직접 개발해야 하는 부담을 갖는다. 본 논문에서는 센서추상화를 지원하는 운영체제 구조를 제시한다. 제안된 운영체제는 추상화된 센서 HW 인터페이스 기반의 HAL을 제공하고 센서 접근을 위한 추상화된 API를 제공한다. 센서제작자는 HAL을 이용하여 센서 디바이스 드라이버를 작성한다. 응용프로그래머는 센서 API를 이용하여 응용을 작성한다. 이러한 개발방식은 응용프로그래머의 부담을 크게 줄여서 USN 응용 개발을 활성화할 수 있다.

본 논문에서는 첫째로, 센서장착을 용이하게 하는 표준화된 센서 HW 인터페이스를 정의하였다. 둘째로, 센서를 추상화한 센서접근 API를 제공하였다. 셋째로, 센서 디바이스 드라이버를 작성할 때 활용될 HAL 라이브러리를 정의하였다. 예제 응용 프로그램을 작성하여 본 논문에서 제안한 센서노드 운영체제가 센서 추상화를 성공적으로 지원하는 것을 보였다.

키워드 : 센서노드 운영체제, 센서추상화, USN 응용 개발, 센서 HAL, 센서 디바이스 드라이버, 센서 인터페이스

Abstract Conventional sensor node operating systems do not support sensor abstraction for sensor applications. So, application programmers have to take charge of developing the hardware and the device drivers for the applications by themselves. In this paper, we present an OS architecture to support sensor abstraction. The OS provide not only application programmers with API library to access sensor devices, but also sensor developers with HAL library to access sensor hardware. This can reduce the development burden of application programmers significantly.

In this paper, at first, we define the sensor HW interface to ease the attachment of sensors. Second, we describe the sensor access API for application programmers. Third, we define the HAL library for sensor device programmers to use. Finally, we show that the OS can support sensor abstraction by illustrating the sample programs.

Key words : Semsor Node OS, Sensor Abstraction, USN Application Development, Sensor HAL, Sensor Device Driver, Sensor Interface

· 이 논문은 2009학년도 한남대학교 학술연구조성비 지원을 일부 받아 연구되었음
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '센서투명성을 지원하는 센서노드 운영체제 구조'의 제목으로 발표된 논문을 확장한 것임

† 종신회원 : 한남대학교 정보통신공학과 교수
sbeun@hnu.kr
†† 비 회 원 : 공주대학교 컴퓨터공학부 교수
triples@kongju.ac.kr
††† 비 회 원 : 경성대학교 컴퓨터공학과 전임강사
bkim@ksu.ac.kr

논문접수 : 2008년 8월 25일
심사완료 : 2009년 6월 2일

정보과학회논문지: 시스템 및 이론 제36권 제5호(2009.10)

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 개념 면에서나 저전력 구현 면에서 1~2개 정도의 센서와 RF 정도만을 갖는 단순한 구조를 갖는다.

문제는 센서의 종류가 매우 많고 특성이 다양하다 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 수온을 측정하는 센서노드는 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 이점이 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 도움 없이 FW를 직접 개발한다. 이때 Tiny-OS나 Nano-Q+ 등의 운영체제들을 참고할 수 있으나 개발자가 많은 부분 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 개발자가 잘 개발된 센서노드 운영체제, 그리고 디바이스 드라이버 위에서 응용을 개발한다면 개발 효율을 극대화할 수 있을 것이다.

하지만 기존의 Tiny-OS[2-4], Nano-Q+[5] 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 못한다. 둘째로, 응용 개발자에게 확정된 API를 제공하지 못한다. USN 응용은 센서의 활용이 매우 다양한데 기존 운영체제들은 센서를 추상화하지 못함으로써 센서마다 별도의 API가 필요하다[6]. 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 운영체제의 센서관련 HAL 라이브러리가 제공되지 못하므로 개발할 수 없다. Sensos[6]와 Nano-Q+의 스마트센서관리시스템[7]에서는 센서추상화를 제시하고 Linux[1]와 유사한 구조의 센서접근 API를 제공하였다. 하지만 센서 인터페이스를 위한 HW 추상화가 지원되지 못하고 HAL라이브러리가 제공되지 않는다는 문제를 갖는다.

본 논문에서는 표준화된 센서노드 운영체제 기반의 USN 응용개발 방법을 제시한다. 응용 개발자는 운영체제가 제공하는 확정된 API 위에서 응용프로그램을 개발하면 된다. 또한 센서 디바이스 공급자는 표준화된 HW, SW 인터페이스에 맞게 디바이스 드라이버를 개발, 공급하면 된다. 본 논문에서는 이를 위하여 첫째로, 센서 장착을 용이하게 하는 표준화된 센서 HW 인터페이스를 기술한다. 둘째로, 센서를 추상화한 센서접근 API를 제공한다. 셋째로, 디바이스 드라이버를 개발할 때 요구되는 HAL 라이브러리를 정의한다. 예제 응용 프로그램을 작성하여 본 논문에서 제안한 센서노드 운영체제 구조가 성공적으로 센서추상화가 지원됨을 보인다.

2장에서는 배경으로서 기존의 USN 개발방식을 제시하며 기존의 센서노드 OS의 기능을 분석한다. 또한, IEEE1451 표준의 한계를 지적한다. 3장에서 센서노드 운영체제 기반 개발방법을 제시하고 운영체제의 구조를 제시하며 기술적 해결방안을 제시한다. 4장에서는 프로그래밍 예를 통하여 센서추상화가 잘 지원됨을 보인다. 5장에서 결론 및 향후 연구 방향을 기술한다.

2. 배 경

2.1 기존의 USN 개발 방법

기존의 USN 응용 개발 방법을 그림 1에서 도식하였다. 그림에서 응용 개발자는 사용자의 응용 개발 요구를 받아서 센서와 MCU, RF 모듈등을 선정, HW를 조립한다. SW 개발은 Tiny-OS[2-4]나 Nano-Q+[5] 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.

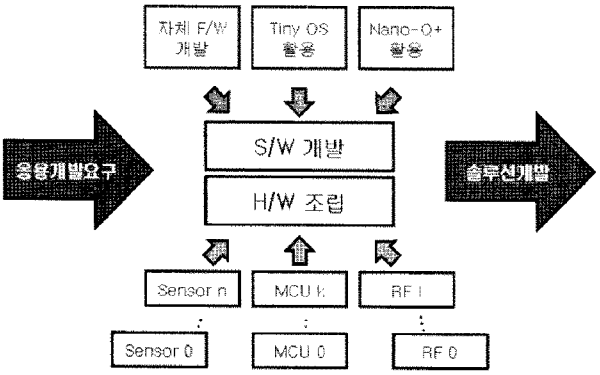

그림 1 기존의 개발방법

그 이유는 크게 3가지이다. 첫째는 Tiny-OS나 Nano-Q+ 등의 기존 HW 플랫폼이 USN 응용 요구를 만족하지 못한다는 것이다. 다양한 센서인터페이스와 전원이 지원되지 못한다. 둘째는 Tiny-OS나 Nano-Q+ 등의 센서노드 운영체제가 응용프로그래머에게 다양한 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 제공하지 못하므로 센서 생산자가 디바이스 드라이버를 만들 수 있는 구조도 제공하지 못한다.

2.2 기존 센서노드 운영체제

기존 센서노드 운영체제로서 가장 먼저 개발된 Tiny-OS[2-4]의 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. 또한, 센서 디바이스 드라이버를 접속하는 인터페이스도 제공하지 못한다.

SOS[8]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 응용프로그램 수준이며 센서 디바이스와 운영체제를 연결할 HW, SW 인터페이스를 제공하지 못한다.

MANTIS[9]는 레이어 기반의 운영체제이며, 하드웨어를 추상화시키는 디바이스 드라이버의 특징을 가진다. 그러나 디바이스의 순서가 커널에 고정되어 있어 새로운 디바이스의 추가가 쉽지 않다는 단점이 있다. 또한, 마찬가지로 센서와 OS를 연결하기 위한 인터페이스를 제공하지도 못한다.

Nano-Q+[5] 운영체제는 한국전자통신연구원(ETRI)에서 개발된 센서 네트워크를 위한 초소형 운영제제로써 C 기반의 프로그램을 지원하며 멀티 스레드 스케줄러 방식 등의 특징을 가진다. 그러나 HW 플랫폼이 갖추어야 할 센서 인터페이스가 고려되지 않았고 센서를 위한 표준 API도 제공하지 못한다. 또한 디바이스 드라이버가 OS와 연결될 인터페이스도 제공하지 않는다.

Sensos[6]와 Nano-Q+의 스마트센서관리시스템[7]에서는 센서 추상화를 제시하고 Linux[1]와 유사한 구조의 센서접근 API를 제공하였다. 하지만 센서 인터페이스를 위한 HW 추상화가 지원되지 못하고 HAL라이브러리가 제공되지 않는다는 문제를 갖는다.

본 논문에서는 추상화된 센서 접근 API들과 추상화된 센서 HW인터페이스를 정의하고 이를 기반으로 센서노드 OS 구조를 제시한다.

2.3 IEEE1451

IEEE1451[10]은 미국의 NIST[11]에서 주도하고 있는 표준으로서 1451.0~1451.6, 총 7개의 표준으로 나뉘어 진행 중이다. 이 표준은 트랜스듀서의 다양성을 극복하는데 초점을 맞추고 있다.

센서나 트랜스듀서 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 추상화를 지원받는다.

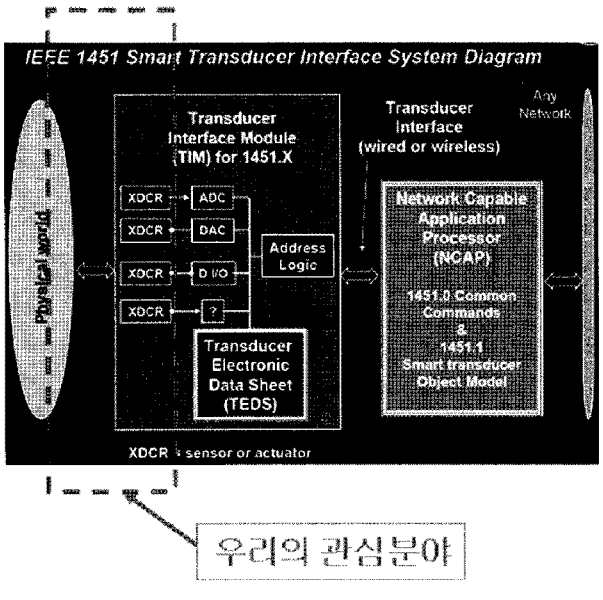

그림 2 IEEE 1451 스마트 트랜스듀서 구조

IEEE1451의 TIM은 센서노드 자체이며 NCAP은 싱크노드에 해당한다. 그림 2에서 볼 수 있는 것처럼 우리의 관심분야는 TIM내에서 물리적 센서와의 연결 부분이므로 IEEE1451은 관련 없다고 말할 수 있다.

3. 센서노드 운영체제 구조

3.1 센서노드 운영체제 기반 개발 방법

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

각 주체는 자신의 역할에 충실하며 그 결과, 선순환 효과가 기대된다. 플랫폼 공급자는 플랫폼의 성능을 높이기 위하여 최선을 다한다. 플랫폼이 응용에 독립적이므로 매출이 신장되며 이는 규모의 경제를 가능하게 한다. 따라서 플랫폼 성능이 높아지면서 가격은 낮아진다. 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.

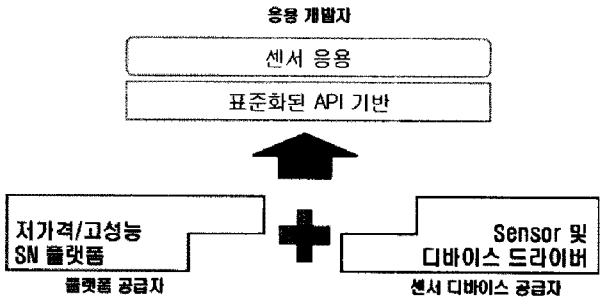

그림 3 센서노드 플랫폼 기반 개발 방법

3.2 전체 구조

센서노드 운영체제에서의 USN 개발을 그림 4에서 도식화하였다. 응용프로그래머는 센서접근 API를 기반으로 응용을 개발한다. 그림에서 API는 open(), close(), read(), write(), ioctl() 등이며 Linux와 비슷하다. 이때 센서 데이터 처리는 센서 제작자가 개발한 디바이스 드라이버가 담당한다. 센서 제작자는 표준화된 센서 HW

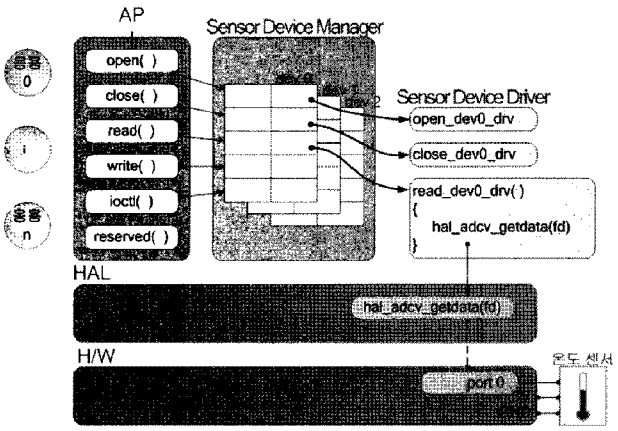

그림 4 전체 구조

인터페이스와 HAL 라이브러리를 기반으로 센서 디바이스 드라이버를 작성한다. 그림에서 드라이버는 HAL 라이브러리 hal_adcv_getdata(fd) 함수를 활용하여 플랫폼의 HW를 접근한다. HW를 직접 접근하지 않고 HAL 라이브러리를 통하여 접근하므로 센서추상화를 얻을 수 있다.

3.3 센서 인터페이스 추상화

본 논문에서는 센서를 크게 9개의 인터페이스로 추상화하였다. ADCV는 센서 출력이 전압이며 ADC가 요구된다는 것을 의미한다. ADCWV는 출력이 미약한 전압이므로 증폭 후에 ADC를 해야 한다는 의미이다. ADCA는 출력이 전류라는 것을 말하며 ADCWA는 전류가 미약해서 증폭 후 사용하라는 의미이다. ADC도 10비트ADC도 가능하고 14비트, 24비트 등, 다양하나 이는 인터페이스의 종류를 구분하는 것은 아니고 단지 인터페이스의 성능이 다르다는 것을 의미한다. 따라서 인터페이스 종류에 포함시키지는 않았다.

1. ADCV: ADC일반전압, 1선, ADC포트 연결
2. ADCWV: ADC 미약전압, 1선, 증폭 후 ADC 포트 연결
3. ADCA: ADC 일반전류, 1선, 저항 연결 후 ADC 포트 연결
4. ADCWA: ADC 미약전류, 1선, 증폭 후 저항 연결 ADC 포트 연결
5. FREQ: 주파수, 1선, 인터럽트 or 일반 포트에 연결
6. INTR: 인터럽트 방식, 1선, 인터럽트 포트 연결
7. I2C: 클럭과 데이터, 2선, 일반 포트에 연결 또는 하드웨어 I2C
8. SPI: 클럭, DI, DO, EN, 4선, 일반 포트 연결 또는 하드웨어 SPI에 연결
9. SERI: RS232, RS485 등, 3선 또는 2선, 하드웨어 직렬 있어야 함.

이와 별도로 전원을 추상화하였는데 V0(전원 없음),

V1(3V), V2(5V), V3(9V), V4(12V), V5(24V) 등이다.

3.4 표준화된 API

3.4.1 센서 디바이스 종류

Linux에서 디바이스를 3종류로 분류한 것과 같이 본 논문에서는 센서를 SENSOR형, EVENT형, ACTUATOR형의 3종류로 분류한다. SENSOR형과 EVENT형 센서는 입력 디바이스이며 ACTUATOR형 센서는 출력 디바이스이다. SENSOR형은 read() 함수 호출시 센서 값을 반환한다는 점에서 EVENT형 센서와 다르다. EVENT형 센서의 경우 사용자가 정의한 handler가 이벤트 발생 시 호출된다.

온도 센서나 조도 센서 등은 SENSOR형 디바이스에 해당되며 초음파 리시버는 동작이 시작된 후 특정 시간이 지나야 초음파를 수신하므로 EVENT형 센서에 속한다. 초음파 발신기는 정해진 시간만큼 초음파를 발신하므로 ACTUATOR형에 속한다.

3.4.2 디바이스 API

응용 프로그램이 사용하는 센서 디바이스 API는 모두 5가지이다.

1) int fd = open("device name")
   디바이스 이름을 이용하여 open()함수를 호출하면 TDT에 등록된 이름과 비교, 일치하는 디바이스의 _open() 드라이버를 호출하고 인덱스를 fd로 반환한다. 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 반환한다.
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 반환한다.
5) 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. 또한, 이벤트 센서의 핸들러를 ioctl()에서 지정한다.

3.5 트랜스듀서 디바이스 메니저

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

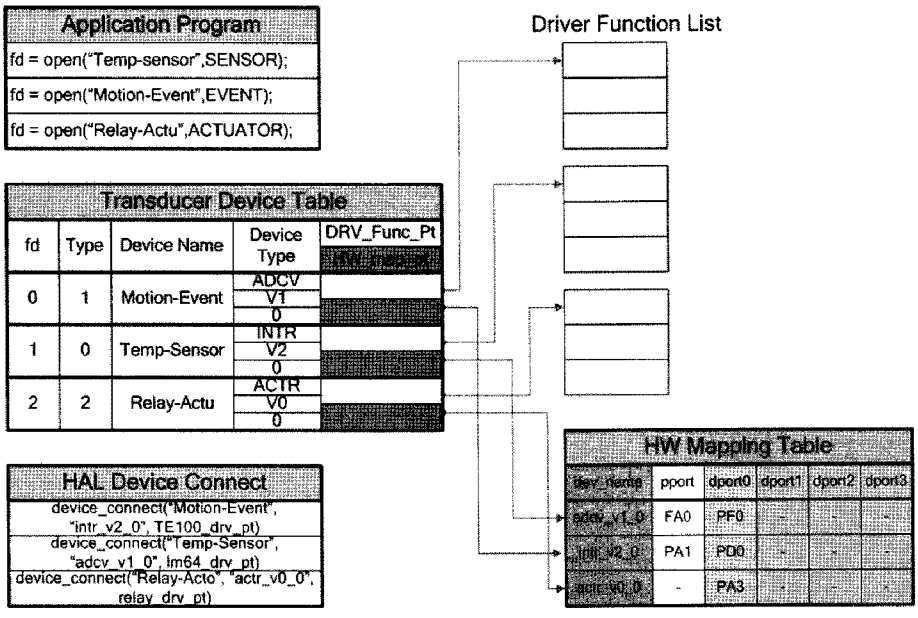

그림 5 트랜스듀서 관리 메니저의 자료구조

트랜스듀서가 등록되는 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.

HW mapping table은 실제 트랜스듀서 디바이스가 하드웨어의 어떤 포트에 연결됐는지를 지정한다. 이 tbale은 운영체제가 초기화 때 지정하며 device_connect() 함수를 호출할 때 논리적 센서와 연결된다.

3.6 인터럽트 처리 구조

EVENT형 센서가 인터럽트를 발생시킬 때 운영체제는 응용프로그램이 지정한 이벤트 핸들러를 수행하여야 한다. 본 논문에서는 이를 위하여 그림 6과 같은 인터럽트 처리 구조를 제시한다. EVENT형 센서에서 인터럽트가 발생하면 공통 인터럽트 ISR루틴이 먼저 호출된다. 이때 기본적인 인터럽트 불가 등이 수행되고 user_ISR_Table에 저장된 사용자의 핸들러가 호출된다. 그 후, 후처리 과정을 거쳐서 인터럽트를 종료한다.

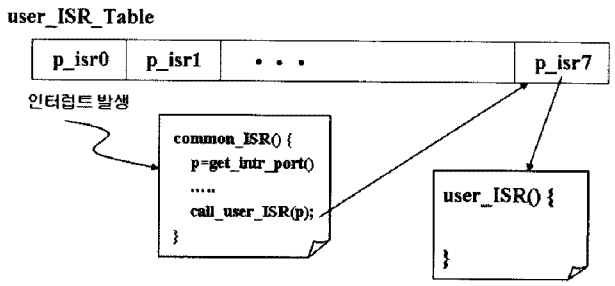

그림 6 인터럽트 처리 구조

4. 센서 추상화 평가

본 논문에서 제안한 운영체제가 센서추상화를 어떻게 지원하는지 보이기 위하여 예제 응용 프로그램을 제시하고 분석하였다.

4.1 프로그램 예제

그림 7은 예제 응용프로그램으로서 스마트 선풍기를 나타낸다. 스마트 선풍기는 온도 센서, 동작 감지 센서의 2개의 센서를 가지며 1개의 On-Off 릴레이를 갖는다. 온도 센서는 SENSOR 형이고, 동작감지 센서는 EVENT 형이다. On-Off 릴레이는 ACTUATOR 형이다.

동작은 온도가 특정 값 이상이고 동작이 있을 때 선풍기를 켜는 것이다. 온도가 특정 값 이하이거나 동작이 일정 시간동안 없을 때에는 선풍기를 끈다.

4.2 플랫폼 HW

그림 8은 스마트 선풍기가 하드웨어에 어떻게 연결됐는지를 보여준다. MCU는 ATmega128을 가정하고 있으며 어떤 포트가 온도 센서에 연결됐는지 전원 스위치

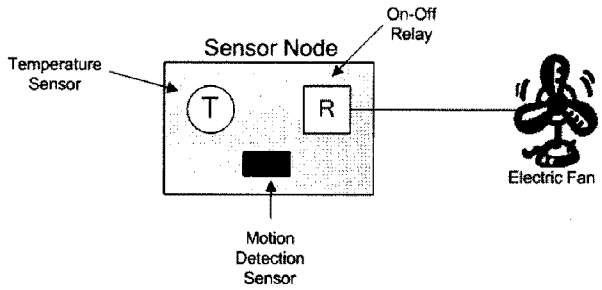

그림 7 스마트 선풍기

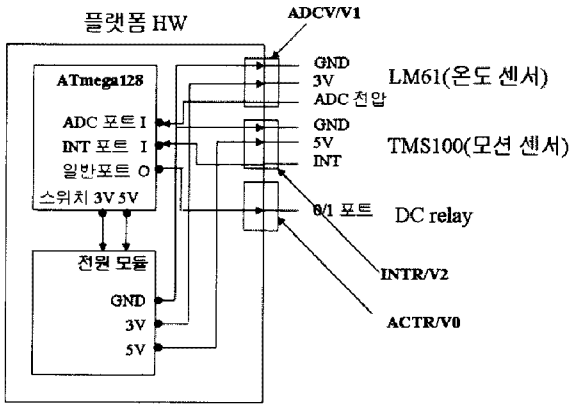

그림 8 플랫폼 HW

는 어떤 포트에 연결됐는지를 나타낸다. 그림에서처럼 온도센서는 ADCV 형의 인터페이스에 연결됐으며 전원은 3V라는 것을 알 수 있다. TMS모션센서는 INTR 형의 인터페이스에 5V 전원을 사용하는 것을 알 수 있다. DC 릴레이는 ACTR형이며 전원은 없다. 각각의 센서 인터페이스가 어떤 포트에 연결됐는지는 HW를 개발한 플랫폼 공급자가 알 수 있으므로 이를 OS가 초기화될 때 그림 5의 HW mapping table에 저장한다. 하지만 어떤 센서가 어떤 인터페이스에 저장되는지는 응용프로그래머만이 알 수 있으므로 "device_connect()"를 이용하여 응용 초기화 때 지정한다.

4.3 응용 프로그램

아래 프로그램은 응용 프로그램의 예를 보여준다. 각 센서는 Motion_Event, Temp_Sensor, Relay_Actu 라는 논리적 이름을 갖는다. 응용의 초기화 때 4.2절에서 지적한 데로 device_connect()함수를 이용, 어떤 센서가 실제 어떤 인터페이스에 연결됐는지를 지정한다. 이때 그 센서의 디바이스 드라이버도 함께 연결한다. 그 다음은 open()함수를 이용, 디바이스를 초기화한다. EVENT 센서의 경우, ioctl() 함수를 이용, motion_handler() 함수를 인터럽트 처리 루틴으로 등록한다. 그 후, read(), write() 문을 이용하여 응용 프로그램을 작성한다.

이 응용프로그램에서 볼 수 있듯이 사용자는 센서 처리를 위한 구체적인 정보 없이 API 수준에서 프로그램

을 작성하면 된다. 이는 센서추상화가 응용프로그램 단계에서 지원되는 것을 의미한다.

```
int fd1, fd2, fd3;
int MOTION = FALSE;
smart_electric_fan() {
    int temp;
/* hal device connect */
    device_connect("Motion_Event", "adcv_v1_0",
    lm64_drv);
    device_connect("Temp_Sensor", "intr_v2_0",
    te100_drv)
    device_connect("Relay_Actu", "actr_v1_0",
    relay_drv);
/* open transducer */
    fd1 = open("Temp_Sensor", SENSOR);
    fd2 = open("Motion_Event", EVENT);
    fd3 = open("Relay_Actu", ACTUATOR);
/* periodic_temp() is called after 1 sec. */
    ioctl(fd2, motion_handler);
/* infinite loop and do something */
    infinite_loop _and_do_something();
}
motion_handler() {
    MOTION = TRUE;
}
infinite_loop _and_do_something() {
    int temp;
    while(LOOP) {
        read(fd1, &temp, sizeof(temp));
        if(temp >= HLIMIT && MOTION)
            write(fd3, ON, 1);          // fan ON
        if(temp < LLIMIT || MOTION == FLASE)
            write(fd3, OFF, 1);
        sleep(SLEEP_PERIOD);
         MOTION = FLASE;
    }
}
```

4.4 디바이스 드라이버

아래 프로그램은 디바이스 드라이버 중, Temp_Sensor 관련 드라이버를 보여준다. SENSOR 형이므로 open(), close(), read()등이 작성되면 된다. 디바이스 드라이버는 센서 제작자가 작성하는 것인데 플랫폼 HW의 포트나 ADC 정보를 알 필요 없이 HAL 라이브러리만을 활용하여 작성할 수 있다. 이때 센서 인터페이스의 추상화에 따라 적절한 HAL 라이브러리를 호출한다. 이 프로그램에서는 hal_adcv_init(), hal_adcv_onoff(), hal_adcv_getdata() 등의 함수를 호출하였다.

센서 제작자가 신경 써서 작성해야 할 부분은 센서에 종속적인 내용인데 프로그램의 Temp_Sensor_Read()에서 볼 수 있는 것처럼 ADC된 값을 실제 온도로 변환하는 부분이 그 예이다. 이 프로그램에서 볼 수 있는 것처럼 드라이버 제작자는 HAL 라이브러리 함수를 이용, 센서 인터페이스의 구체적인 내용을 몰라도 드라이버를 작성할 수 있다. 이는 센서추상화가 센서 디바이스 드라이버 작성단계에서 지원된다는 것을 의미한다.

```
extern INT8 fd;            // fd는 OS가 제공함
INT8U Temp_Sensor_Open(void)
{
    hal_adcv_init(fd);
    hal_adcv_onoff(fd, ON);
    PutString("Temp Sensor Open\r\n");
    return fd;
}
void Temp_Sensor_Close(void)
{
    hal_adcv_onoff (fd, OFF);
    PutString("Temp Sensor Close\r\n");
    return 0;
}
INT16U Temp_Sensor_Read(Read_Data *pData,
INT8U size)
{
    INT16U t, temp;
    t = hal_adcv_getdata(fd);
    temp = (100 * t)/1024;  // 10비트 값을 실제
    온도로 변환
    memcpy(pData, &temp, size);
    return size;
}
```

4.5 HAL 라이브러리

아래 프로그램은 ADCV 형의 인터페이스 관련 HAL 라이브러리를 보여준다. hal_adcv_init()에서 get_HW_map() 함수를 활용, adcv 인터페이스가 실제 어떤 포트에 연결돼 있는지를 얻는다. 이 정보는 그림 5의 HW_mapping_table에 저장되어 있는 것으로서 fd를 가지고 접근할 수 있다. 기타 HAL 라이브러리들이 실제 포트를 읽고, 쓰고, 제어하는 기능들을 지원하고 있음을 알 수 있다.

이 HAL 라이브러리를 이용하여 디바이스 제작자가 센서 인터페이스의 구체적인 내용을 몰라도 디바이스 드라이버를 작성할 수 있다, 이는 추상화된 센서인터페이스가 센서 제작자에게 추상화를 제공한다는 것을 의미한다.

```
hal_adcv_init(int fd) {
    pmap = get_HW_map(fd);   // TDT의 HW 포트 정보 검색
    // normal voltage 형태로 ADC 초기화
    init_port(pmap, NORMAL VOLTAGE);
}
hal_adcv_onoff(int fd, int ON) {
    pmap = get_HW_map(fd);   // TDT의 HW 포트 정보 검색
    switch(ON) {
    case 1: port_set(pmap.pport, 1); break; // power switch on
    case 0: port_set(pmap.pport, 0); break; // power switch off
    default: return(ERROR_UNDEFINED_PARAM);
    }
    return(0);
}
hal_adcv_getdata(int fd) {
    pmap = get_HW_map(fd);    // TDT의 HW 포트 정보 검색
    select_channel(pmap.dport);
    // wait till ADC conversion completed
    while((ADCSRA & 0x10) != 0x10);

    adc_low_data = ADCL;   // read ADC low half
    adc_high_data = ADCH;  // read ADC high half
    return(adc_low_data);
}
hal_adcv_getstatus(int fd) {
    pmap = get_HW_map(fd);    // TDT의 HW 포트 정보 검색
    select_channel(pmap.dport);
    return(ADCSRA);           //return ADC status
}
```

5. 결론 및 향후 연구 방향

본 논문에서는 센서추상화를 지원하기 위한 센서노드 운영체제 구조를 제안하였다. 그 구조는 센서 HW 인터페이스를 추상화한 HAL을 제공하고 센서 접근을 위한 추상화된 API를 제공한다. 이를 바탕으로 센서제작자는 HAL을 이용하여 센서 디바이스 드라이버를 작성하고 응용프로그래머는 센서 API 를 이용하여 응용을 작성한다.

본 논문에서는 이를 위하여 첫째로, 센서장착을 용이하게 하는 표준화된 센서 HW 인터페이스를 정의하였다. 둘째로, 센서를 추상화한 센서접근 API를 제공하였다. 셋째로, 센서 디바이스 드라이버를 작성할 때 활용될 HAL 라이브러리를 정의하였다.

2개의 센서와 1개의 구동기를 갖는 스마트 선풍기의 구현 예를 제시하였다. 그것은 센서추상화 API를 활용한 응용프로그램, HAL 라이브러리를 활용한 디바이스 드라이버, 그리고 센서 HW 인터페이스를 추상화한 HAL 라이브러리로 구성 된다.각 단계의 프로그램들을 분석하여 본 논문에서 제안한 센서노드 운영체제가 센서 추상화를 성공적으로 지원하는 것을 보였다. 이러한 센서추상화 지원 OS 구조는 응용프로그래머의 개발 부담을 크게 줄여서 USN 응용 개발을 활성화시킬 것이다.

현재, 센서추상화 OS 구조를 간단한 응용 정도에 적용했을 뿐이며 다양한 응용에 적용하여 세부 사항을 보완해 나가야 한다. 또한, Nano-Q+ 기반으로 제안된 OS 구조를 구현할 것이다.

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 amd A. Ledeczi, "Component-based development of networked embedded applications," *Proc. of 28th Euromicro Conference*, 2002.

[5] 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.

[6] 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.

[7] Bumsuk Kim, Supsup So, Byeongho Kim, and Sengbae Eun, "A Smart Sensor Device Manage-

ment System in Nano-Q+," *Journal of KIISE: Computing Practices and Letters*, vol.14, no.1, Feb. 2008.
[8] 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.
[9] 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.
[10] 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.
[11] http://ieee1451.nist.gov/

은 성 배
1985년 서울대학교 전산학과(학사). 1987년 KAIST 전산학과(석사). 1995년 KAIST 전산학과(박사). 1995년~현재 한남대학교 정보통신공학과 교수, 관심분야는 실시간 시스템, 유비쿼터스 센서네트워크

소 선 섭
1986년 이화여자대학교 전산학과(학사) 1998년 KAIST 전산학과(석사). 2001년 KAIST 전산학과(박사). 1988년~1995년 국방과학연구소 연구원. 1995년~현재 공주대학교 컴퓨터공학부 교수. 관심분야는 소프트웨어 테스팅, 임베디드 소프트웨어, 센서네트워크

김 병 호
정보과학회논문지 : 시스템 및 이론
제 36 권 제 2 호 참조