상태머신 프레임워크를 지원하는 멀티쓰레드 기반 센서 네트워크 운영체제의 설계

이승근^O 허신
한양대학교 컴퓨터공학과
{leesk^O, shinheu}@cse.hanyang.ac.kr

Design of State machine frameworks based on Multi-Thread Sensor Network Operating System

Seungkeun Lee^O Shin Heu
Dept. of Computer Science and Engineering, Hanyang University

요 약

무선 센서 네트워크는 유비쿼터스 컴퓨팅에서 생활환경과 컴퓨터 사이의 중계자 역할을 하는 매우 중요한 연구 분야이다. 매우 제약적인 자원 환경에서 동작하여야 하는 센서 노드의 특성 때문에 제한된 자원을 효율적으로 관리할 수 있는 센서 노드용 운영체제가 요구된다. 또한 센서 네트워크는 외부 물리 환경의 변화에 반응하여 동작하는 시스템이기 때문에 여러 이벤트를 동시에 재빠르게 처리 할 수 있는 기능을 제공해야 하며, 센서네트워크 어플리케이션 프로그래머에게 이러한 반응형 어플리케이션 개발이 용의하도록 하는 프레임워크를 제공해야 한다. 이를 위해 본 논문에서는 반응형 시스템에 적합한 상태머신 프레임워크를 멀티쓰레드 기반의 Nano-Qplus 운영체제 상에서 센서 네트워크의 자원적 제약을 준수하면서 효율적으로 이벤트를 처리 할 수 있는 프레임워크를 지원하는 센서네트워크용 운영체제의 구조를 제안한다.

1. 서 론

무선 센서 네트워크는 군사 작전 지역, 산업 시설, 생태 환경 등에 배치되어 물리 환경으로부터 발생하는 데이터를 수집하고 이를 가공하여, 이 가공된 데이터를 센서 노드간의 무선 통신을 통하여 최종 사용자가 이 데이터를 활용할 수 있도록 해주는 기술이다[1][2]. 센서 네트워크를 구성하는 센서 노드는 극도로 제한된 자원을 가지고 있어 , 8bit MCU와 8~128KB 정도의 프로그램 가능한 메모리 , 512B~4KB 정도의 RAM으로 동작한다[3]. 이를 위해 센서네트워크용 운영체제는 운영체제 자체의 크기가 매우 작아야 하며, 사용하는 메모리 관리가 효율적이어야 한다. TinyOS[3], MANTIS[4], SOS[5]와 같은 범용 센서네트워크용 운영체제는 이러한 자원의 효율적인 사용을 위해 설계된 운영체제이다.

센서 네트워크 시스템은 외부 물리 환경의 이벤트에 반응하는 반응형 시스템이다. 또한 같은 이벤트에 대해서도 프로그램의 모드에 따라 입력 이벤트를 다르게 처리하는 경향이 있기 때문에 이러한 시스템을 설계하기 위해서는 상태 머신 기반의 컴퓨팅 모델을 이용하는 것이 접합하다. TinyOS[3]나 SOS[5]는 이러한 상태기반의 프로그래밍 환경을 제공하여 반응형 시스템의 설계를 좀더 용의하도록 하고 있다. 하지만 TinyOS와 SOS는 싱글 스택(single stack) 기반의 스택을 공유하는 시스템이기 때문에 , Run-To-Completion을 지키기 위해 많은 시간을 하나의 이벤트 처리에 사용하게 되고, 그에 따라 어플리케이션 개발에 있어서 이벤트 처리 코드를 최대한 짧게 유지해야 한다는 문제점이 있다.[4]

본 논문에서는 센서네트워크가 갖는 자원의 제약을 준수 하면서, 멀티 쓰레드 환경의 특징을 이용해 반응 시간이 빠른 상태기반 프레임워크를 설계하고 제안하고자 한다.

2. 관련 연구

2.1 센서네트워크 운영체제

대표적인 센서네트워크용 운영체제로 버클리 대학의 TinyOS[3]가 있다. 이는 제한된 자원에서 실행되기 위한 구조로 설계되었으며, 컴포넌트 기반의 구조를 가지고 있어서 계층적인 컴포넌트가 모여 하나의 시스템을 이룬다. 운영체제 이미지는 컴파일 할 시점에 사용되는 응용에 필요한 모듈만 적제 시킴으로서, 최소한의 크기로 운영체제 이미지를 생성 할 수 있다. 또한 이벤트 기반

(Event Driven) 실행 모델을 가지고 있어서, 외부 이벤트에 반응하고 그 이벤트가 계층적 모듈을 통해 상위 모듈로 전달되면서 시스템이 동작하도록 설계 되어있다.

각 모듈은 (그림1)과 같이 EVENT/COMMAND구조로 하위 모듈로부터 발생하는 이벤트를 상위 모듈에서 이벤트 핸들러를 이용하여 이벤트에 대한 처리를 하며 처리된 이벤트는 다시 상위 모듈로 전달하여 하드웨어에서 발생한 이벤트가 최상위 응용 프로그램 모듈로 전달되게 한다.

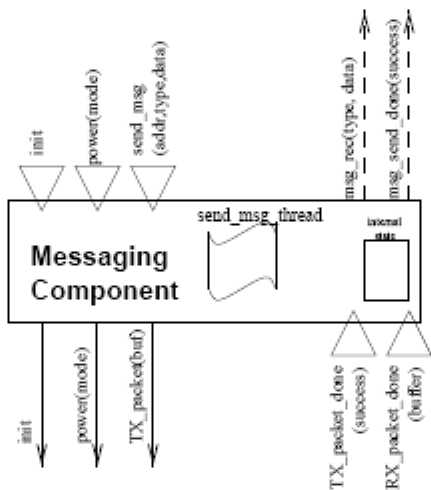

그림1 EVENT/COMMAND구조 모듈[1]

상위 모듈에서는 COMMAND를 하위 모듈로 전달하며 이는 다시 HW모듈까지 순차적으로 전달되어 시스템을 동작하게 한다.

이 이벤트 기반(Event Driven) 실행 모델은 각 모듈을 FSM에 대응하여 설계 할 수 있으며, 따라서 TinyOS를 상태머신기반의 운영체제라고 한다. 하지만 TinyOS는 시스템적으로 내부 상태를 기술할 방법은 명시하지 않고 있으며, 상태를 전이하는 방법도 특별히 제공하지 않는다. 따라서 모듈 개발 시 상태는 Frame영역에 내부 상태를 변수나 플래그로 애매하게 표현하게 된다. 그러므로 특정 시점에 이벤트 핸들러가 어떤 모드에 있는지 정확히 알기 어려워지고, 현재 모드를 검사하려면 복잡한 표현식을 검사해야한다. 또한 모드 간 전이를 수행하려면 많은 변수를 고쳐야 하므로 쉽게 일관성을 읽게 되고, 코드 곳곳에 산재해 있는 이와 같은 표현식은 불필요하게 복잡할 뿐 아니라, 런타임에서 검사하기에도 부담이 크다.[8]

TinyOS는 메모리 자원의 효율적 이용을 위해 싱글 스택(single stack)방식으로 모든 태스크가 하나의 스택을 공유하는 방법으로 구현되어 있다. 따라서 (그림2)와 같이 한 이벤트 핸들러가 종료되기 전까지는 다른 task는 수행 될 수 없으며, 이러한 문제점 때문에 모듈을 개발하는 입장에서는 모듈의 코드 크기를 최소한으로 작은 상태로 유지해야하고, 블로킹(blocking)을 유발 할 수 있는 함수에 대해 사용을 금지해야하는 추가적인 부담이 발생하게 된다. 또한 프로그램 로직의 오류로 인해 한 모듈이 무한 루프에 빠지게 될 경우 시스템 전체가 정지되는 단점이 있다[4].

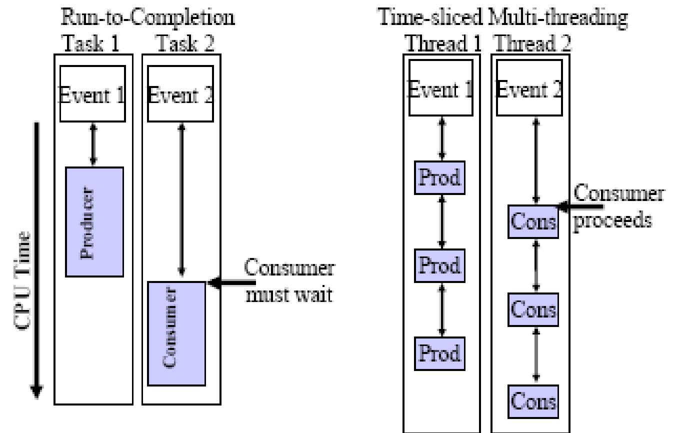

그림 2 TinyOS와 MANTIS의 실행 비교[4]

이러한 싱글 스택기반의 실행 구조의 단점을 극복하기 위해 콜로라도 대학에서는 MANTIS라는 멀티 쓰레드 기반의 센서네트워크용 운영체제를 개발하였다. 이는 전통적인 UNIX시스템과 비슷한 구조를 가지고 있으며, 이러한 구조 때문에 기존 UNIX에 적용되었던 응용 코드를 별다른 수정 없이 MANTIS에서도 사용 할 수 있는 장점을 갖고 있다.

2.2 Nano-Qplus 운영체제

Nano-Qplus는 한국전자통신연구원(ETRI)에서 개발한 센서 노드용 운영체제로써 다음과 같은 특징을 가진다.

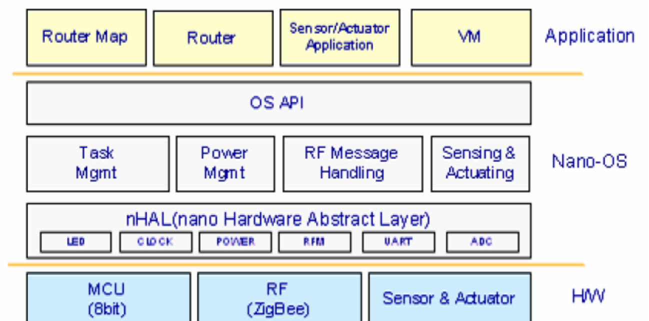

그림 3 Nano-Qplus 운영체제 구조[6]

- 에너지 소모를 최소화하기 위해 네트워크를 구성하는 노드 사이에 동기화된 시간을 기반으로 주기적으로 휴면과 활성화 모드를 반복하면서 메시지를 효율적으로 처리
- 제한된 메모리의 사용을 최소화하기 위해 스레드 사이의 스택 공유
- 멀티 쓰레드 스케줄러

(그림3)은 Nano-Qplus의 구조를 보여준다. Nano-Qplus는 기존의 유닉스 시스템과 유사한 계층적 구조를 가진다. 기존의 클래식한 운영체제 형태를 유지하면서 불필요한 모듈을 제거하고, 운영체제의 각 기능들을 경량화 하였다. 전체적인 운영체제의 형태는 기존의 운영체제와 유사하지만 센서 네트워크에 최적화하기 위하여 모듈화를 통한 재구성이 가능하다[6, 7].

3. 상태 기반 프레임워크 설계

상태 기반 프레임워크를 제공하기 위해서는 최소한 다음 요소가 필요하다.

첫 번째, ISR이나 네트워크 디바이스, 센서로부터 발생하는 이벤트를 메시지로 가공하며, 가공된 메시지를 구독하기 원하는 상태머신에 전달하는 이벤트 브로커 모듈이다. 이 모듈을 통해 상태 머신은 이벤트의 구독을 설정하고 데이터 수집 주기를 조절 할 수 있다.

두 번째, 전달된 메시지를 차례대로 FIFO구조의 큐로 관리하는 메시지 큐이다. 이 메시지 큐는 각 상태 머신마다 별도로 존재한다.

세 번째, 메시지를 메시지 큐로부터 읽어와 상태머신의 전이를 실행하고 상태머신의 문맥을 실행하는 실행모듈이다. 이 모듈은 Nano-Qplus의 쓰레드에 매핑 되며, 각 상태 머신마다 각자 실행 모듈과 별도의 큐를 가지고 있기 때문에, 상태 머신의 전의 과정에서 다른 상태머신이 선점하더라도, 선점된 상태머신의 Run-To-Completion은 위배되지 않는다.

3.1 이벤트 브로커 모듈 설계

이벤트 브로커 모듈은 ISR이나 네트워크 디바이스 , 센서 등 외부로부터 발생할 수 있는 이벤트를 관리하고 발생된 이벤트를 상태 머신에서 사용 할 수 있는 메시지 형태로 가공하여, 이 메시지의 구독을 원하는, 상태 머신에게 전달하는 역할을 한다. 본 논문에서 제안하는 메시지 브로커 모듈의 구조는 (그림 4)와 같다. 이 모듈을 이용해 상태머신은 자신이 받기를 원하는 메시지를 선택하거나 주기적으로 획득해야할 센서 데이터에 대해, 수집 주기와 데이터의 종류를 등록하게 된다. 이 모듈은 이러한 구독과 수집에 대한 정보를 유지하며 , 타이머 인터럽트에 의해 특정 주기로 센서로부터 데이터를 수집해 전달하거나 비동기적으로 발생하는 이벤트에 대해 즉각적으로 전달하여야 한다. 이를 위해 이벤트 브로커 모듈은 2가지 형태로 동작을 해야 한다.

첫 번째 주기적인 데이터 수집에 대해 상태머신에서 등록한 주기와 데이터 종류를 관리하여, 동일한 반복문 안에서 타이머 인터럽트에 의해 깨어나 저장된 주기와 현재 주기를 비교하여, 올바른 주기에 데이터를 센서로부터 수집/전달해야 한다. 또한 상태머신 별로 설정된 데이터 수집 주기를 정규화 하여 한번 수집한 데이터에 대하여 많은 상태머신에게 전달하여, 불필요하게 반복적인 데이터 수집을 피하도록 하여야 한다.

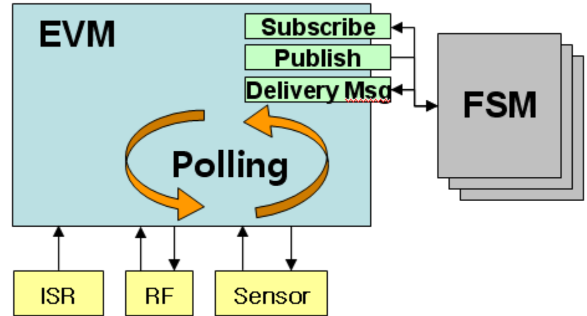

그림 4 이벤트 브로커 모듈

두 번째 비동기적으로 발생하는 이벤트에 대해서는 이벤트가 발생한 즉시, 이벤트를 메시지로 가공하여 구독을 원하는 상태 머신에게 전달해야 한다. 이를 위해 인터럽트 서비스 루틴 안에서 이벤트를 가공하고 전달하는 함수를 호출하도록 하고, 비동기적으로 발생하는 네트워크 데이터에 대해서는 이를 반복적으로 체크하여, 데이터가 전달된 즉시 이를 상태머신으로 전달하도록 하여야 한다. 또한 추가적으로 상태머신들 간의 통신을 위해 이 모듈을 사용할 수 있다. 상태머신은 이벤트 브로커 모듈이 전달하는 이벤트에 의해서만 전의가 발생하기 때문에 한 상태머신에서 다른 상태머신으로 메시지를 전송하기 위해서는 이벤트 브로커 모듈로 전송하려는 메시지를 등록하여 이 메시지가 다시 타깃 상태머신의 메시지 큐에 등록되도록 하는 과정을 거치도록 하여야 한다.

3.2 메시지 자료구조 설계

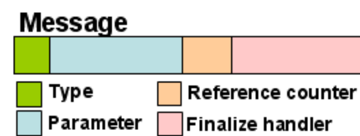

그림 5 메시지 자료구조

외부 환경에서 발생한 이벤트는 메시지 형태로 가공하여 상태 머신에 전달되게 된다. 이 메시지는 메시지의 타입과 메시지가 담고 있는 부수적인 데이터를 포함하고 있으며, 이러한 메시지는 여러 상태 머신 간에 공유하여 사용하여 메모리의 사용을 최소한으로 줄일 수 있도록 한다. 또한 메시지를 공유하기 위해 메시지 내부에 참조 카운터를 만들어 메시지를 사용하는 상태머신의 수 정보를 유지하다가 참조 카운터가 0이 될 경우 메시지를 삭제하는 과정이 필요하다. 메시지 삭제 과정에서도 각 메시지 타입에 따라 삭제해야할 추가 정보들이 다르기 때문에 메시지 타입에 따라 적절한 삭제루틴을 선택하여 자동적으로 데이터를 회수 할 수 있도록 하여야 한다. (그림 5)는 위의 요구사항을 고려하여 설계한 메시지의

자료구조 형태이다.

그리고 메시지 자료구조는 상태기반 시스템에서 매우 빈번히 생성되고 삭제되는 데이터이므로 임베디드 시스템의 메모리 단편화 현상을 줄이기 위해서는 메시지 자료구조를 풀로 관리하여 메시지 생성 시 이 풀에서 메시지를 꺼낸 후 사용이 끝난 메시지는 다시 풀로 환원하는 방법으로 구현해야 한다. 이 메시지 풀의 크기는 최대 시스템에서 사용되는 상태머신 수 * 메시지 큐의 크기로 설정 할 수 있으며, 이는 각 구현 응용프로그램에 따라 반복적인 다양한 실험을 통해 최적의 값으로 최소의 풀 크기를 설정 할 수도 있다.

3.3 메시지 큐

멀티 쓰레드 기반 시스템에서 메시지가 전달되더라도 이 메시지를 즉각적으로 처리 하지 못하고, 스케줄러의 선택을 기다리거나 다른 이벤트에 대기하는 등, 메시지 처리를 미루어야 하는 상황이 발생 한다. 이러한 상황을 위해 메시지를 저장할 장소가 필요하며, FIFO방식의 큐를 이용하여 메시지가 전달된 순으로 큐에 메시지를 저장한다. 큐에 저장된 메시지는 도착한 순서대로 다시 꺼내져서 처리를 하기 때문에 메시지의 시간적 의미가 일관성을 유지할 수 있도록 해야 하며, 큐의 입력은 ISR이나 이벤트 브로커 모듈의 쓰레드 등 여러 곳에서 일어날 수 있지만 큐를 읽는 쓰레드는 오직 그 큐를 소유한 실행 모듈의 쓰레드만 가능해야 한다. 즉, 상태 머신의 큐는 multiple-write/single-read 액세스를 필요로 한다[8]. 그렇기 때문에 메시지의 처리가 완료되기 전까지 다른 메시지를 처리하지 않고 큐에 대기하기 때문에 상태 머신의 Run-to-completion규칙은 지켜지게 된다. 메시지 큐는 각 실행 모듈에 소속되며 따라서 상태 머신의 개수만큼 메시지 큐가 존재한다. 정적 크기의 메시지 큐는 자원 제약이 심한 시스템에서 낭비가 심하기 때문에 실행되는 컨텍스트의 크기와 이벤트 발생 주기에 따라 시스템마다 그 크기를 최소로 설정해야한다. 그리고 메시지의 참조 카운터 유지를 위해 메시지가 큐에 삽입 될 때 메시지의 참조 카운터를 증가 시키고, 큐에서 나와 처리가 끝난 메시지는 참조 카운터를 감소 시켜야 한다.

3.4 실행 모듈

실행 모듈은 상태머신의 전이를 수행하고 전이에 동반된 전이액션을 수행하는 모듈이다. 상태머신 마다 각각의 실행 모듈을 가지고 있으며, 이 실행모듈은 각각 별도의 쓰레드로 실행된다. 메시지 큐 또한 실행모듈마다 별도로 가지고 있으며, 실행모듈은 이 메시지 큐로부터 데이터를 읽어 전이액션을 수행하는 동작을 반복한다. 모든 상태머신들은 각기 다른 쓰레드로 실행모듈을 동작시키고, 메시지 큐는 오직 그 메시지 큐를 소유한 실행모듈만이 읽을 수 있으므로, 전이 액션 중 다른 쓰레드에게 선점당하더라도 Run-To-Completion규칙은 지켜지게 된다. 또한 전이액션 코드에서 블로킹을 유발하는 함수를 사용하더라도, 스케줄러의 선점에 의해 다른 상태머신의 실행모듈을 실행시키므로 전체 시스템이 멈추는 현상은 없어진다. 실행모듈은 서로 관련이 없는 직교역역의 컴포넌트(Orthogonal Component)들을 서로 다른 실행모듈로 나누어 상태머신을 작성하고 ,서로 연관이 있는 모듈의 경우 같은 실행모듈에서 전이가 일어나도록 한 상태머신을 통해 구현한다[9]. 실행 모듈은 각기 우선순위를 가질 수 있어 상태머신의 중요도에 따라 실행모듈의 우선순위를 다르게 지정할 수 있다. Nano-Qplus에서는 멀티쓰레드를 지원하며, 실행모듈 하나는 쓰레드 하나로 매핑 시켜 구현한다.

3.5 상태 전이 구현

상태머신 프레임워크에서 사용될 상태머신 모델은 출력(전이 액션)이 현재 상태의 함수가 되는 무어 머신(Moore machine)이 아닌, 출력이 현재 상태와 입력함수가 되는 밀리 머신(Mealy machine)의 경우를 묘사한 것이다. 상태머신을 구현 하는 방법에는 상태를 정의하고 저장하는 방법과 상태를 전이하는 방법에 따라 여러 가지 구현 방법이 있다.

첫 번째 중첩된 switch문을 이용하는 방법이 있다. 이는 상태 저장을 위해 스칼라 변수 1개를 사용하고, 첫 번째 switch문으로 상태 변수를 구별하고 두 번째 switch문으로 메시지 종류를 구별하는 방법이다. 이는 구현이 간단하고, 메모리 소모가 적은 장점이 있지만, 하나의 상태에 관련된 코드가 여러 곳에 분산되고 반복됨에 따라 상태 토폴로지의 변화가 자유롭지 못하며, 상태가 많을 경우 O(log n)로 처리 속도가 떨어지는 단점이 있다.

두 번째 상태 테이블을 이용하여 구현하는 방법이 있다. 이는 현재 상태를 구별하고 메시지에 따라 메시지를 처리함수에 전달하는 처리속도가 O(1)로 성능이 좋으며, 공통된 메시지 처리기의 코드 재사용이 용의한 장점이 있다. 하지만 테이블 엔트리마다 액션을 나타내는 상세한 함수가 많이 필요하고, 메시지의 종류의 개수에 따라 상태 테이블의 크기를 유지해야 하므로 상태 테이블에 사용되는 메모리가 많이 소모되기 때문에 상태머신 프레임워크처럼 메시지의 종류가 실제 응용프로그램에서 사용되는 메시지보다 많은 경우 상태테이블을 위한 메모리 낭비가 심한 단점이 있다.

이 프레임워크에서 사용될 상태머신 구현 방법은 현재 상태를 상태 핸들러에 대한 함수의 포인터로 저장하고, 메시지의 종류는 switch문을 사용해 메시지를 식별하는 방법을 사용 한다. 이는 현재 상태를 구별하는 것과 상태를 전의하는 방법이 함수포인터를 읽고 변경하는 것으로 간단하고 빠르며, 사용되는 메모리도 상태 저장을 위해 함수포인터만 사용하므로 메모리 사용이 효율적이다. 또한 한 상태에 대한 처리 코드가 한 함수로 분할되므로 상태 토폴로지 변화에 쉽게 적응 할 수 있는 장점이 있다[8].

3.6상태기반 프레임워크를 지원하는 Nano-Qplus구조

본 논문에서 제안하는 상태 머신 프레임워크가 적용된 센서네트워크 운영체제의 구조는 (그림6)과 같다. 기존 Nano-Qplus에서 제공하는 MCU, RF통신, 센서 하드웨어를 제어하는 하드웨어 계층과, 멀티 쓰레드를 위해 태스크를 관리하는 모듈, 파워관리모듈, RF통신 관리 모듈, 센서 관리 모듈이 있다. 상태머신 프레임워크에서는 이

러한 모듈을 이용하여 이벤트 수집, 메시지 전달, 메시지 큐, 상태머신 실행 모듈을 제공한다. 이 프레임워크는 커널 모듈 형태로 제공되며 따라서 컴파일 시 선택적으로 프레임워크를 포함시킬 수 있다. 최상위의 응용 프로그-

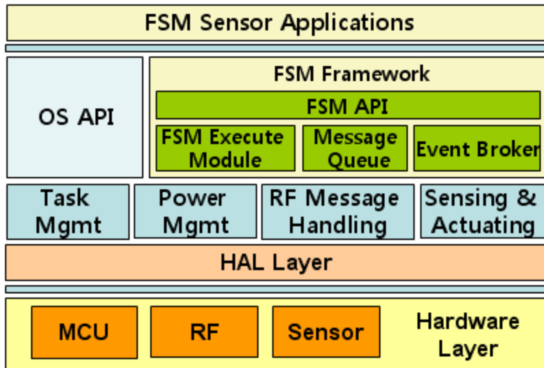

그림 6 상태머신 프레임워크 운영체제 구조

램 계층에서는 상태 프레임워크에서 제공하는 API와 운영체제에서 제공하는 API를 사용하여 응용 프로그램의 로직을 구현한다.

4. 결론 및 향후 과제

센서 노드의 제한된 자원에서 빠른 동작을 위한 많은 범용 센서네트워크용 운영체제가 개발되었다. 그 중 대표적인 TinyOS[3]는 이벤트 기반(Event Driven)의 실행 모델을 가지고 있어, 반응형 시스템을 디자인 하는데 있어 매우 좋은 구조를 가지고 있다. 하지만 상태를 구별하고 전이, 저장 하는 방법을 제공하지 않으며, 싱글스택 기반이기 때문에 이벤트 핸들러에서 긴 코드를 제공하지 못하는 단점이 있다.

본 논문에서는 센서 노드의 자원적 제한을 지키면서 응용프로그램을 상태머신 기반으로 개발할 때 기반이 되는 프레임워크의 구조를 제안하였다. 이 프레임워크는 상태 표현과 전이, 저장에 대한 방법을 API를 통해 제공하고, 멀티 쓰레드 기반의 운영체제 사용하여 Run-To-Completion를 지키면서 메시지 핸들러를 위한 긴 코드를 지원한다.

향후 과제로, 본 논문에서 제시한 프레임워크를 실제 센서네트워크 운영체제에 구현하여, 프레임워크로 인한 코드 크기의 증가와 실행 시간에 따른 메모리 오버헤드를 측정하여 기존 응용프로그램에서 상태머신을 구현한 것과 비교, 평가해야 하며, 이 프레임워크를 사용하여 응용프로그램을 작성할시 프로그래머가 좀 더 쉽게 FSM 모델을 검증하고 구현할 수 있도록 코드를 자동 생성할 수 있는 CASE툴의 설계와 개발에 대한 연구가 이루어져야 한다.

5. 참고 문헌

[1]I.F.Akyildiz,W.Su,Sankarasubramaniam,E.Cayirci,"A Survey on Sensor Networks", IEEE Communications Magazine, August 2002

[2] J.Kumagi,"The Secret Life of Birds", IEEE Spectrum, April 2004, vol. 41, issue 4, pp. 42-49

[3] J.Hill, R.Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister "System Architecture Directions for Networked Sensors". Proceedings of Ninth International Conference on Architectural Support for Programming Languages and Operating Systems(ASPLOS), November 2000.

[4] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, R. Han, "MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms, " ACM/Kluwer Mobile Networks & Applications (MONET), Special Issue on Wireless Sensor Networks, vol. 10, no. 4, August 2005, guest co-editors P. Ramanathan, R. Govindan and K. Sivalingam, pp. 563-579.

[5] C. Han, R. Rengaswamy, R. Shea, E. Kohler and M. Srivastava. "SOS: A dynamic operating system for sensor networks" Proceedings of the Third International Conference on Mobile Systems, Applications, And Services (Mobisys), 2005

[6] Nano-Qplus, (Web Page) http://qplus.or.kr

[7] Seungmin Park, Jin Won Kim, Kee-Young Shin, Daeyoung Kim, “A nano operating system for wireless sensor networks,” Advanced Communication Technology, 2006. ICACT 2006. The 8th International Conference Vol 1, pp. 20-22 Feb. 2006.

[8] Miro Samek "Practical Statecharts in C/C++: Quantum Programming for Embedded Systems" CMP BOOKS(2002)

[9] Douglass, Bruce Powell "Doing Hard Time, Developing real-time systems with UML, Object, Frameworks, and Patterns" Addison Wesley(1999)