DOI: 10.3745/KIPSTA.2010.17A.3.127

멀티쓰레드 기반 센서네트워크 운영체제에서 동작하는 상태머신 프레임워크

이 승 근[†] · 김 병 곤[††] · 최 병 규[†††] · 허 신[††††]

요 약

무선 센서네트워크는 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 센서네트워크 운영체제가 요구된다. 또한 센서네트워크는 외부 물리 환경의 변화에 반응하여 동작하는 시스템이기 때문에 여러 이벤트를 동시에 신속하게 처리 할 수 있어야 한다. 이러한 요구조건을 만족시킬 수 있도록 TinyOS나 MANTIS, NanoQplus 등 센서네트워크용 운영체제에서 다양한 기법들이 제시되고 있다. 하지만, 센서네트워크 응용 프로그램을 개발하는 프로그래머 입장에서는 제약이 심한 개발 환경과 개발을 용이하게 할 수 있도록 하는 프레임워크가 부족한 문제점도 매우 크다. 이를 위해 본 논문에서는 반응형 시스템에 적합한 상태머신 프레임워크를 멀티쓰레드 기반의 센서네트워크운영체제인 NanoQplus에 구현하였다. 또한 효과적인 이벤트 처리를 위한 이벤트 브로커 모듈 및 상태머신간 메시지 공유를 위한 메시지 자료구조와 메시지 및 메시지큐를 핸들링하고 상태머신의 전이를 수행하는 실행 모듈을 제안한다. 추가적으로 상태머신 프레임워크기반의 응용 프로그램을 좀 더 용이한 개발을 지원할 수 있는 CASE(Computer-aided software engineering)툴을 개발하였다.

키워드 : 센서네트워크, 상태머신, 센서네트워크 운영체제

State Machine Frameworks Operating in Sensor Network Operation System based on Multi-Thread

Seung Keun Lee[†] · Byung Kon Kim[††] · Byoung Kyu Choi[†††] · Heu Shin[††††]

ABSTRACT

A wireless sensor network(WSN) which roles as a mediator between living environment and computers in ubiquitous computing is very essential research area. Due to the constraint that sensor nodes should work in very resource-restricted circumstances, an operating system that can manage resources effectively is demanded. Also, a sensor network should be able to deal with many events quickly and simultaneously in order to respond to various physical changes in outer environment. The Sensor Network Operating System such as TinyOS, MANTIS and NanoQplus is much designed so that it can satisfy such requirement. But, for programmers who develop application program for sensor networks, they have lack of frameworks which the development is easily possible from restricted development environment. In this paper for this, we implemented a state machine framework apt for responsive systems in NanoQplus which is multi-thread-based sensor network operating system. In addition we propose an event broker module(EBM) for effective event dispatching, a message data structure for message sharing among state machines, and an execution module that handles messages and their queue and performs state transition of the machines. Furthermore, we could do the development more easily an application program with a state-based framework by developing CASE tools.

Keywords : Sensor Network, State Machine, Sensor Network Operation Systems

1. 서 론

무선 센서네트워크(Wireless Sensor Network; WSN)는 군사 작전 지역, 산업 시설, 생태 환경 등에 다량의 센서를 배치하여 데이터를 수집/가공하고, 가공된 데이터를 센서 노드간 무선 통신을 통하여 최종 사용자에게 필요한 데이터를 전송하는 기술[1, 2]로, 국내에서는 유비쿼터스 센서네트워크

† 정 회 원 : 삼성전자 DMC연구소
†† 정 회 원 : 한국건설기술연구원 건설정보연구실 수석연구원
††† 준 회 원 : 한양대학교 컴퓨터공학과 박사 과정 수료
†††† 정 회 원 : 한양대학교 컴퓨터공학과 교수

논문접수 : 2010년 4월 12일
수 정 일 : 1차 2010년 5월 25일
심사완료 : 2010년 5월 31일

(Ubiquitous Sensor Network; USN)으로 지칭되기도 한다. 일반적으로 센서네트워크를 구성하는 센서 노드는 극도로 제한된 자원을 가지고 있어, 8bit MCU(Micro Controller Unit; MCU)와 8~128KB 정도의 프로그램 가능한 메모리, 512B~4KB 정도의 RAM으로 동작한다[3]. 따라서 센서네트워크용 운영체제는 크기가 매우 작아야 하며, 메모리와 전원 등 제한된 자원을 효율적으로 이용할 수 있는 기법이 필수적으로 요구된다. TinyOS[3], MANTIS[4], SOS[5], NanoQplus[6]와 같은 범용 센서네트워크용 운영체제는 컴팩트한 크기와 자원의 효율적인 활용을 위한 기법이 적용된 운영체제로 일반적은 운영체제에 비하여 개발 및 사용자 환경이 열악하다.

센서네트워크 시스템은 외부 물리 환경의 이벤트에 반응하는 반응형 시스템이다. 같은 이벤트에 대해서도 프로그램의 모드에 따라 입력 이벤트를 다르게 처리하는 경우가 있기 때문에, 시스템 설계 시 상태머신(State Machine) 기반의 컴퓨팅 모델을 이용하는 것이 적합하다. TinyOS[3]나 SOS[5]는 이러한 상태머신 기반의 프로그래밍 환경을 제공하여 반응형 시스템 설계를 좀 더 용이하도록 하고 있다. 하지만 TinyOS와 SOS는 싱글 스택(single stack) 기반으로 스택을 공유하는 시스템이기 때문에, Run-To-Completion을 지키기 위해 하나의 이벤트를 처리 하는 동안에는 다른 이벤트를 처리 할 수가 없다. 따라서 어플리케이션 개발에 있어서 이벤트 처리 코드를 최대한 짧게 유지해야 하는 제약 사항이 있다[4]. MANTIS나 NanoQplus의 경우 유닉스와 비슷한 프로그래밍 환경을 제공한다는 점에서 기존 시스템 프로그래머가 센서네트워크용 어플리케이션을 작성할 때 좀 더 익숙한 환경에서 개발할 수 있다는 장점과 기존의 유닉스 환경의 C 코드를 거의 수정하지 않고 사용할 수 있는 장점이 있다. 그러나 상태차트를 이용한 상태머신 기반으로 프로그래밍할 때, 이를 지원할 프레임워크가 없기 때문에 상태머신 기반 프로그램의 모든 요소를 직접 개발해야 하는 어려움이 있어, 반응형 시스템에 적합한 상태머신 기반 컴퓨팅 모델을 사용하는데 큰 부담으로 작용한다.

본 논문에서는 센서네트워크가 갖는 자원의 제한조건을 준수하면서, 멀티 쓰레드 환경의 특징을 이용하여 반응 시간이 짧은 상태머신 기반 프레임워크를 설계하고 구현하였다.

본 논문의 구성은 다음과 같다. 2장에서 관련연구로 기존 센서네트워크 운영체제의 특징과 한계를 설명하고, 3장에서 센서네트워크 환경에서 상태머신 프레임워크의 설계와 구현 사항을 기술한다. 4장에서는 구현한 프레임워크를 사용하여 응용프로그램을 작성한 후, 프레임워크를 사용 했을 때 메모리 사용량을 조사한 후, 기존 프레임워크가 없는 상황과 비교를 통하여 제안한 프레임워크의 효용성에 대해 설명하고 결론을 맺는다.

2. 관련 연구

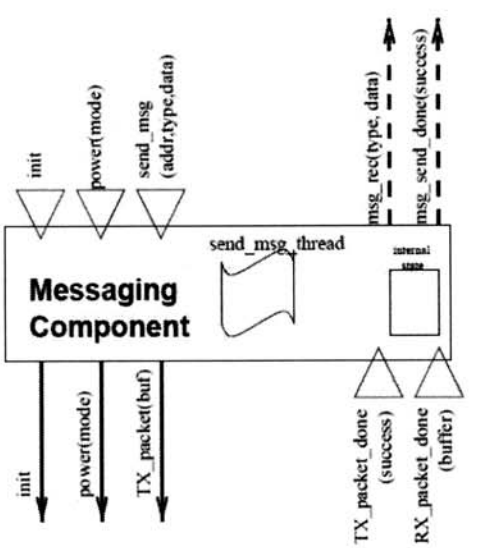
2.1 범용 센서네트워크 운영체제

대표적인 센서네트워크용 운영체제로 버클리 대학의 TinyOS[3]가 있다. 이는 제한된 자원에서 실행되기 위한 구조로 설계되어 있으며, 컴포넌트 기반의 구조를 가지고 있어 계층적인 컴포넌트가 모여 하나의 시스템을 이룬다. 운영체제 이미지는 컴파일할 시점에 사용되는 응용에 필요한 모듈만 적재시킴으로서, 최소한의 크기로 운영체제 이미지를 생성할 수 있다. 또한 이벤트 기반(Event Driven) 실행 모델을 가지고 있어, 외부 이벤트에 반응하고 그 이벤트가 계층적 모듈을 통해 상위 모듈로 전달되면서 시스템이 동작하도록 설계되어 있다.

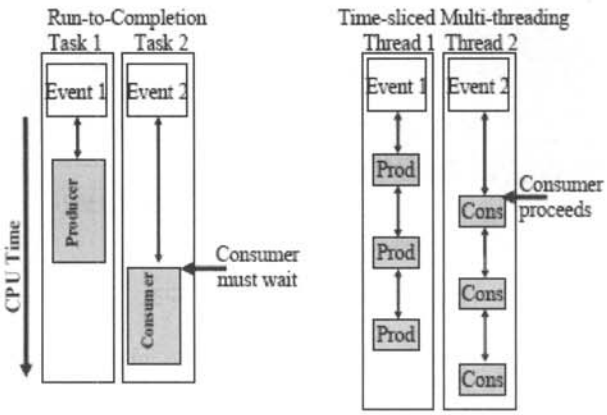
각 모듈은 (그림 1)과 같이 EVENT/COMMAND 구조로 하위 모듈로부터 발생하는 이벤트를 상위 모듈에서 이벤트 핸들러를 이용하여 이벤트에 대한 처리를 한다. 처리된 이벤트는 다시 상위 모듈로 전달하여 하드웨어에서 발생한 이벤트가 최상위 응용 프로그램 모듈로 전달되게 한다.

상위 모듈에서는 COMMAND를 하위 모듈로 전달하며 이는 다시 H/W모듈까지 순차적으로 전달되어 시스템을 동작하게 한다. 이 이벤트 기반(Event Driven) 실행 모델은 각 모듈을 유한 상태머신에 대응하여 설계할 수 있으며, 따라서 TinyOS를 상태머신 기반의 운영체제라고 한다. 하지만 TinyOS는 시스템적으로 각 모듈의 내부 상태를 기술할 방법은 명시하지 않고 있으며, 상태를 전이하는 방법도 특별히 제공하지 않는다. 따라서 모듈 개발 시 상태는 Frame영역에 내부 상태를 변수나 플래그로 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, 런타임시에서 검사하기에도 부담이 크다[8].

TinyOS는 메모리 자원의 효율적 이용을 위해 싱글 스택(single stack)방식으로 모든 태스크가 하나의 스택을 공유하는 방법으로 구현되어 있다. 따라서 (그림 2)와 같이 한



(그림 1) EVENT/COMMAND구조 모듈[3]



(그림 2) TinyOS와 MANTIS의 실행 비교[4]

이벤트 핸들러가 종료되기 전까지는 다른 task는 수행될 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]. 반면 콜로라도 대학에서 개발한 MANTIS라는 멀티 쓰레드 기반의 센서네트워크용 운영체제는 전통적인 UNIX시스템과 비슷한 구조를 가지고 있어 멀티 쓰레드 기반으로 동작하기 때문에 한 로직 수행에서 긴 수행 시간이 소모 되더라도, 우선순위가 높은 다른 쓰레드에 의해 선점이 가능하므로 잘못된 기능으로 인한 전체 시스템이 정지되는 오류를 방지할 수 있다. MANTIS는 C언어로 응용 프로그램을 작성하도록 되어 있으며 이는 순차적인 프로그램을 작성하기에는 적합하지만, 외부에서 발생하는 이벤트에 반응하는 시스템을 작성하기에는 적합하지 못한 구조를 가지고 있다. 그 이유는 하드웨어로부터 이벤트가 발생하고, 이벤트를 큐에 넣고, 이벤트를 처리하는 로직에서 큐로부터 이벤트를 꺼내 처리하는 모든 과정을 응용 프로그래머가 작성해야 하기 때문에 실제로 이벤트를 처리하는 코드에 집중할 수 없다.

2.2 멀티 쓰레드 상에서 상태머신(FSM)기반 실행 환경

많은 임베디드 응용 프로그램은 외부 이벤트, 입력에 의해 특정한 결과를 내는 반응형 시스템의 특성을 가진다. 이러한 시스템을 모델링하는 적합한 방법으로 *Harel*은 시각적인 형태의 모델링을 위해 상태 차트[7]를 이용한 방법을 소개하였다. 센서네트워크 응용 프로그램 또한 외부 이벤트 입력에 의해 특정 행동을 수행하는 반응형 시스템의 한 예이다. 연속된 이벤트/입력에 의해서 시스템의 내부 모드가 변경되며, 이에 따라 시스템이 행동하는 방법이 달라진다. 상태머신 기반 프로그래밍 방법으로 이러한 시스템을 구현할 경우 상태머신은 이러한 다양한 모드를 상태로 구현하며, 연속된 여러 입력에 의해 상태가 다른 상태로 전이하며, 그 상태에 적합한 전이 액션을 수행한다. 이 전이 액션에 의해 그 상황에 적합한 행동을 수행한다. 상태머신을 이용한 설계 방법은 반응형 시스템을 모델링하기 좋은 방법이며, 시스템 구현시에도 적합한 방법이다. 이러한 상태머신 기반의 프로그래밍은 싱글 쓰레드상에서 구현시, 코드의 가독성이나 오류처리 및 유지보수에 장점을 가지나 이벤트 처리코드를 짧게 유지해야하며 오류 시 전체 시스템이 중단되는 단점을 가진다. 이에 본 논문에서는 멀티 쓰레드 기반의 NanoQplus상에서 FSM을 구현하였다. 멀티 쓰레드 환경에서 상태머신 모델링을 적용했을 때 얻는 이점은 다음과 같다. 병렬적인 작업을 각 쓰레드를 통해 처리하는 것 대신에 FSM을 이용해 논리적으로 모델링할 수 있으며, 싱글 쓰레드 상에서 동작하는 FSM의 제약을 극복할 수 있기 때문이다.

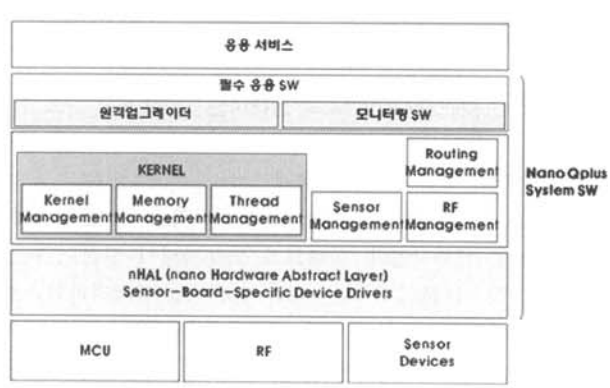
2.3 NanoQplus 운영체제

NanoQplus는 한국전자통신연구원(ETRI)에서 개발 중인 센서네트워크용 운영체제로써 다음과 같은 특징을 가진다[13].

- 에너지 소모를 최소화하기 위해 네트워크를 구성하는 노드 사이에 동기화된 시간을 기반으로 주기적으로 휴면(Sleep)과 활성화(Active) 모드를 반복하면서 메시지를 효율적으로 처리
- 멀티 쓰레드 스케줄러
- 표준형 및 마이크로 임베디드 OS API와 동일한 API를 사용함으로써 기존 시스템 프로그램 개발자에게 친밀감 제공

(그림 3)에서 볼 수 있듯이 NanoQplus 운영체제는 기존의 리눅스 시스템과 유사한 계층적 구조를 가지고 있다. 기존의 클래식한 운영체제의 형태를 유지하면서 불필요한 모듈을 제거하고, 운영체제의 각 기능들을 최소화함으로써 센서네트워크에서 사용 가능하도록 크기를 줄였다. 전체적인 운영체제의 형태는 기존의 운영체제와 비슷하지만 실제 센서네트워크에 최적화하기 위하여 모듈화를 통한 재구성이 가능하다.

센서네트워크응용에 따라 불필요한 모듈은 완전하게 제거가 가능하다. 실제로 응용 프로그램을 만들 때, 스케줄러와



(그림 3) NanoQplus 운영체제 구조[6]

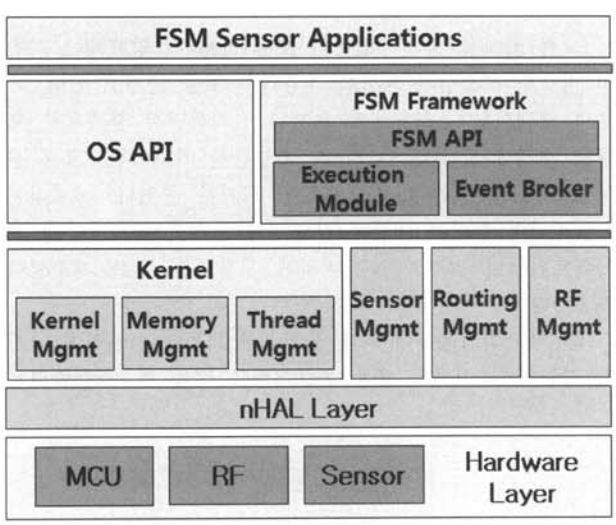
같은 기능이 필요 없다면 nHAL의 드라이버만 이용하여 프로그램을 작성하는 것도 가능하다. 일반적으로 생각하는 커널의 개념과는 완전히 다른 개념으로 동작한다. 또한 센서 노드용 프로세서에는 보호 모드가 없고 응용 프로그램과 커널 코드가 같은 모드에서 동작하므로 유저 레벨과 커널 레벨의 구분이 무의미하다.

3. 상태머신(FSM) 프레임 워크의 설계 및 구현

3.1 NanoQplus상에서 구현한 상태기반 프레임워크 구조

본 논문에서 제안하는 상태머신 프레임워크는 (그림 4)과 같은 구조를 갖고 있으며 NanoQplus 상에서 구현하였다. 상태머신 프레임워크는 운영체제에서 제공하는 센서 모듈, 네트워크 모듈, 메시지 큐, 멀티쓰레드 모듈을 이용하여 Event Broker Module 및 FSM Execute Module을 구현하였다. 상태머신 프레임워크의 footprint를 최소로 하기 위해 운영체제에서 제공하는 기능을 최대한 이용하여 구현하는 것을 원칙으로 하였다. 이를 위해 센서 운영체제에서는 쓰레드API, 메시지 큐, 세마포어를 지원해야 한다. NanoQplus 2.4.0 에서는 멀티쓰레드를 위한 세마포어, 메시지 큐를 제공하고 있어서 대부분 기능을 OS API로써 구현 가능하다.

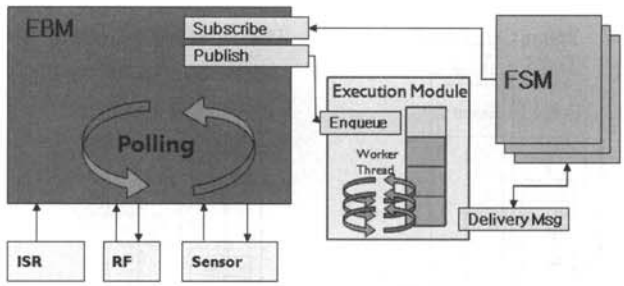
응용 프로그램 개발자는 상태머신 프레임워크 API와 OS API를 이용하여 상태머신 기반 응용 프로그램을 개발한다.



(그림 4) NanoQplus상에서의 상태머신 프레임워크 구조

3.2 상태머신 프레임워크의 설계

본 논문에서 구현한 상태머신 프레임워크는 (그림 5)와 같은 구조로 설계되었다. 이벤트를 발생시키기 위해 인터럽트 서비스루틴(ISR)이나 네트워크 디바이스, 센서디바이스로부터 데이터를 읽어와 특정한 메시지 자료구조로 가공 후 상태머신으로 전달하기 전까지의 작업을 담당하는 이벤트 브로커 모듈(이하 EBM모듈)이 있고, EBM에서 발생된 이벤



(그림 5) 상태머신 프레임워크 구조

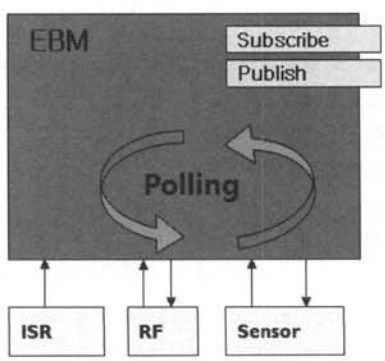
트를 상태머신 모듈(이하 FSM)로 전달하고, 이벤트에 해당하는 전이 액션을 수행하기 위해 Execution Module이 있다. 이 모듈은 전이를 수행하는 Worker Thread로 구성되며 쓰레드 수는 설정 가능하다. FSM은 상태머신의 현재 상태를 기록하며 각 상태마다 전이 액션을 정의할 수 있도록 인터페이스를 제공한다. 각 상태머신의 전이는 다른 상태 전이를 수행하기 전에 이전 전이가 완료되어야 하는 Run-To-Completion을 지켜야 하기 때문에 각 FSM 인스턴스마다 세마포어를 이용하여 전이 수행 중 다른 이벤트에 의해 전이가 발생하지 않도록 한다.

3.2.1 이벤트 브로커 모듈(EBM) 설계 및 구현

EBM모듈은 인터럽트 서비스 루틴이나 네트워크 디바이스 , 센서 등 외부로부터 발생할 수 있는 이벤트를 관리하고 발생된 이벤트를 상태머신에서 사용할 수 있는 메시지 형태로 가공하여, 이 메시지의 구독을 원하는 상태머신에게 전달하는 역할을 한다. 본 논문에서 제안하는 메시지 브로커 모듈의 구조는 (그림 6)과 같다.

이 모듈을 이용해 상태머신은 자신이 주기적으로 받기를 원하는 센서 데이터에 대해, 수집 주기와 센서 데이터의 종류를 등록하게 된다. 이 모듈은 이러한 구독과 수집에 대한 정보를 유지하며 , 타이머 인터럽트에 의해 특정 주기로 센서로부터 데이터를 수집해 전달하거나 비동기적으로 발생하는 이벤트에 대해서는 즉각적으로 전달한다. 이를 위해 이벤트 브로커 모듈은 2가지 형태로 동작을 해야 한다.

첫번째 주기적인 데이터 수집에 대해 상태머신에서 등록한 주기와 데이터 종류를 관리한다. 이때, 동일한 반복문 안



(그림 6) 이벤트 브로커 모듈

에서 타이머 인터럽트에 의해 깨어나 저장된 주기와 현재 주기를 비교하여, 올바른 주기에 데이터를 센서로부터 수집/전달해야 한다. 또한 상태머신별로 설정된 데이터 수집 주기를 정규화하여 한 번 수집한 데이터에 대하여 다른 상태머신에게 전달하여, 불필요하게 반복적인 데이터 수집을 피하도록 하여야 한다.

이를 위해 주기적으로 센서 데이터를 수집하는 Ebm_polling함수를 구현하였다. 이 함수는 루프문 안에서 미리 설정한 Polling 시간 주기마다 쓰레드가 깨어나 구독을 원하는 센서 데이터에 대해 읽기를 시도한다. 이 시간 주기는 모든 FSM에 공통으로 적용되는 시간 주기이며, EBM모듈을 처음 초기화 할 때 설정한다. 각 FSM마다 다른 시간 주기를 설정하기 위해는 term이라는 수집 주기 빈도를 나타내는 값을 사용한다. 이 값에 저장된 수 만큼 앞에서 설정한 Polling 주기 시간을 보낸 후 데이터 수집을 수행한다. 즉, Polling 주기 * term 값이 각 FSM의 데이터 수집 주기로써 사용된다.

두번째 비동기적으로 발생하는 이벤트에 대해서는 이벤트가 발생한 즉시, 이벤트를 메시지로 가공하여 구독을 원하는 상태머신에게 전달해야 한다. 이를 위해 인터럽트 서비스 루틴 안에서 이벤트를 가공하고 전달하는 함수를 호출한다. NanoQplus상 구현에서는 네트워크 데이터 수신 callback 함수에서 이 기능을 구현하였다. 네트워크 데이터가 수신된 후 callback함수가 호출되면 이 데이터를 메시지 자료구조로 변환한 후 FSM에 바로 전달 된다.

EBM모듈은 각 FSM마다 구독을 원하는 센서 데이터와 구독 수집 주기를 저장하고 있어야 한다. 이벤트 구독자 목록 저장을 위해, 각각의 FSM의 구독 여부를 1bit로 On/Off 하여 16bit 변수에 구독 여부를 저장하고, 이러한 16bit 변수를 발생 가능한 이벤트 종류의 하나로 매핑하여 관리한다. 아래 코드와 같이 발생할 수 있는 이벤트 수만큼 16bit변수를 배열로 선언한다.

```
UINT16 EventSubscriber[FSM_S_MAXINDEX];
```

변수의 크기를 16bit로 선택한 이유는 동시에 실행될 FSM머신의 총 개수를 반영한 것이기 때문에 센서 노드 상에 사실상 8개 이상의 모듈이 동시에 수행되는 것은 거의 없기 때문에 메모리 최소 사용을 위해 8bit변수로 변경 가능하다. 이러한 방법으로 구독자 목록을 저장하였기 때문에, 특정 이벤트 발생 시 구독자 존재 여부를 찾는 연산은 O(1)의 연산으로 검사가 가능하다.

```
// 빛정보 이벤트에 대한 구독자 존재 여부 판별
if( EventSubscriber[FSM_S_LIGHT] )
{ ...          }
```

또한 해당 센서 데이터 구독을 원하는 FSM목록을 구축하는 것 또한 O(1)의 연산으로 가능하다. 그렇기 때문에 발생한 이벤트에 대해 구독을 원하는 모듈로 선별하고 전달하는 작업이 큰 부하 없이 수행 가능하다.

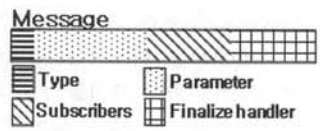
3.2.2 메시지 자료구조와 전달 메커니즘 설계 및 구현

외부 환경에서 발생한 이벤트는 메시지 형태로 가공하여 상태머신에 전달되게 된다. 이 메시지는 메시지의 타입과 메시지가 담고 있는 부수적인 데이터를 포함하고 있다. 이러한 메시지는 여러 상태머신 간에 공유하도록 하여 사용하는 메모리를 최소한으로 줄일 수 있도록 한다. 이 때문에 메시지는 구독을 원하는 상태머신에 순차적으로 전달할 수 있는 방법이 필요하다. 이를 위해 메시지를 구독하기 원하는 상태머신을 저장하기 위한 필드가 존재한다. 이 필드의 각 비트는 각 상태머신의 구독 여부를 나타내며, 메시지를 전달 받은 상태머신은 메시지를 처리한 후 자신의 상태머신에 해당하는 비트를 0으로 변경 후 다음번 비트가 켜진 상태머신을 검색하여 해당 상태머신으로 메시지를 전달한다. 이런 과정이 반복되면 구독자 비트가 모두 0으로 바뀌게 되며 메시지 전달 과정이 종료된다.

위 전달 과정이 끝난 후 메시지는 삭제 과정을 거쳐야 하며 각 메시지 타입에 따라 삭제해야할 추가 정보들이 다르기 때문에 finalize에 저장된 삭제 루틴을 호출하여 적절한 삭제 과정을 거치도록 하였다.

(그림 7)는 위의 요구사항을 고려하여 설계한 메시지의 자료구조 형태이다.

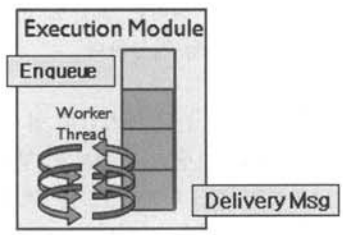
그리고 메시지 자료구조는 상태기반 시스템에서 매우 빈번히 생성되고 삭제되는 데이터이므로 임베디드 시스템의 메모리 단편화 현상을 줄이기 위해서는 메시지 자료구조를 풀로 관리하여 메시지 생성 시, 이 풀에서 메시지를 꺼낸 후 사용이 끝난 메시지는 다시 풀로 환원하는 방법으로 구현해야 한다. 이 메시지 풀의 크기는 최대 시스템에서 사용되는 상태머신 수 * 메시지 큐의 크기로 설정할 수 있으며, 이는 각 구현 응용 프로그램에 따라 반복적인 다양한 실험을 통해 최적의 값으로 최소의 풀 크기를 설정할 수도 있다.



(그림 7) 메시지 자료구조

3.2.3 실행 모듈 설계 및 구현

실행 모듈은 상태머신의 전이를 수행하고 전이에 동반된 전이액션을 수행하는 모듈이다. 실행 모듈은 코드를 수행하는 모듈로 센서운영체제의 쓰레드 모듈로 구현 하였으며 그 구조는 (그림 8)과 같다. 센서 운영체제는 사용 가능한 자원이 매우 부족하여 센서 응용에 따라 사용 가능한 쓰레드 개수에도 제약을 받는다. 따라서 실행 모듈은 최소 1개의 쓰레드를 가지고도 동작 가능하며, 자원의 여유에 따라 쓰레



(그림 8) 실행 모듈 구조

드 개수를 늘려 사용 할 수 있도록 설계 하였다.

EBM에서 발생된 이벤트는 실행 모듈로 전달되며 이는 실행모듈의 이벤트 큐에 저장된다. 이렇게 큐로 입력된 이벤트는 N개의 쓰레드가 큐에서 꺼내와 구독을 원하는 상태머신에 전달하고 전이 액션을 수행한다. 전이 액션을 수행할 때는 이전 전이 액션의 수행이 끝난 후 수행을 시작해야 하는 Run-to-completion을 지키도록 구현 되어야 하며, 이를 위해 각 상태머신은 자신만의 세마포어를 가지고 있어서 전이 액션의 수행 중에 다른 전이 액션이 수행되지 않도록 한다. 하지만 긴 전이 액션을 수행하더라도, 여러 개의 쓰레드로 실행모듈이 동작하기 때문에 전이 액션중인 상태머신이 아닌 다른 상태머신은 계속 전이를 수행 할 수 있다. 따라서 전이 액션 중 긴 wait를 필요로 하는 시스템 콜을 호출 하더라도 (예를 들어 뮤택스 획득 시도) 전체 시스템이 그 시간동안 정지하는 경우를 방지할 수 있다. 위와 같은 시스템 작동을 위해 상태머신은 서로 관련이 없는 직교 영역의 컴포넌트(Orthogonal Component)들을 서로 다른 상태머신으로 나누어 작성하여 의존 관계가 없는 전이 액션이 서로 수행에 방해 받지 않도록 해야 한다[9]

3.2.4 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 구현하는 방법에는 상태를 정의하고 저장하는 방법과 상태를 전이하는 방법에 따라 여러 가지 구현 방법이 있다.

첫번째 중첩된 switch문을 이용하는 방법이 있다. 이는 상태 저장을 위해 스칼라 변수 1개를 사용하고, 첫 번째 switch문으로 상태 변수를 구별하고 두번째 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.

두번째 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 용이한 장점이 있다. 하지만 테이블 엔트리마다 액션을 나타내는 상세한 함수가 많이 필요하고, 메시지 종류의 개수에 따라 상태 테이블의 크기를 유지해야 하므로 상태 테이블에 사용되는 메모리가 많이 소모되기 때문에 상태머신 프레임워크처럼 메시지 종류가 실제 응용프로그램에서 사용되는 메시지보다 많은 경우 상태 테이블을 위한 메모리 낭비가 심한 단점이 있다.

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

3.2.5 FSM Engine 모듈 구현

FSM Engine 모듈은 상태머신 프레임워크의 많은 기능을 구현한 모듈로 FSM의 등록/삭제 관리와 FSM의 메시지 처리 루틴, 상태전이 루틴, FSM 정보 쿼리, 메시지 풀 초기화 등을 담당하고 있다.

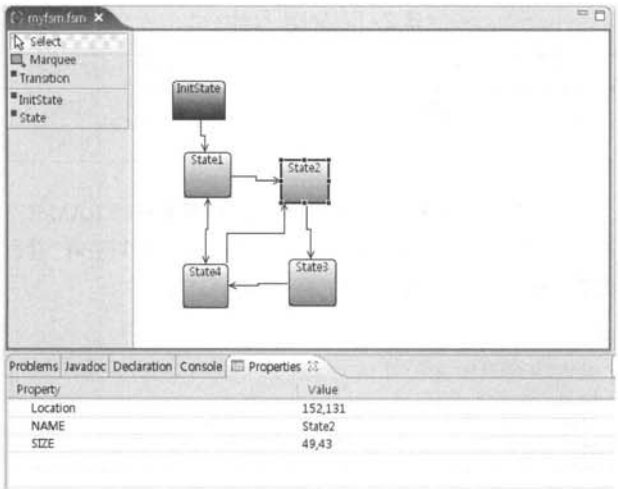
FSM모듈의 자료구조는 다음과 같다

```
typedef struct _FSM_Module{
        UINT8 ID;
        UINT16 subscribes;
        void (*init)(void *);
        void (*state)(void *);
        SEMAPHORE sm;
}FSM_Module;
```

ID는 모듈을 식별하는 식별자로 FSM Engine에 모듈을 등록할 때 자동으로 부여된다. subscribes는 현재 구독중인 이벤트 목록을 저장한다. init함수 포인터는 FSM모듈이 처음 구동되기 전 변수의 값을 초기화하는 등의 초기화 루틴을 수행하는 것을 제공하기 위해 존재하며, FSM엔진에 FSM을 등록할 때 자동으로 init함수를 실행하도록 구현하였다. state는 상태 처리함수의 포인터로 현재 상태에 따라 그 처리함수를 state함수 포인터에 저장한다. 따라서 현재 상태는 state에 저장되고, 현 상태의 메시지처리 방법 또한 state함수를 통해 처리된다. 세마포어는 FSM의 run-to-completion을 지키기 위해 사용되었다. 따라서 전이 액션인 state함수를 수행하기 전에 세마포어를 획득 후 호출하고, 수행이 끝난 후 세마포어를 반환한다.

3.3 상태머신 프레임워크를 위한 CASE툴 설계 및 구현

상태머신 프레임워크는 상태기반 프로그래밍을 위한 프로그래밍적 요소뿐만 아니라 상태차트를 이용하여 설계한 것을 프로그램 코드로 변환하는 CASE (computer-aided software engineering)도구가 필요하다. 센서네트워크 응용프로그램 개발자는 이 도구를 이용하여 시각적인 방법으로 응용프로그램을 설계하고, 상태 차트를 통해 프로그램 동작을 검증한 후, 이를 구현 코드로 변환하는 과정을 통해 응용프로그



(그림 9) NanoCASE 상태다이어그램 편집창

램을 개발한다.

본 논문에서는 그래픽적인 상태 다이어그램으로 상태 머신을 설계하고, 이 설계에 대응하는 상태머신 프레임워크용 C언어 소스로 변환하는 툴을 이클립스[10] 기반의 플러그인으로 구현하였다(NanoCASE). 이클립스 플러그인으로 구현한 이유는 NanoQplus의 어플리케이션을 개발하는 IDE가 이클립스로 되어 있고, 이클립스 플러그인을 통해 개발할 경우 많은 GUI 요소는 기존의 이클립스에서 제공하는 고수준의 GUI 툴킷인 SWT/JFace[11]를 이용하여 쉽게 작성할 수 있기 때문이다.

(그림 9)과 같이 상태다이어그램을 IDE 환경에서 디자인하게 되며 이는 논문에서 구현한 프레임워크에 적용할 수 있는 C언어 코드로 변환 된다. 이 CASE툴을 사용하는 응용프로그램 개발자는 툴에 의해 생성된 코드를 기반으로 응용 프로그램을 작성하기 때문에 응용프로그램 개발 시간을 단축시킬 수가 있다.

4. 실험 및 성능 평가

본 논문에서 제안하는 상태머신 프레임워크는 Nano-24 타겟에서 구현 및 테스트 되었다. Nano-24 타겟은 8bit 마이크로컨트롤러(ATMega128L)를 사용하며, RF(Chipcon CC2420) 통신과 Serial(RS-232c)통신 작업을 할 수 있다. 마이크로컨트롤러의 내, 외부 Timer Interrupt를 통해 여러 작업을 관리할 수 있으며, Real -Time 작업을 수행할 수 있다. 외부 Interrupt와 ADC Port등은 Sensor값을 읽어 올 수 있다. 마이크로컨트롤러 내부에 프로그램 저장 용도의 128kbyte의 Flash Memory와 4kbyte의 EEPROM이 있으며, 4kbyte의 SDRAM이 있다. 외부에 528kbyte의 외부 Flash Memory가 있지만 논문 구현에서는 사용되지 않았다.

4.1 상태머신 프레임워크를 이용한 응용 예제

상태머신 프레임워크의 정상 동작을 확인하기 위해 일반적인 센서 응용 프로그램 시나리오를 제시하고, 본 논문에서 구현한 프레임워크를 사용하여 응용 프로그램을 작성 후 올바른 로직으로 동작하는지 실험한다.

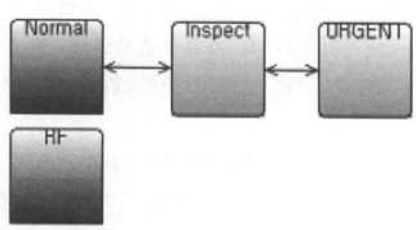
가상의 산불 감지 응용 프로그램의 시나리오는 다음과 같다.

```
상태 = { NORMAL, INSPECT, URGENT }

NORMAL 상태
 1. 1분 간격 온도 데이터 수집
 2. 온도가 N℃ 이상 증가 INSPECT 상태로 전이
 3. 근접 노드로부터 데이터 수신
    3.1 URGENT정보를 전달 받으면 INSPECT 상태로 전이

INSPECT 상태
 1. 30초 간격 온도 데이터 수집
 2. 30초 간격 빛 데이터 수집
 3. 빛 밝기가 N 이상이면 URGENT 상태로 전이
    3.1 다른 로드로 URGENT 정보 전달
 4. N시간 동안 빛 정보에 이상값이 발견되지 않을 경우
    NORMAL 상태로 전이

URGENT 상태
 1. 30초 간격 온도 데이터 수집
 2. 30초 간격 빛 데이터 수집
 3. 빛 정보와 온도 정보가 N값 이하로 떨어질 경우 INSPECT
    상태로 전이

모든 상태
 1. 10분 간격으로 수집한 데이터를 RF로 sink노드로 전달
```

위 응용 예제는 3개의 상태로 구현가능하며 NanoCASE 툴을 이용하여 아래 그림과 같이 디자인 가능하다.

2개의 상태머신으로 구현이 가능하며, 첫 번째 상태머신은 3개의 상태를 가지고 센서데이터를 수집 후 시나리오의 조건에 따라 각 상태로 전이하며 두 번째 상태머신은 독립적으로 센서데이터를 수집 후 RF통신을 이용하여 sink노드로 데이터를 전달하는 역할을 한다.

위 시나리오를 상태머신 프레임워크를 이용하여 구현하는데 2,512 바이트의 응용 프로그램 코드가 사용되었으며, NanoCase툴에 의해 생성된 코드의 크기를 제외하면 1,597 바이트 코드를 작성하였다. 그리고 상태머신 프레임워크를 사용하지 않고 작성한 경우 4,597 바이트의 코드가 필요하였다. 이는 프레임워크가 없는 경우 센서데이터를 획득하고 이를 처리할 코드로 전달하는데 필요한 코드를 모두 응용 프로그래머가 직접 작성해야 하기 때문에 더 많은 코드를 작성해야 했다. 이렇게 상태머신을 사용하여 프로그램에서



(그림 10) 상태머신 디자인

공통적으로 사용되는 코드를 프레임워크로부터 제공받아 프로그래머는 실제 응용 프로그램을 위한 코드에 집중 할 수 있는 장점이 있다.

4.2 상태머신 프레임워크의 Footprint와 RAM 사용량

센서 노드의 제한된 자원을 생각하면 프레임워크의 footprint와 RAM사용량은 매우 중요한 요소가 된다. 이를 측정하기 위해 상태머신 프레임워크의 footprint와 RAM 사용량을 측정하였다. 측정한 NanoQplus 커널 설정 다음과 같다.

```
#define ATMEGA128 1
#define NANO24 1
/* Basic Functions */
#define LED_M 1
#define UART_M 1
#define ADC_M 1
/* Platform Specific Devices and Functions */
#define SENSOR_LIGHT_M 1
#define SENSOR_STK1_M 1
#define SENSOR_TEMP_M 1
/* Kernel */
#define KERNEL_M 1
#define ENABLE_SCHEDULING 1
#define SCHED_PERIOD_10 1
#define SEM_M 1
#define MSGQ_M 1

/* Network Protocol */
#define MAC_M 1
#define NANO_MAC_M 1
#define RF_M 1
#define CC2420_M 1
#define ROUTING_M 1
#define RENO_M 1
```

프레임워크의 footprint를 측정하기 위한 방법은 컴파일 후 text영역을 디어셈블 하여(lst파일 이용) 각 주소별 차지하는 코드를 파악하여 상태머신 프레임워크가 차지하는 크기를 측정하였으며 결과는 <표 1>과 같다.

프레임워크의 코드 크기를 보면 1982byte로 전체 크기의 7.42%로 매우 작은 크기를 차지하고 있다. 커널에서 제공하는 메시지큐 구현의 크기가 664byte를 차지하는 것을 보면 큰 크기가 아님을 알 수 있다. 그리고 Atmega128에서는 프로그램 코드공간으로 128Kbyte를 사용 할 수 있기 때문에 복잡한 응용 프로그램을 포함하더라도 충분히 적용 가능한 크기이다.

프레임워크가 사용하는 RAM크기는 <표 2>에 정리 하였다. 프레임워크의 쓰레드에서 사용하는 스택크기와 전역변수로 선언한 메시지풀과 각 자료구조 데이터를 포함하였다. 동적으로 메모리를 할당 받는 부분이 없기 때문에 정적 분석만으로도 총 사용 크기를 알 수 있다. 사용되는 RAM크기는 상태머신의 설정에 따라 많이 달라지는데 다음과 같은 설정에서 사용하는 RAM크기를 측정하였다.

〈표 1〉 프레임워크 Footprint

전체	커널	프레임워크	응용프로그램
26700byte	24012byte	1982byte	706byte
100%	89.93%	7.42%	2.64%

〈표 2〉 RAM사용 Footprint

단위	전체크기	스택 공간	데이터 영역	남은 공간
byte	4096	200	176	3720
%	100%	4.88	4.30	90.82

```
실행모듈 쓰레드 개수 : 1
FSM모듈 최대 개수 : 3
상태머신 개수 : 2
메시지풀 크기 : 10
```

쓰레드 스택의 크기가 Atmega128에서는 기본으로 200byte로 설정되어 있고, 1개의 쓰레드만 사용되도록 정하였으므로 200byte를 스택용으로 사용되고 나머지 데이터 영역에 176byte의 RAM을 사용한다. Atmega128에서는 총 4K(4096byte)의 RAM을 사용할 수 있으며, 프레임워크에서 사용하는 376byte를 제외 하더라도 3720byte의 영역이 남아 있다.

4.3 센서네트워크 운영체제 비교

<표 3>은 TinyOS와 NanoQplus, 상태머신 프레임워크를 적용한 NanoQplus의 응용 프로그램 개발 시 제공되는 사양을 비교한 표이다. TinyOS는 컴포넌트에 기반한 이벤트 드리븐 방식을 사용하며, NanoQplus는 멀티 쓰레드 방법을 지원한다. 상태머신 프레임워크를 적용한 NanoQplus는 추가적으로 이벤트 드리븐 방식의 실행 모델을 지원한다. 응용프로그램을 시각적인 방법으로 표현하기위한 상위 레벨 디자인의 지원은 NanoQplus에서 NanoESTO를 통해 prototype 코드 생성만을 제공하고 디자인 방법은 제공하지 않는다. TinyOS는 Rhapsody[14]나 다른 CASE툴을 통해 시각적 설계를 지원한다. 상태머신 프레임워크는 본 논문에서 NanoESTO 플러그인으로 구현한 NanoCASE툴을 사용하여 상태머신을 시각적으로 설계할 수 있다. 비교를 통해 알 수 있듯이 상

〈표 3〉 센서네트워크 운영체제 비교

	실행모델	응용 프로그램의 상위 레벨 디자인 지원	관련도구/툴
TinyOS	컴포넌트베이스 이벤트 드리븐	Y	TOSSIM, TOSVIS, TinyDB, TinySEC Rhapsody[14]
NanoQplus	멀티 스레드	N (Prototype 코드만 제공)	NanoESTO, NanoMON, 원격업그레이더
NanoQplus with FSM Framework	멀티 스레드 이벤트 드리븐	Y	NanoCASE*

* 본 논문에서 구현한 CASE 툴

태머신 프레임 워크를 사용하여 응용프로그램 개발자에게 추가적인 개발 방법을 제공한다.

5. 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, 저장하는 방법을 제공하지 않으며, 하나의 스택을 공유하는 방식으로 동작하기 때문에 이벤트를 처리하는 핸들러에서 많은 처리를 하거나 블로킹을 가져올 수 있는 함수를 호출하지 못하는 등의 응용프로그램을 개발 시, 제한이 되는 조건이 많이 발생 한다. 그 밖에 일반 전통적인 유닉스 형태를 가지는 MANTIS[4]나 NanoQplus[6]는 프로그래머에게 친숙한 개발 환경을 가지고 있지만, 센서네트워크용 응용프로그램을 개발할 프레임워크가 부족하여 응용 프로그램을 작성하기 어려운 점이 많이 있다.

본 논문에서는 NanoQplus상에서 자원적 제한을 지키면서 응용프로그램을 상태머신 기반으로 개발할 때 기반이 되는 어플리케이션 프레임워크의 구조를 제안하고 구현하였다. 프레임워크에서 메시지를 공유하여 전달하는 방식을 통해 자원 절약 및 메시지 풀을 사용하여 메모리 단편화 문제를 해결하였고, 상태를 표현하고 전이 하는 방법에 대해서 API를 통해 제공하였다. 또한 멀티 쓰레드 기반에서 Run-To-Completion을 지키면서 메시지 핸들러에서 긴 지연을 허용하도록 하였다. 그리고 구현한 프레임워크를 좀 더 쉽게 사용 할 수 있도록 상태 차트를 통해 디자인한 상태머신을 프레임워크에서 사용할 수 있는 코드로 변환시키는 CASE툴을 구현하였다.

향후 과제로, 현재 상태 머신은 정적으로 운영체제에 포함되어 노드에 올라가는 상태이지만, 이를 상태 머신만 분리하여 동적으로 운영체제가 실행 중에 상태 머신만을 교체할 수 있도록 동적 상태 머신 교체 모듈 개발에 대한 연구가 필요하다. 동적으로 상태 머신을 교체할 경우 응용프로그램의 많은 부분을 차지하는 프레임워크 부분은 고정되고 로직을 결정하는 적은 양의 코드만 동적으로 교체가 가능하므로 센서 노드의 효율적인 동적 재구성이 가능할 것이다. 그 외 현재 구현한 상태 머신 프레임워크는 표준 상태머신만을 지원하고 있다. 복잡한 응용프로그램을 작성할 때는 이벤트에 대한 액션을 상속할 수 있는 개념이 들어간 계층형 상태 머신의 구조가 매우 유용하기 때문에 적절한 양의 메모리와 계산을 필요로 하는 계층형 상태머신에 대한 연구가 필요하다.

참 고 문 헌

[1] I.F.Akyildiz, W.Su,Sankarasubramaniam, E.Cayirci, "A Survey on Sensor Networks," IEEE Communications Magazine, pp.102-114, 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] NanoQplus, (Web Page) http://qplus.or.kr

[7] Harel, D.: Statecharts: A Visual Formalism for Complex Systems, The Science of Computer Programming, pp.231-274.

[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)

[10] Eclipse, (Web Page) http://www.eclipse.org

[11] Graphical Editing Framework, http://www.eclipse.org/ gef/

[12] T.-H. Kim, Seongsoo Hong, "State machine based operating system architecture for wireless sensor networks," Lecture Notes in Computer Science, Vol.3320 No. pp.803-803, Dec., 2004

[13] 송준근, 마평수 "IP-USN을 위한 센서네트워크운영체제 동향" 전자통신동향분석, 제23권 1호 2008년 2월 pp.12-20.

[14] Rhapsody http://modeling.telelogic.com/products/rhapsody/index.cfm

이 승 근

e-mail : sngn.lee@samsung.com

2006년 한양대학교 컴퓨터공학과(학사)

2008년 한양대학교 컴퓨터공학과(공학석사)

2008년~현 재 삼성전자 DMC연구소

관심분야 : Mobile S/W Platform, Linux Platform

김 병 곤

e-mail : bkkim@kict.re.kr

1991년 한양대학교 컴퓨터공학과(학사)

1993년 한양대학교 컴퓨터공학과(공학석사)

2003년 한양대학교 컴퓨터공학과(박사과정 수료)

1993년~현 재 한국건설기술연구원 건설정보연구실 수석연구원

관심분야 : 운영체제, 무선센서네트워크(WSN), 건설정보화

최 병 규

e-mail : mysaint@naver.com

2002년 2월 한양대학교 전자컴퓨터공학부 (학사)

2004년 2월 한양대학교 컴퓨터공학과(공학석사)

2004년~현 재 한양대학교 컴퓨터공학과 (박사과정 수료)

관심분야 : Sensor-Network, TMO(Time-triggered Message-triggered Object)

허 신

e-mail : shinheu@hanyang.ac.kr

1973년 서울대학교 전기공학과(학사)

1979년 미국 University of Southern California (전산학 석사)

1986년 미국 University of South Florida (전산학 박사)

1980년~1986년 미국 University of South Florida 연구원보

1986년~1988년 미국 The Catholic University of America 조교수

1988년~현 재 한양대학교 컴퓨터공학과 교수

관심분야 : 분산컴퓨팅, 결함허용시스템, 실시간운영체제등