센서 네트워크를 위한 실시간 운영체제 및 컴포넌트 모델 설계

맹지찬°, 김종혁, 유민수
한양대학교 정보통신대학원
msryu@hanyang.ac.kr, {jcmaeng, ntkjh}@ihanyang.ac.kr

Designing a Component-based Model and Real-Time Operating System for Networked Sensors

Ji Chan Maeng°, Jong-hyuk Kim, Minsoo Ryu
The Graduate School of Information and Communications, Hanyang University

요 약

최근 유비쿼터스 컴퓨팅의 핵심인 센서 네트워크에 많은 관심이 집중되고 있다. 센서 네트워크를 구성하는 요소중 센서노드에 탑재되는 운영체제의 경우, 제한된 컴퓨팅 자원을 가지는 센서 네트워크의 특성상 효율성, 초경량성, 실시간성, 병렬성, 이벤트 기반, 저전력, 재사용성, 프로그래밍 용이성, 그리고 이식성과 같은 요구사항들을 가진다. 본 논문에서는 이러한 요구사항들을 기반으로 (1) 유한상태기계(finite state machine)에 기반한 컴포넌트 소프트웨어 모델을 정의하고, (2) 이러한 컴포넌트를 효과적으로 수행시킬 수 있는 실시간 운영체제를 제안한다.

1. 서 론

센서 네트워크는 인간과 컴퓨터가 유기적으로 연계되어 다양하고 편리한 서비스를 제공해주는 유비쿼터스 컴퓨팅의 핵심으로서 최근 많은 관심이 집중되고 있다.

센서 네트워크를 구성하는 가장 중요한 요소인 센서노드는 마이크로 컨트롤러를 내장한 초소형 컴퓨터 시스템으로서 보통 저사양의 CPU와 메모리, 무선 통신 모듈, MEMs 센서 등을 사용한다. 이에 따라 센서노드에서 가용할 수 있는 컴퓨팅 자원(computing resources)과 배터리 전력은 극히 제한적이므로 효율적이고도 초경량의 소프트웨어의 탑재가 필수적이다. 아울러, 대부분의 센서 네트워크 응용에서 데이터 감지/프로세싱/전달의 실시간 처리를 요구함에 따라, 센서 노드에 내장되는 소프트웨어는 실시간성을 필수적으로 보장해 줄 수 있어야 한다. 그 외에 병렬성(concurrncy), 이벤트 기반(event-driven), 저전력(low-power), 재사용성(reusability), 프로그래밍 용이성, 이식성(*portability*) 등이 센서 네트워크에 탑재되는 소프트웨어의 중요한 요구사항이다.[1]

최근 미국 국방성의 DARPA 프로젝트의 일환으로 버클리대학에서는 Mote 센서에서 수행되는 TinyOS[4] 운영체제와 nesC[3]라는 프로그래밍 언어를 개발한 바 있다. TinyOS[4]상에서 수행되는 응용 소프트웨어는 nesC[3]라는 프로그래밍 언어를 통해 병렬적(concurrent)으로 수행될 수 있는 컴포넌트의 집합으로 기술되며, TinyOS[4]는 응용 소프트웨어의 컴포넌트를 수행시키는 서비스를 제공한다. TinyOS[4]는 센서 네트워크 응용의 event-driven 특성을 충실히 반영하고, 컴포넌트 기반의 병렬성을 제공함에 따라 다수의 상용 센서와 연구 목적으로 널리 사용되고 있다. 그러나, TinyOS[4]의 개발진들이 밝히듯이 TinyOS[4]는 실시간성에 대한 고려가 없으며, nesC[3] 프로그래밍 언어의 사용으로 인해 재사용성과 프로그래밍 용이성이 부족하다는 단점을 가진다.

본 논문에서는 유한상태기계(finite state machine)에 기반한 컴포넌트 소프트웨어 모델을 정의하고, 이러한 컴포넌트를 효과적으로 수행시킬 수 있는 실시간 운영체제를 제안하고자 한다.

2. 컴포넌트 모델

각각의 센서에서 수행되는 응용 소프트웨어의 병렬성 및 재사용성을 극대화하기 위하여 센서 네트워크 응용에 적합한 새로운 컴포넌트 소프트웨어 모델을 정의한다. 정의될 컴포넌트는 기존의 UML(Unified Modeling Language)[9]과 ROOM(Real-Time Object Oriented Modeling)[10]의 오브젝트 모델에 기반함에 따라, 응용 프로그래머가 Rose와 같은 상용 소프트웨어를 사용하여 소프트웨어를 용이하게 설계할 수 있으며, 또한 자동코드 생성(automatic code generation) 기능을 통해 소프트웨어를 쉽게 구현할 수도 있다.

2.1 컴포넌트 정의 및 아키텍쳐

응용 소프트웨어는 다수의 컴포넌트로 구성되며, 각 컴포넌트는 최소한 한 개 이상의 독립적인 쓰레드 제어권(thread of control)을 가지는 동시(concurrent)에 실행 가능한 수행 단위로 정의할 수 있으며, 컴포넌트는 새로운 응용 소프트웨어 개발에 재사용(reuse)될 수 있는 단위로도 사용된다. 컴포넌트는 아래와 같이 두 가지 측면의 아키텍쳐로 정의된다.

- **구조적 측면(structural aspect)**: 컴포넌트는 자신의 상태(state)를 표현하는 속성(attributes), 자신 또는 다른 컴포넌트에 의해 수행될 수 있는 오퍼레이션(operations, 또는 function), 다른 컴포넌트가 보내는 메시지를 받을 메시지 큐(message queue)를 가진다.
- **행위적 측면(behavioral aspect)**: 컴포넌트 동적인 행동양식

(dynamic behavior)는 statechart로 표현된다.

2.2 컴포넌트 관계(relationship)

컴포넌트간에는 다음의 두 가지 관계가 존재한다. 그중 하나는 집합 관계(aggregation, "has" relationship)로, 이는 하나의 컴포넌트가 여러 개의 컴포넌트로 구성될 때, 상위 컴포넌트가 하위 컴포넌트를 포함(has)하는 관계를 말하며, 다른 하나는 사용 관계(association, "uses" relationship)로 하나의 컴포넌트 A가 다른 컴포넌트 B의 오퍼레이션(operation)을 호출할 때, A가 B를 사용(uses)하는 관계로 정의한다.

2.3 컴포넌트 오퍼레이션

컴포넌트 오퍼레이션은 컴포넌트간에 또는 컴포넌트 자체의 동작을 말하며, 외부 및 내부 오퍼레이션으로 나누어진다. 외부 오퍼레이션(external operations)은 다른 컴포넌트에 의해 호출될 수 있는 외부 인터페이스 함수로 정의된다. 외부 오퍼레이션의 수행은 호출 컴포넌트(caller component)의 문맥(context)에서 수행되며, 오퍼레이션을 제공하는 컴포넌트의 상태머신(state machine)에는 영향을 주지 않는다. 반면에, 내부 오퍼레이션(internal operations)은 상태 머신의 이벤트 처리, 상태 전이, 액션을 수행할 때 커널에 의해서만 호출되는 함수로 정의된다.

2.4 상태기계 모델

일반적으로 유한상태기계는 <I, O, S, S0, Δ, Γ>로 정의되는데, I는 유한개의 입력 이벤트 집합을, O는 유한개의 출력 액션 집합을, S는 유한개의 상태 집합을, S0 ∈ S는 초기 상태를, Δ:S * I -> S는 상태 전이 함수를, 그리고 Γ:S * I -> O는 출력 함수를 지칭한다. 보통 유한상태머신은 상태 전이 다이어그램(state transition diagram)으로 정의되기도 하는데, 본 논문에서는 계층성(hierarchy)과 직교성(orthogonality) 개념을 도입한 David Harel의 상태챠트(statechart) 모델[7]을 사용한다. 계층성은 상태 안에 상태가 내포되어(nested) 계층적 상태 분할을 의미하며, 이 때 외곽의 상태를 상위 상태(superstate), 내곽의 상태를 하위 상태(substate)라고 정의한다. 직교성은 하나의 상태챠트 안에 두 개 이상의 독립적인 상태챠트를 허용하는데, 이는 UML에서의 'AND-states'로 알려진 개념이기도 하다.

2.5 컴포넌트 기반 응용 소프트웨어 명세 기술

컴포넌트 기반의 응용 소프트웨어 명세를 위한 기술로는 다음의 두가지를 사용한다.

- **Use Case를 이용한 응용 시나리오 기술**: Use Case는 시스템 내부 사이에서 교환되는 메시지의 흐름과, 외부 사용자(또는 시스템)와 내부 시스템의 상호 작용에 의해서 수행되는 행동을 표현하는 방식으로서, 시스템에 의해 수행되는 각각의 처리 자체를 Use Case라 하며, 시스템으로부터 정보를 받거나 혹은 시스템으로 정보를 보내는 역할을 하는 외부 요소를 Actor로 표현한다. 이를 이용, 해당 응용 소프트웨어가 실제로 구현되었을 시, 동작하게 되는 시나리오를 기술함으로서, 해당 응용 소프트웨어가 구현해야하는 기능과 고려해야하는 요소들을 분석하는 동시에, 추후 설계 및 구현시의 기본 자료를 제공한다. 본 논문에서는 UML에서 정의하고 있는 표현 기준과 방식을 따른다.
- **Sequence Diagram을 이용한 컴포넌트의 동적 관계 기술**: Sequence Diagram은 시간에 따른 메시지 발생 순서를 강조하는 그림으로써, 교류를 주도하는 컴포넌트를 왼쪽에 놓고 그 오른쪽으로는 각 컴포넌트들을 배치시키며, 메시지들을 시간의 흐름에 따라 위에서 아래로 세로축에 따라 배치하여 나타낸다. 또한 Lifeline을 이용하여 특정 시간(기간)동안 객체가 존재하는 것을 표현할 수 있으며, 제어 초점(Focus of Control)을 두어 객체가 활동하는 시간대를 보인다. 여기서 순차도 한 개는 제어 흐름 한 개만을 표현한다. 이를 통해, 컴포넌트들과 그들 간의 관계로 구성된 교류를 표현하는 동시에, 각 컴포넌트들의 행동을 설명하는 시나리오 문맥에서 구성 요소 간에 전달되는 메시지를 나타낼 수 있다.

3. 센서 네트워크를 위한 실시간 운영체제

본 논문에서는 유한상태기계의 이벤트 처리와 상태전이(state transition)에 필요한 최소한의 실시간 서비스만을 제공할 수 있는 실시간 운영체제를 설계한다. 이는 유한상태기계로 표현되는 응용 소프트웨어의 이벤트 핸들링, 상태전이, 상태전이에 수반되는 액션(action)의 수행을 담당하고, 디바이스 드라이버를 포함하는 전통적인 OS 서비스는 사용자 공간(user address space)상에서 실행되도록 한다. 따라서, 개발될 실시간 커널은 수 KB 이만의 메모리만으로도 충분히 실행가능하며, 실시간 스케쥴링 및 실시간 동기화 기법의 제공으로 센서 네트워크 응용의 실시간성을 극대화할 수 있는 장점을 함께 제공한다. 본 논문에서 제안하는 실시간 운영체제의 특징들은 다음과 같다.

- **마이크로커널(icrokernel) 구조의 초경량 RTOS**: 커널에서는 인터럽트 처리, 타이머, IPC(inter-process communication), 스케쥴링, 동기화(synchronization) 등의 서비스만을 제공한다. 파일시스템, 가상메모리의 사용을 배제하며, 응용 소프트웨어별로 사용되는 다양한 디바이스 드라이버는 사용자 공간(user address space)에서 수행. 네트워크 프로토콜 처리도 사용자 공간에서 수행한다. 최종 커널은 1 KB 미만의 코드 사이즈 및 메모리 사용량을 가진다.

  다양한 아키텍쳐로의 이식(porting)을 용이하도록 하기위한 최소한의 HAL(hardware abstraction layer) 개념을 도입한다. 하드웨어에 의존적인 부분과 비의존적인 부분을 구분하기위한 계층 구조 및 HAL을 정의한다.
- **상태기계와 이벤트 처리 중심의 멀티쓰레드 커널**: 응용 소프트웨어를 다수의 상태챠트로 기술한다. 상태챠트로 기술된 각각의 쓰레드를 이벤트 구동(event-driven) 방식으로 수행할 수 있도록 하는 상태기계 엔진(state machine engine)이 커널의 핵심 요소가 되며, 여기에서는 발생한 이벤트를 해당 쓰레드로 디스패치하는 이벤트 디멀티플렉싱(event demultiplexing to destination thread), 응용 쓰레드의 상태전이, 상태전이에 수반되는 액션 수행 등을 처리한다.
- **실시간성 보장**: 우선순위 기반의 이벤트 큐(priority-based event queue)를 이용하며, 이벤트 큐에 EDF(earliest deadline

first) 및 RM(rate monotonic) 등의 동적/정적 우선순위를 적용하여 응용 쓰레드의 실시간 스케쥴링을 가능하게 한다. 또한, PIP(Priority Inheritance Protocol) 및 PCP(Priority Ceiling Protocol)[11] 서비스를 제공하고, 이에 기반한 뮤텍스(mutex)를 사용하여 실시간 동기화 및 예측성(predictability)을 제공한다.

- **센서 네트워크 응용에 필요한 최소한의 응용 프로그래밍 인터페이스 제공**: 타이머, IPC, 동기화 등을 위한 시스템 콜(system call)을 제공하며, 사용자 공간에서의 디바이스 드라이버 작성을 위한 HAL API를 정의한다.

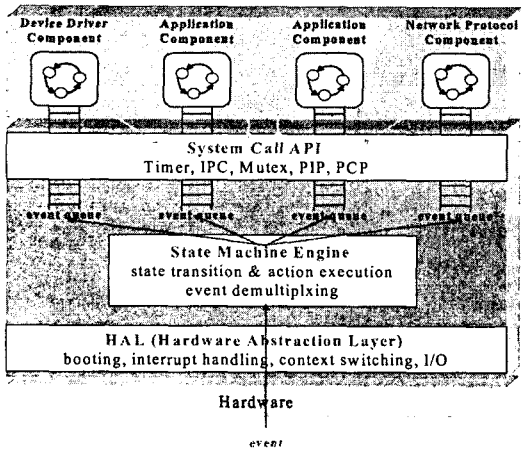

그림 1. 실시간 운영체제 구조

4. 결 론

버클리 대학에서 개발된 TinyOS[4]와 nesC[3]에 기반한 센서 네트워크 소프트웨어 개발은 컴포넌트 모델의 한계성으로 인해 실시간성을 보장하기 어려움은 물론, 재활용성 및 프로그래밍 용이성 측면에서 만족할 만한 성과를 보여주기 어렵다.

본 논문에서는 최근 SenOS[5]에서 소개된 개념을 확장하여, 이러한 문제를 극복하고자 한다. 응용 소프트웨어 프로그래밍상의 재사용 및 프로그래밍 용이성을 위해 컴포넌트 기반 소프트웨어 모델을 제시한다. 각 컴포넌트 간의 관계 및 오퍼레이션을 정의하고, 상태기계 모델과 컴포넌트 명세 기술을 제시한다. 또한 컴포넌트 기반 소프트웨어 모델을 지원하는 실시간 운영체제로 마이크로커널 기반의 새로운 RTOS를 제안한다. 이 운영체제는 상태기계 엔진을 핵심 요소로 가지며, 우선순위 기반의 이벤트 큐와 실시간 스케쥴링 기법들을 적용하여 운영체제의 실시간성을 보장한다.

5. 참고문헌

[1] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, "System Architecture Directions for Networked Sensors", in *9th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 93--104, November 2000.

[2] John Regehr, Alastair Reid, Kirk Webb, Michael Parker, and Jay Lepreau, "Evolving Real-time Systems Using Hierarchical Scheduling and Concurrency analysis", in *Proceedings of the 24th IEEE Real-Time System Symposium (RTSS 2003)*, December 2003.

[3] David Gay, Philip Levis, Revert von Behren, Matt Welsh, Eric Brewer, and David Culler, "The nesC language: A Holistic Approach to Networked Embedded Systems", in *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, 2003, San Diego, California, USA

[4] David E. Culler, Jason Hill, Philip Buonadonna, Robert Szewczyk, and Alec Woo, "A Network-Centric Approach to Embedded Software for Tiny Devices", in *Proc. EMSOFT'01, volume 2211 of LNCS*, pages 114-130, Springer-Verlag, 2001.

[5] Seongsoo Hong, Tae-Hyung Kim, "SenOS: State-Driven Operating System Architecture for Dynamic Sensor Node Reconfigurability", in *The First International Conference on Ubiquitous Computing*, October, 2003.

[6] C. Lui, J. Layland, "Scheduling Algorithm for Multiprogramming in a Hard Real-Time Environment", in *Journal of the ACM*, Vol. 20, No. 1, pages 46-61, January 1973.

[7] D. Harel, "Statecharts: A Visual Formalism for Complex Systems", in *Science of Computer Programming*, volume 8, number 3, pages 231--274, June, 1987

[8] Hassan Gomma, "Designing Concurrent, Distributed, and Real-Time Applications with UML", Addison Wesley, 2000.

[9] OMG Unified Modeling Language Specification, Version 1.5, 2003.

[10] Bran Selic, Garth Gullekson, and Paul T. Ward, "Real-Time Object Oriented Modeling", Wiley, 1994.

[11] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization", in *IEEE Transactions on Software Engineering*, Volume 39, Pages 1175 - 1185, September, 1990

[12] M. Saksena, P. Freedman, and P. Rodziewicz, "Guidelines for Automated Implementation of Executable Object Oriented Models for Real-Time Embedded Control Systems", in *Proceedings of the 18th IEEE Real-Time Systems Symposium (RTSS '97)*, December 03-05, 1997.

[13] M. Saksena, A. Ptak, P. Freedman, and P. Rodziewicz, "Schedulability Analysis for Automated Implementations of Real-Time Object-Oriented Models", in *Proceedings of the IEEE Real-Time Systems Symposium*, 1998.