State Machine Based Operating System Architecture for Wireless Sensor Networks

Tae-Hyung Kim[1] and Seongsoo Hong[2]

[1] Department of Computer Science and Engineering, Hanyang University, Ansan, Kyunggi-Do, 426-791, South Korea
tkim@cse.hanyang.ac.kr

[2] School of Electrical Engineering and Computer Science, Seoul National University, Seoul 151-741, South Korea
sshong@redwood.snu.ac.kr

Abstract. A wireless sensor network is characterized as a massively distributed and deeply embedded system. Such a system requires concurrent and asynchronous event handling as a distributed system and resource-consciousness as an embedded system. State machine based software design techniques are capable of satisfying exactly these requirements. In this paper, we present how to design a compact and efficient operating system for wireless sensor nodes based on a finite state machine. We describe how this operating system can operate in an extremely resource constrained sensor node while providing the required concurrency, reactivity, and reconfigurability. We also show some important benefits implied by this architecture.

1 Introduction

Sensor networks consist of a set of sensor nodes, each equipped with one or more sensing units, a wireless communicating unit, and a local processing unit with small memory footprint [1]. In recent advancement of wireless communication and embedded system technologies, the wireless and distributed sensor networks become a prime technical enabler that can provide a way of noble linkage between the computational and the physical worlds. Since the precise delivery of real-time data on the spot is an essential basis for constructing a context-aware computing platform, the recent advancement of low-cost sensor node provides an important opportunity towards the new realm of ubiquitous computing. Positioned at the very end-terminal from the computational world side, wireless sensor nodes convey unique technical challenges and constraints that are unavoidable to system developers, which can be characterized by three aspects. First, they bear extremely limited resources including computing power, memory, and supplied electric power. Nonetheless, a sensor network can be perceived as a traditional distributed computing platform consisting of tens of thousands of autonomously cooperating nodes. Third, the computing platform does not allow recycling of the network, thus is disposable without having re-programmability.

© Springer-Verlag Berlin Heidelberg 2004

Such characteristics of a networked sensor node call for a unique operating system architecture that can not only run on an extremely lightweight device with very low power consumption but can also support dynamic reconfigurability to cope with changing environments and applications. Such an operating system should also possess concurrent and asynchronous event handling capabilities and support distributed and data-centric programming models. In order to meet such seemingly contradictory requirements, we propose a state machine based operating system architecture, rather than following a traditional structure of an operating system and adopting it for sensor nodes like TinyOS [2]. To provide re-programmability, TinyOS employs the bytecode interpreter called Maté that runs on it. In a state machine based operating system like ours, each node is allowed to simply reload a new state machine table. Moreover, the state machine based software modeling offers a number of benefits: (1) it enables designers to easily capture a design model and automatically synthesize runtime code through widely available code generation tools; (2) it allows for controlled concurrency and reactivity that are needed to handle input events; and (3) it enables a runtime system to efficiently stop and resume a program since the states are clearly defined in a state machine. In this paper, we explore a state machine based execution model as an ideal operating system design for a networked sensor node and present the end result named SenOS.

2 State Machine Based Execution Environment

While many embedded applications should exhibit a reactive behavior, dealing with such reactivity is considered to be the most problematic. To cope with the complexity of designing such systems, Harel introduced a visual formalism referred to as statecharts [3]. Since then, a state machine has been recognized as a powerful modeling tool for reactive and control-driven embedded applications. Sensor network applications are one of those applications that can mechanize a sequence of actions, and handle discrete inputs and outputs differently according to its operating modes. Being in a state implies that a system reacts only to a predefined set of legal inputs, produces a subset of all possible outputs after performing a given function, and changes its state immediately in a mechanical way. Formally, a finite state machine is described by a finite set of inputs, outputs, states, a state transition function, an output function, and an initial state. When a finite state machine is implemented, a valid input (or event) triggers a state transition and output generation, which moves the machine from the current state to another state. A state transition takes place instantaneously and an output function associated with the state transition is invoked.

A state machine based program environment is not only suitable for modeling sensor network applications but also can be implemented in an efficient and concise way. Since sensor node functionalities are limited, although multi-functional, all those possible node functionalities are defined statically in a callback function library in advance. All we need to do as a programmer is simply to define a legal sequence of actions in tabular forms. To this end, SenOS has four system-level components: (1) an event queue that stores inputs in a FIFO order, (2) a state sequencer that accepts an input from the event queue, (3) a callback function library that defines output functions, and (4) a re-loadable state transition table that defines each valid state transition