SenOS: State-driven Operating System Architecture for Dynamic Sensor Node Reconfigurability*

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

[1] School of Electrical Engineering and Computer Science
Seoul National University, Seoul 151-742, Korea
sshong@redwood.snu.ac.kr
[2] Department of Computer Science and Engineering
Hanyang University
Ansan, Kyunggi-Do, 426-791, Korea
tkim@cse.hanyang.ac.kr

Abstract. The operating environment and architecture of wireless sensor networks pose very unique design challenges and constraints to developers. Each node in a wireless sensor network must operate with extremely limited hardware resources yet possess wireless communication capability and support concurrency and reactive event processing. Also, it should be dynamically reconfigured at runtime to react to changes in environment and applications. In designing a runtime environment for wireless sensor nodes, it is thus essential to provide an operating system architecture that can meet these seemingly contradictory design requirements and constraints. In this paper, we present SenOS, an efficient and effective finite state machine-based operating system for wireless sensor nodes. We show that SenOS can operate in an extremely resource constrained sensor node while providing the desired concurrency, reactivity, and reconfigurability.

1 Introduction

Wireless sensor networks have emerged as one of the key enabling technologies for ubiquitous computing since wireless intelligent sensors connected by short range RF communication media serve as a smart intermediary between objects and people in a ubiquitous computing environment [1]. Unlike conventional computing platforms, wireless sensor nodes are characterized by (1) extremely limited resources including computing power, memory, and supplied electric power, (2) data centric programming styles that view a sensor network as a distributed computing platform consisting of tens of thousands of autonomously cooperating nodes, and (3) a disposable computing platform that does not allow recycling of the network. These 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 asynchro-

* The work reported in this paper is supported in part by MOST (Ministry of Science and Technology) under the National Research Laboratory (NRL) grant 2000-N-NL-01-C-136.

nous event handling capabilities and support distributed, data-centric programming models with the aid of middleware. It seems almost impossible to design and implement an operating system to meet such seemingly contradictory requirements.

Fortunately, networked sensor node applications are naturally well suited for a state machine based computing model in that a program sequences a series of actions or handles input events differently depending on the mode of the program. 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 of a state machine, which are to be store and restored during preemption, are clearly defined. Consequently, a state machine based program can be transformed into very compact and efficient executable code when implemented.

In this paper, we exploit a finite state machine based execution model to design an ideal operating system for a networked sensor node and present the end result we name SenOS. It is designed to meet the constraints in code size and power consumption while providing the desired concurrency, reactivity, and runtime reconfigurability. In the remainder of the paper, we present the architecture of SenOS and show how the desired capabilities are realized by SenOS.

2 FSM-driven Program Execution Environment

A finite state machine is a mathematical model of a system, with discrete inputs and outputs. It can be in any one of a finite number of internal states where a state denotes its input history from the initial state. More formally, a finite state machine is described by (1) a finite set of states, (2) a finite set of inputs, (3) a finite set of outputs, (4) a state transition function mapping states and inputs to next states, (5) an output function mapping states and inputs to outputs, and (6) an initial state. This describes a Mealy machine where the outputs are a function of both a current state and an input, as opposed to a Moore machine where the outputs are a function of only a state [3].

When our 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. Such a state transition takes place instantaneously and an output function associated with the state transition is invoked. Using this execution mechanism, a finite state machine sequences a series of actions or handles input events differently depending on the state of the machine.

To implement a finite state machine, we need four components: (1) a state sequencer that accepts an input from an event queue, (2) an event queue that stores inputs in a FIFO order, (3) a callback function library that contains output functions, and (4) a state transition table that defines each__valid state transition and its associated call-

back function. Each callback function should satisfy the "run-to-completion" semantics to maintain the instantaneous state transition semantics.

Many embedded real-time systems are naturally amenable to the finite state machine model and so are networked sensor applications. There exist quite a few CASE tools that help designers capture state machine based system models and automatically synthesize executable code for them. UML-RT is one such tool widely used in the embedded systems industry [4]. SenOS application programmers can take advantage of these tools for developing networked sensor applications.

3 State-driven OS Architecture

The SenOS kernel architecture is based on the finite state machine model. It is comprised of three components: (1) the Kernel consisting of a state sequencer and an event queue, (2) a state transition table, and (3) a callback library. Fig. 1 pictorially depicts the SenOS kernel. The Kernel continuously checks the event queue for event arrivals; if there are one or more inputs in the queue, it takes the first one out of the queue and triggers a state transition if the input is valid. It then invokes an output function associated with the state transition. To do so, the Kernel keeps track of the state of the machine and guards the execution of a callback function with a mutex that can guarantee the run-to-completion semantics.

The call back library provides a set of built-in functions for application programmers, thus determining the capability of a sensor node. The Kernel and callback library should be statically built and stored in the flash ROM of a sensor node whereas the state transition table can be reloaded or modified at runtime. The SenOS can host multiple applications by means of multiple co-existing state transition tables and provide concurrency among applications by switching state transition tables. Note that each state transition table defines an application. During preemption, the Kernel saves the present state of the current application, restores the state of the next application, and changes the current state transition table.

The SenOS architecture also contains a runtime monitor that serves as a dynamic application loader. As shown in Fig. 1, it contains an interrupt filter that is in fact a generic interrupt service handler invoked upon every external interrupt. When the SenOS receives an application reload message via an interrupt from a communication adapter, the Monitor puts the Kernel into a safe state, stops the Kernel, and reloads a new state transition table. Note that the Monitor is allowed to interrupt the Kernel anytime unless it is in state transition. Since state transition is guarded by a mutex, the safety of a finite state machine is not compromised by such an interruption.

It is obvious that SenOS provides the desired concurrency, synchronization, reactivity, and runtime reconfigurability.

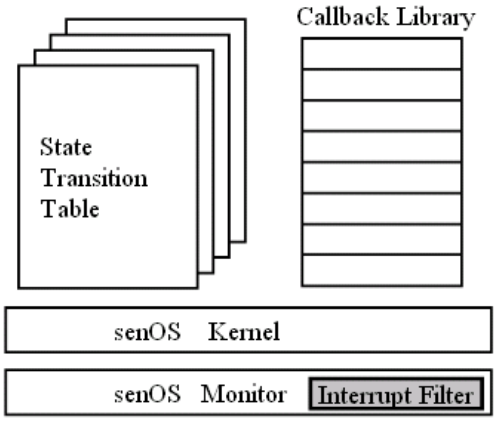

Fig. 1. SenOS Kernel Architecture and its Components.

4 Conclusion

We have presented SenOS, a state machine based operating system for a networked sensor node. It consists of the Kernel, Monitor, and replaceable state transition tables and call back libraries. Programmers can easily write a SenOS application and derive executable code using a SenOS design tool and load the executable code at runtime using the Monitor as an agent. SenOS offers a number of benefits. Its implementation is very compact and efficient since it is based on a state machine model. Also, it supports dynamic node reconfigurabillity in a very effective manner using replaceable state transition tables and callback libraries. These benefits render SenOS ideal for a networked sensor node. Currently, the implementation of SenOS and associated tools are underway and the result looks promising.

References

1. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, System Architecture Directions for Networked Sensors. International Conference on Architectural Support for Programming Languages and Operating Systems (2000)
2. P. Levis and D. Culler, Mate: A Tiny Virtual Machine for Sensor Networks. International Conference on Architectural Support for Programming Languages and Operating Systems (2002).
3. J. Hopcroft and J. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison Wesley (1979).
4. IBM (former Rational Software), http://www.rational.com.