Overview: Chapter 7

- Sensor node platforms must contend with many issues
  - Energy consumption
  - Sensing environment
  - Networking
  - Real-time constraints
- Not typical distributed system application
- Programming Models
  - Programs by end users
    - Encode application logic
    - Abstract details from users
  - Programs by application developers
    - Expose data acquisition and hardware interface to developers

Sensor Node Hardware

- Categories
  - Augmented general purpose
    - PC104, Sensoria WINS NG, PDAs
    - Commercial off-the-shelf (COTS) OS & components: Wndows CE, Linux, Bluetooth, IEEE 802.11
  - Dedicated embedded sensor nodes
    - Berkeley motes, UCLA Medusa, Ember nodes, MIT μAMP
    - COTS chip sets (small form factors), programming languages(e.g., C)
  - System-on-chip (SoC)
    - Smart dust, BWRC picoradio nodes, PASTA nodes

Berkeley Motes

- Limited memory
  - Program memory: 8 - 128 KB
  - RAM: 0.5 - 4 KB
  - External storage (Flash): 32 - 512 KB
- Limited Communication
  - Max 50 kbps (with hardware acccel.)
- Energy savings is important
  - 12 mA to transmit data

Programming Challenges

- Traditional programming models not suitable
  - Programmer must handle with messaging, networking, event synch., interrupts etc.
  - Embedded OS (if any) expose hardware details to programmer
- Distributed algorithms/data structures difficult to implement
- Respond to multiple stimuli quickly

Software Platforms: TinyOS

- Targeted for resource constrained platforms (motes)
- Small memory footprint
  - No filesystem
  - Only static memory allocation
- Software made up of *components*
- Components create *tasks* and added to task scheduler
  - Tasks run to completion: no preempting by other tasks
- Events: interrupts from hardware
  - Run to completion, can preempt tasks

Software Platforms: nesC

- Extension of C for TinyOS
- Components
  - Interface
    - Defines what functionality component uses and provides
  - Implementations
    - Modules: written in C-like syntax
    - Configurations: connect interfaces of existing components
  - Cuncurrency
    - Asynchronous code (AC) vs. Synchronous code (SC)
    - SC atomic w.r.t. other SC
    - Programmers must understand concurrency issues in code

Software Platforms: TinyGALS

- Dataflow language
  - Programming model
    - Supports all TinyOS components
    - Construct asynch. actors from synch. components
    - Construct application by connecting asynch. components through FIFO queues
- Code Generation
  - Map high-level constructs to low-level code for motes
  - Automatically generate code for scheduling, event handling, FIFO queues

Node-Level Simulators

- Sensor node model
  - Mobility of nodes
  - Energy consumption
- Communication model
  - Capture details of communication (RF propagation delay, MAC layer etc.)
- Physical environmental model
  - Model physical phenomena in operating environment
- Statistics and visualization
  - Collect results for analysis

Node-Level Simulator: ns-2 & TOSSIM

- ns-2
  - Originally developed for wired networks
  - Extensions for sensor nodes
    - Node locations vs. logical addresses
    - Energy models
    - Physical phenomena
- TOSSIM
  - Simulator for TinyOS apps on Berkeley motes
  - Compiles nesC source into simulator components

State-Centric Programming

- Applications more than simple distributed programs
  - Applications depend on state of physical environment
- Collaboration Groups
  - Set of entities that contribute to state updates
  - Abstracts network topology and communication protocols
- Multi-target tracking problem
  - Global state decoupled into separate pieces
    - Each piece managed by different principal
    - State updated by looking at inputs from other principals
    - Collaboration groups define communication and roles of each principal