Introduction: Mote Networks are Too Hard to Program

Want reusable application services in TinyOS

- Want to port knowledge from one app to the next
- Speed up development
- Found application services don’t exist

- System components shared often, application components rarely
- So why is it so hard?
- Tight coupling requirements
- Resource constraints (can’t afford inefficiency, abstraction boundaries)
- Can we solve this problem with a programming methodology?

Want a software architecture to directly and easily control fidelity/energy tradeoffs

- Data reduction via filtering and compression on a mote directly results in energy savings
- Without clear models, scientists must request all sampled data because they can’t afford information loss
  - Incorrect thresholds, filtering parameters, degrees of compression on motes may discards vital information
  - Could knobs be provided that shift data processing and reduction from back-end PCs to motes?

Problem Description: We need Reusable Services that Cooperate Across Nodes

Language, Compiler, and Library

- Independently written library services must be able to cooperate to configure and share underlying resources
- Language support must be provided to specify constraints on the context in which an application service is designed to run
- Solving the constraint system imposed by all services in an application requires a compiler

Fidelity Control in High Data-Rate Applications

<table>
<thead>
<tr>
<th>Rate Control</th>
<th>Gate</th>
<th>FIR</th>
<th>Histogram</th>
<th>Waveform Characterization</th>
<th>ADPC</th>
<th>Entropy Coder</th>
</tr>
</thead>
<tbody>
<tr>
<td>enabled on motes</td>
<td>enabled on motes and pc</td>
<td>enabled on pc</td>
<td>disabled on pc</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SNACK: Language, Compiler, and Library Applied to Acoustics and Neural Sensing

Language Design

- Controlled sharing
  - A programmer specifies constraints on how components may be shared
  - The compiler shares identical components whenever possible:
    \[
    \text{a} : \text{Network} \rightarrow \text{a} : \text{Network}
    \]
  - to the equivalent of:
    \[
    \text{a} : \text{Network} \rightarrow \text{n} : \text{Network} \rightarrow \text{a} : \text{Network}
    \]
- Parameters
  - Components and services can take a list of named parameters
    \[
    \text{service SmoothSenseTemp}\{\text{period}: \text{max uint32_t } p = 1000\}
    \]
  - And these parameters can be bound:
    \[
    \text{mean : SmoothSense}\{\text{period} \geq 20, \text{period} \leq 40\}
    \]
- Connectedness constraints
  - Restrictions such as Sense (exactly once), Sensor (at least once), and Event (at most once) affect how interfaces may be used:
    \[
    \begin{align*}
    &\text{i} \rightarrow \text{TreeBuilder} [\text{Put32} \text{mutex}] \rightarrow \text{Sensor} \\
    \text{and these restrictions affect sharing:} \\
    &\text{TreeBuilder} \rightarrow \text{Sensor} \\
    &\text{Sensor} \rightarrow \text{TreeBuilder}
    \end{align*}
    \]
- Exclusive instantiation
  - Sharing may be prevented with the type constraint “my”

These SNACKs have compatible parameters:

\[
\begin{align*}
\text{service SmoothSenseTemp} &\{ \text{mean} \rightarrow \text{mean}, \text{period} \rightarrow \text{period} \} \\
\text{Sensor1} &\{ \text{Sensor1} \rightarrow \text{Sensor1} \} \\
\text{Sensor2} &\{ \text{Sensor2} \rightarrow \text{Sensor2} \}
\end{align*}
\]

But Sensor1 and Sensor2 will each get a unique SNACK instance

- Service weaving
  - Transitive arrows express non-local relationships:
    \[
    \begin{align*}
    &\text{service TreeBuilder}\{\text{mean} \rightarrow \text{mean}, \text{period} \rightarrow \text{period} \} \\
    &\text{Sensor1} \rightarrow \text{TreeBuilder} \rightarrow \text{Sensor2}
    \end{align*}
    \]

待遇 must be provided to specify constraints on the context in which an application service is designed to run

- Solving the constraint system imposed by all services in an application requires a compiler

Compiler Operation

- SNACK’s compiler reduces the recursive expansion of an application’s services to a maximally shared expansion that has the minimum number of instances of any valid expansion
- The maximally shared expansion is found by exhaustive search and must pass four tests:
  - Interface constraints: No bound or bound interface may be directly connected more than once
  - Parameters: The linear system of all component and service parameters has a solution
  - “my” constraints: Two components of the same type may only be shared if recursively imposed “my” constraints do not preclude it
- Transitive constraints: A valid assignment of transitive connections exists

Heuristics reduce the search space:

- Search is ordered by cardinality of expansion so the first valid expansion is the smallest
- Obviously incorrect expansions eliminated from consideration
- For each component type the smallest expansion subset that contains no obvious problems is memoized
- When an invalid expansion is found, the search is advanced to the next expansion that might affect the problem

Performance

- Comparison of two versions of a simple SNACK data forwarder with Surge, a similar nesC application
  - SF uses 10% more ROM than Surge, but 3% less RAM
  - All values are well within the Mica2’s operating constraints

- Bytes sent as a function of network size
  - 600-second simulations at varying network sizes
  - Transitive arrows let unrelated application components share packets so fewer bytes are transmitted

Audio Testbed

- 3-axis accelerometer
- 16-bit ADC

SNACK Software Configuration

Software Architecture (Telos B)

- Application
  - Audio
  - Waveform Characterization
  - Gate
  - Sample Size
  - Buffer Pool
  - DMA
  - ADC

- Control
  - SNACK Software
  - CPU
  - USB

- Interfaces
  - Network
  - Device

- Receiver
  - Reliability
  - SensiMk

- Encoder
  - SampledGate

- Sample Set
  - Double Buffer

- Ready
  - Sensor

- Power
  - Processor

- Processor
  - Vendor

- Firmware
  - Waveform Characterization

Future Work:

Wireless Neural Sensing*

* A collaboration with UCLA NeuroEngineering, Shahin Farshchi, Aleskeys Pesterev, Paul Nuyujukian, Neschie Fernando, and Jack W. Judy