μSETL: A Set Based Programming Abstraction for Wireless Sensor Networks

Author: Mohammad Sajjad Hossain, A.B.M. Alim Al Islam, Milind Kulkarni, Vijay Raghunathan

By: Yi Shao
Background

- **Abstraction:** programming specification to tell you what to do but not how to do.
- **Microprogramming:** node level programming, low level of abstraction complex, difficult to express network level function
- **Macroprogramming:** network-centric view, specifying node group, complexity hidden, difficult to express node level interaction
- **Set-based programming:** compactness, clarity and readability, loss of efficiency, lower speed
Content Overview

• \( \mu \)SETL Expressing example
• Language specification
  • Data types
  • Feature constructor
• Implementation
  • System Architecture
  • Levels of Components
    • \( \mu \)SETL compiler
    • Run-Time Environment
• Evaluation
• Conclusion
μSETL Example

<table>
<thead>
<tr>
<th>Idiom</th>
<th>μSETL expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes with even IDs</td>
<td>{N(i)</td>
</tr>
<tr>
<td>Neighbors of the current node</td>
<td>{i</td>
</tr>
<tr>
<td>Temperature data that are greater than x</td>
<td>{N(i).temp</td>
</tr>
</tbody>
</table>

- Collection of event handlers
- High level expression
- View of individual node
- Applicable to common behavior
Language specification

Data Types

● Node
  • Attributes: location, identifier, temperature, light
  • Example: N(i).light

● Set
  • Example:
    - novolatile {2, 3, 4} (Non-volatile) set of integers
    - {N(2), N(5), N(7)} Set of nodes
    - {i | distance(i, x) == 1} One hop neighborhood of x
    - {N(i).light | i ∈ y} Light data of the nodes in set y

● Other types: integer, float, char and string
Language specification

Constructs

- **Periodic Block:** event handler with time triggered
  
  Example: `period 5000 do
  ........(body)
  end`

- **Monitor Block:** event handler with state triggered
  
  Example: `monitor neighborhood, xyz do
  ........(body)
  end`
μSETL Implementation

System Architecture

μSETL compiler
-----from abstraction to C code

Contiki
-----from C code to binary

RTE
-----communication, data collection to application

An overview of the μSETL architecture.
μSETL Compile

Components

- Emits node-specific source code
- Volatile data detection
- Combine with device information
Volatile Data Detector

To detect volatile variables

- Variable is volatile if
  - Members in set or other variable refers to attributes changing
    In time (distance of node; sensor reading)
  - Variable has dependency on other volatile variables

Types of volatile data
- Distance information
- Resource data
- Data received using receive()

\[
\begin{align*}
\text{neighbors} & := \{i \mid \text{distance}(i) = 1\} \\
\text{temp} & := \{\text{N}(i).\text{temp} \mid i \ \text{IN all}\} \\
\text{received} & := \{i:18 \mid \text{receive(base_station, i)}\}
\end{align*}
\]
μSETL Compile:

Node Specific Code Detector

To detect code for target node

- Node specific by:
  - Name target node for a code
  - By conditional statements
  - Quasi-constants

- Improve run-time efficiency
- Allow code reuse
- Decrease binary size
- Save consumption in nodes (cpu, memory etc.)
- Save power
μSETL Compile:

**Code Generator**

To generate C code for Contiki operating system

- Task performed
  - Code optimization: delete empty loop
    example: period 4000 do.....
  - Calculating a variable: complicated detection overheads
    example: A→B→C→D→E→....
Run Time Environment (RTE)

Support for μSETL program
- communication among nodes
- sensing remote data
- set operation
- avoid packet collision
- update volatile variable

The run-time environment (RTE) in μSETL.
Volatile Data Manager

VDM tasks

- Set registration: during application initialization

- Data update procedure
  - Duration time for volatile data
  - Distance information: ping request <----> ping reply
  - Resource data: resource request <----> resource reply
  - Receive(): waiting for send()

  - Avoid delay and synchronization

- Data consistency
  - No shared variables
  - Wrong loop conditioned by volatile variables
Case 1

**Modified Surge**

Surge: a data collection where base station periodically gather data

\[
\begin{align*}
&\text{period 4000 do} \\
&\quad \text{send(base station, N(node_id).temp);} \\
&\quad \text{temp := \{N(i).temp | i IN all\};} \\
&\quad \text{monitor temp do} \\
&\quad \quad \text{print(temp);} \\
&\quad \text{end} \\
&\text{neighbors := \{i | distance(i) == 1\};} \\
&\text{temp := \{N(i).temp | i IN neighbors\};} \\
&\text{period 4000 do} \\
&\quad \text{avg_i8 := average(temp);} \\
&\quad \text{send(base station, avg_i8);} \\
&\text{end}
\end{align*}
\]

(a) Distributed Surge  (b) Centralized Surge  (c) Modified Surge

\(\mu\text{SETL} \) code for different versions of Surge.

**Topology used for Modified Surge.** Nodes had a transmission range of 50m and an interference range of 100m.

<table>
<thead>
<tr>
<th>Program</th>
<th>Code size (Approximate # of lines)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline Version</td>
<td>(\mu\text{SETL} ) Program</td>
</tr>
<tr>
<td>Modified Surge</td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>6</td>
</tr>
</tbody>
</table>

Result: concise code

less RAM needed
Modified Surge

Packet amount & Energy Cost

Result: node further from base station have more packet loss

Result: μSETL consumes slightly more energy due to overhead in RTE
**Modified Surge**

**Update Packets & Time efficiency**

Result: control packets to communicate among nodes updating data in 200 seconds

Result: \( \mu \)SETL is older version data less efficiency for set operation

Number of non-data packets sent by the RTE on each node for Modified Surge.

Comparison of the age of temperature data between the \( \mu \)SETL and the baseline versions of Modified Surge.
Modified Surge  

Data Consistency

Result: the shorter data life time, the fresher data, more update control packet, more collision.
Case 2

Object Tracking

- Eliminating extra code, save resource by node specific
- Avoid control overhead by monitor block
Conclusion

- μSETL is a set based abstraction for WSN microprogramming
- Set theory provides macro level simplicity, easy for programmer
- It decrease code size efficiently more than 90%