Token Machine Language (TML): An Intermediate Language for Sensor Networks

Ryan Newton, Arvind
MIT CSAIL
{newton, arvind}@mit.edu

Matt Welsh
Harvard University
mdw@eecs.harvard.edu
High Level Languages

- Query Langs (Cougar, TinyDb)
- Spatial Views, EIP
- Data parallel langs
- Regiment
- Rule-based langs

Middleware

Node-level runtime
High Level Languages

- Query Langs (Cougar, TinyDb)
- Spatial Views, EIP
- Data parallel langs
- Regiment
- Rule-based langs

Middleware

- Routing
- Neighborhood mgmt
- Gradients
- Localization
- Heartbeats
- Caching
- Naming/Discovery

Node-level runtime
High Level Languages
- Query Langs (Cougar, TinyDb)
- Spatial Views, EIP
- Data parallel langs
- Regiment
- Rule-based langs

Middleware
- Routing
- Neighborhood mgmt
- Neighborhood mgmt
- Gradients
- Localization
- Heartbeats
- Caching
- Naming/Discovery

Node-level runtime
- TinyOS/NesC
- Sensing
- Event handling
- Local messaging
- Concurrency
- Resource mgmt
High Level Languages
- Query Langs (Cougar, TinyDb)
- Spatial Views, EIP
- Data parallel langs
- Regiment
- Rule-based langs

Middleware
- Routing
- Neighborhood mgmt
- Gradients
- Localization
- Heartbeats
- Caching
- Naming/Discovery
- Neighborhood mgmt

Node-level runtime
- TinyOS/NesC
- Sensing
- Event handling
- Local messaging
- Concurrency
- Resource mgmt
High Level Languages

- Query Langs (Cougar, TinyDb)
- Spatial Views, EIP

Middleware

- Routing
- Neighborhood mgmt
- Naming/Discovery
- Localization
- Gradients
- Heartbeats
- Caching

Node-level runtime

- TinyOS/NesC
- Sensing
- Event handling
- Local messaging
- Concurrency
- Resource mgmt

Compile Down
Question
Question

Can an intermediate language for sensor networks
Question

Can an intermediate language for sensor networks fit Tiny architectures?
Can an intermediate language for sensor networks fit Tiny architectures and be conducive to building higher abstractions (expressive, extensible)?
Question

Can an intermediate language for sensor networks

- fit Tiny architectures
- be conducive to building higher abstractions (expressive, extensible)
- be semantically *simple* and easy to reason about
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>TinyOS/NesC</td>
<td>T</td>
<td>T</td>
<td>?</td>
</tr>
<tr>
<td>TinyDB</td>
<td>T</td>
<td>?</td>
<td>T</td>
</tr>
<tr>
<td>JVM</td>
<td>?</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>
Our approach: token machines

- Atomic actions
- Unified control, communication, and storage model (tokens)
- Simple and lightweight model
Token Machine Model
TM Model
TM Model: concurrency

```{...}
...

```
TM Model: concurrency

Atomic Blocks

{ 
  ... 
}

}
TM Model : concurrency

Atomic Blocks

```c
{  
    if (...)  
    {...} 
}
```
TM Model : concurrency

Atomic Blocks

```
{
  if (...)
  {...}
}
```

One execution context
TM Model : concurrency

Atomic Blocks

token T
{
    if (...) 
    {
        ...
    }
}
TM Model: concurrency

Atomic Blocks

token T
{
    if (...
    {
        ...
    }
}

One execution context
TM Model : big picture
TM Model: big picture
TM Model : big picture
TM Model: big picture
token T
{
    if (...
    {...
}
token T1
{
    if (...)
    {
        ...
    }
}

T1
T2
T3
T4
token T1
{
  if (...)
  {...}
}

Scheduler

T1

T2

T3

T4
token T1
{
    if (...)
    {...}
}

Scheduler

T1
T2
T3
T4
token T1(x,y)
{
  if (...)
  {
    ...
  }
}
token T1(x,y)
{
    if (...)
    {
    ...
    }
}
token T₁(x,y)
{
    if (...
    {...}
}

Scheduler

T₁

T₂

T₃

T₄
TM Model: tokens

token T1(x,y)
{
    if (...)
    {
        ...
    }
}

Scheduler

T1

T2

T3

T4
token $T_1(x,y)$
{
  if (...) 
  {
  ...
  
  }
}
Token Store

Scheduler

T_1(x, y)

{ if (...) { ... } }

Token Store

T_1

T_2

T_3

T_4

TM Model : tokens
**TM Model: tokens**

```ruby
token T1(x, y)
{
  if (...)
  {...}
}
```

Scheduler

Token Store

- $T_1$
- $T_2$
- $T_3$
- $T_4$
token T1(x,y)
{
  if (...)
  {...}
}

TM Model: tokens

Scheduler

Token Store
token T1(x,y)
{
    if (...) {
        ...
    }
}

Scheduler

Token Store

T1

T2

T3

T4
token \( T_1(x,y) \)
{
    if (...)
    {...}
}

Scheduler

Token Store

\( T_1 \)
\( T_2 \)
\( T_3 \)
\( T_4 \)
token $T_1(x, y)$
{
    if (...) 
    {...}
}

Scheduler

Token Store
TM Model: memory allocation

Token Store

T₁  0  1  2  3  ...

T₂  0  1  2  3  ...

T₃  0  1  2  3  ...

Token Store
TM Model : memory allocation

Token Store

Consistent Inter-node virtual addresses: $T_2[4]$
Token Store replaces Stack

- call Red[1](x);
- (subcall Red[1](x)) + 8;
- Uses implicit continuation objects.
User code API

- call
- timed_call
- bcast
- is_scheduled
- deschedule
- is_loaded
- evict
- subcall
Token-unified framework

- Concurrency: atomic token handlers
- Communication: token messages
  - both local and remote messaging
- Memory: token objects on heap
  - subtokens allow for dynamic allocation
Building on TML
Gradients
Gradients

gemit(T, v)
Gradients

gemit(T,v)
grelay(T,v)
Gradients

gemit(T,v)
grelay(T,v)
greturn(v, T_{to}, T_{via}, v_{seed}, T_{aggr})
Gradients

gemit(T, v)
grelay(T, v)
greturn(v, T_{to}, T_{via}, v_{seed}, T_{aggr})

Agnostic to routing and aggregation method
A data gathering program

```
startup Gather;
base_startup SparkGlobal;

token SparkGlobal() {
    gemit GlobalTree();
    timed_schedule SparkGlobal(10000);
}

token GlobalTree() {
    grelay GlobalTree();
}

token Gather() {
    greturn(subcall sense_light(),
           BaseReceive,
           GlobalTree,
           NULL, NULL); 
    timed_schedule Gather(1000);
}
```
Protocol Stacks and Language Towers
Protocol Stacks and Language Towers

Bare Token Machines
Protocol Stacks and Language Towers

Returning Subcalls

Bare Token Machines
Protocol Stacks and Language Towers

Returning Subcalls

Bare Token Machines

subcall $T(x) + 3$
Protocol Stacks and Language Towers

Gradients

Returning Subcalls

Bare Token Machines
Protocol Stacks and Language Towers

Gradients

Returning Subcalls

Bare Token Machines

gemit(T, x...),
grelay(...), greturn(...)

Protocol Stacks and Language Towers

Gradients

Returning Subcalls

Bare Token Machines
Protocol Stacks and Language Towers

Macros
Gradients
Returning Subcalls
Bare Token Machines
Protocol Stacks and Language Towers

- Macros
- Gradients
- Returning Subcalls
- Bare Token Machines

\( \text{flood}(T) \)
Protocol Stacks and Language Towers

Macros

Gradients

Returning Subcalls

Bare Token Machines
Protocol Stacks and Language Towers

?  

Macros

Gradients

Returning Subcalls

Bare Token Machines
Compiling to TML
High level language: Regiment example
let $R = \text{khops} \ 1 \ A$
Wins for Regiment + TML

Token namespace serves for region coordination:

- Region membership = holding a token
- Gradients used for constructing and aggregating all continuous regions
Future Work and Open Questions

- Dynamic loading
- Optimization: size
  - eliminating generated extra args
- Some dirty work.
- Implementation in real time OS?
The End.

Ryan Newton, Arvind
MIT CSAIL
{newton, arvind}@mit.edu

Matt Welsh
Harvard University
mdw@eecs.harvard.edu
TML Advantages

- Atomicity is a simple semantic model for code generators and transformers to target
- Lightweight - no GC, no threads, no blocking
- Action abortion => real-time potential
Implementing Regiment

Stage 1: Static Elaboration

2. Query Circuit

3. Token Machine

4. TinyOS program

5. Runtime

world

rmap

rfold

until

f

g
### Motivation: macroprogramming

Sensor Data

<table>
<thead>
<tr>
<th></th>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>t+1</td>
<td>349</td>
<td>23</td>
<td>2934</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>302</td>
<td>10</td>
<td>29</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>299</td>
<td>0</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>105</td>
<td>0</td>
<td>1099</td>
<td>0</td>
</tr>
</tbody>
</table>

...
Motivation: macroprogramming

<table>
<thead>
<tr>
<th>Sensor Data</th>
<th>Program P over data</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>349</td>
<td>23</td>
</tr>
<tr>
<td>302</td>
<td>10</td>
</tr>
<tr>
<td>299</td>
<td>0</td>
</tr>
<tr>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>s2</td>
<td>2934</td>
</tr>
<tr>
<td>302</td>
<td>29</td>
</tr>
<tr>
<td>299</td>
<td>0</td>
</tr>
<tr>
<td>105</td>
<td>1099</td>
</tr>
<tr>
<td>s3</td>
<td>1</td>
</tr>
<tr>
<td>302</td>
<td>3</td>
</tr>
<tr>
<td>299</td>
<td>8</td>
</tr>
<tr>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>s4</td>
<td>2900</td>
</tr>
<tr>
<td>302</td>
<td>3</td>
</tr>
<tr>
<td>299</td>
<td>8</td>
</tr>
<tr>
<td>105</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>t</td>
<td>t+1</td>
</tr>
</tbody>
</table>

...
Motivation: macroprogramming

"Full" access

Program P over data

Sensor Data

<table>
<thead>
<tr>
<th>s1</th>
<th>s2</th>
<th>s3</th>
<th>s4</th>
<th>t</th>
<th>t+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>349</td>
<td>23</td>
<td>2934</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>302</td>
<td>10</td>
<td>29</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>299</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>0</td>
<td>1099</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

... ... ... ... ...
Motivation: macroprogramming

```
<table>
<thead>
<tr>
<th>Sensor Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>s1</td>
</tr>
<tr>
<td>349 23 2934 1</td>
</tr>
<tr>
<td>s2</td>
</tr>
<tr>
<td>302 10 29 3</td>
</tr>
<tr>
<td>s3</td>
</tr>
<tr>
<td>299 0 0 8</td>
</tr>
<tr>
<td>s4</td>
</tr>
<tr>
<td>105 0 1099 0</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
```

“Full” access

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>309 21 2900 1</td>
</tr>
<tr>
<td>302 10 29 3</td>
</tr>
<tr>
<td>309 21 2900 1</td>
</tr>
</tbody>
</table>
```

1. RESTRICT P

Program P over data
Motivation: macroprogramming

1. RESTRICT P

"Full" access

Program P over data

2. PUSH P IN-NETWORK