Collaborative Reactive Behavior in Heterogeneous Wireless Sensor Networks

Jedidiah McClurg, Goce Trajcevski, Jesse Yanutola
{jrm807, goce}@eecs.northwestern.edu, jesseyanutola2012@u.northwestern.edu

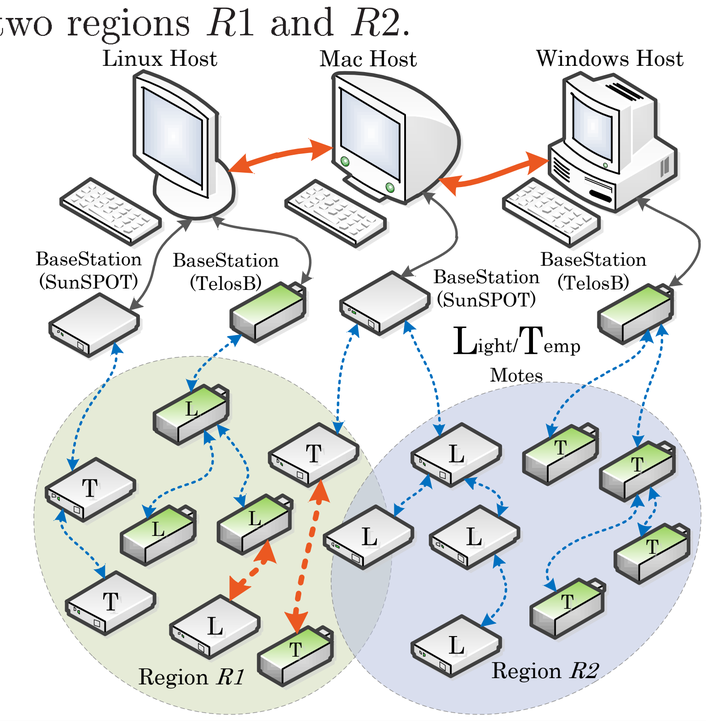
NORTHWESTERN UNIVERSITY

Problem

Heterogeneous Wireless Sensor Network (WSN) development is especially difficult, in terms of both *programmatic details* of the various sensor motes, and *energy-efficiency* of the communication between them. The ECA (on Event if Condition then Action) paradigm from Active Databases offers a **Trigger-based** methodology which could help with both of these, but can we build a system which allows development using this approach?

Example Topology

The following figure shows an example heterogeneous WSN with motes grouped into two regions $R1$ and $R2$.



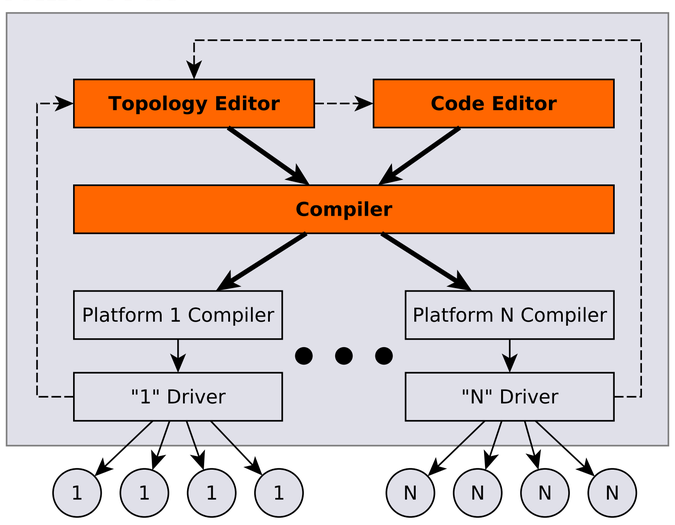
Example Query

In the above WSN, we could issue a query with respect to the regions:

Q1: *Whenever the average temperature over the last 30 seconds in Region R1 exceeds 90° F, if the average light intensity in Region R2 is ≥ 80 lumens, then switch the sensors to a fast sampling mode, and report the average readings to the sink.*

Our Solution

Our system allows high-level triggers and WSN toplogy to be specified. A compiler then translates the specification to mote-specific code.



Specifying Triggers

The code editor allows a Trigger-based query to be specified.

```
ON EVENT (AVG(R1.temp[-30,0]) > 90)
IF (AVG(R2.light) >= 80) (
   SET R1.freq[0,30] = 1;
   SELECT AVG(R1.temp);
   SELECT AVG(R2.light)
)
```

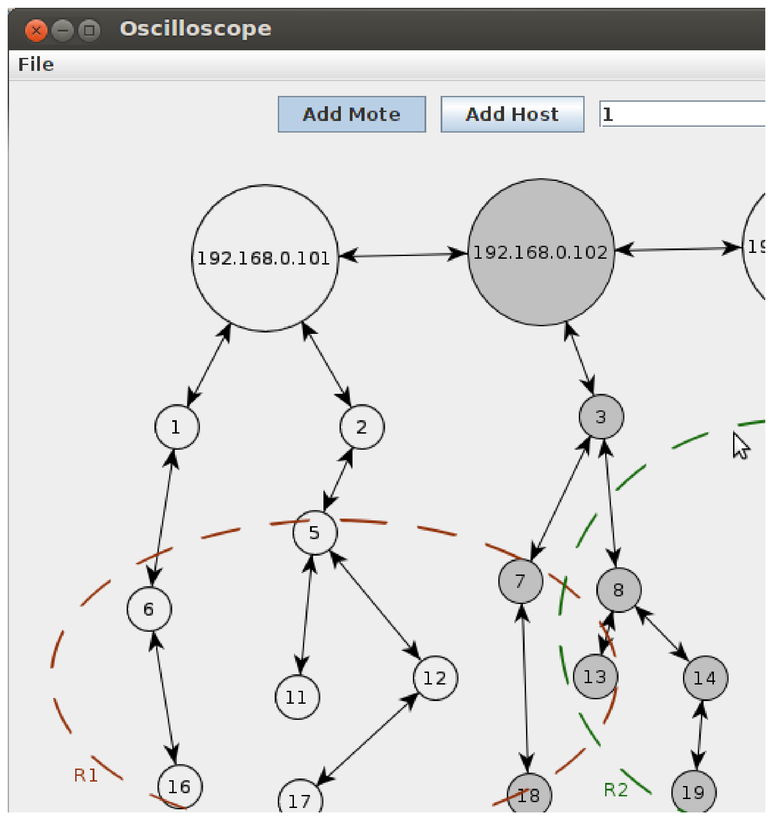
Compilation

Once the user has specified the WSN topology and a query with respect to the specified regions, the compiler is invoked, and the compilation proceeds in two stages:

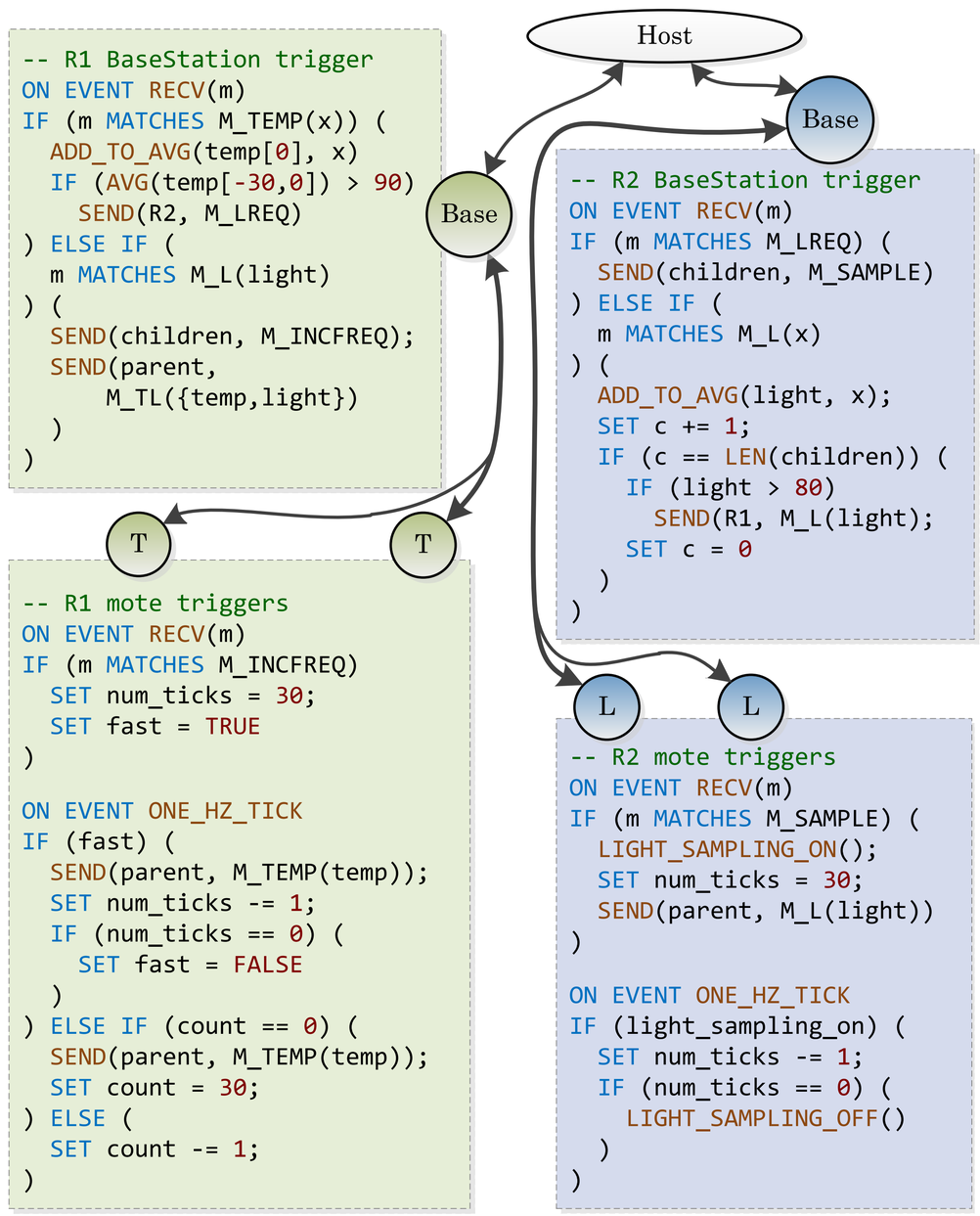
1. Compile each system-level trigger T_i to a set of mote-specific triggers $\{M_{j,i}\}$.
2. For each mote j, compile the mote-specific triggers $\{M_{j,i}\}$ to native code for that mote.

Specifying WSN Topology

The topology editor allows the user to chose motes and hosts (sinks) from among the reachable devices, and organize them into a desired tree toplogy. Regions can also be specified at this point.



Compiling System-level Triggers to Mote-level Triggers




```
-- R1 BaseStation trigger
ON EVENT RECV(m)
IF (m MATCHES M_TEMP(x)) (
  ADD_TO_AVG(temp[0], x)
  IF (AVG(temp[-30,0]) > 90)
    SEND(R2, M_LREQ)
) ELSE IF (
  m MATCHES M_L(light)
) (
  SEND(children, M_INCFREQ);
  SEND(parent,
      M_TL({temp,light})
  )
)
```

```
-- R1 mote triggers
ON EVENT RECV(m)
IF (m MATCHES M_INCFREQ)
  SET num_ticks = 30;
  SET fast = TRUE
)

ON EVENT ONE_HZ_TICK
IF (fast) (
  SEND(parent, M_TEMP(temp));
  SET num_ticks -= 1;
  IF (num_ticks == 0) (
    SET fast = FALSE
  )
) ELSE IF (count == 0) (
  SEND(parent, M_TEMP(temp));
  SET count = 30;
) ELSE (
  SET count -= 1;
)
```

```
-- R2 BaseStation trigger
ON EVENT RECV(m)
IF (m MATCHES M_LREQ) (
  SEND(children, M_SAMPLE)
) ELSE IF (
  m MATCHES M_L(x)
) (
  ADD_TO_AVG(light, x);
  SET c += 1;
  IF (c == LEN(children)) (
    IF (light > 80)
      SEND(R1, M_L(light);
    SET c = 0
  )
)
```

```
-- R2 mote triggers
ON EVENT RECV(m)
IF (m MATCHES M_SAMPLE) (
  LIGHT_SAMPLING_ON();
  SET num_ticks = 30;
  SEND(parent, M_L(light))
)

ON EVENT ONE_HZ_TICK
IF (light_sampling_on) (
  SET num_ticks -= 1;
  IF (num_ticks == 0) (
    LIGHT_SAMPLING_OFF()
  )
)
```

The first stage of the compilation involves translating the system-level triggers into a set of specific triggers for each mote.

There are several ways to do this step which provide varying degrees of WSN communication efficiency, but we focus on the *pull-based* mode, in which a trigger

`ON EVENT` E `IF` C `THEN` A

is compiled so that the values mentioned in event E are sent to the *head* of E (i.e. a node which is a parent of all regions in E) every time they change, and the values mentioned in the condition C are requested (*pull-based*) from the regions in C when E evaluates to *true*. Then, if the condition is *true*, the head for each statement $S \in A$ is notified to execute the statement.

Compiling Mote-level Triggers to Native Code

In the second stage, the mote-specific triggers $\{M_{j,i}\}$ are collected for each mote j, and triggers with the same event are combined. Because of the transformations performed in the first stage, this results in at most two triggers, one with event `RECV(m)` and one with event `TICK`, which are easily converted to their analogues in nesC or Java for the mote.