Demo Abstract: Collaborative Reactive Behavior in Heterogeneous Wireless Sensor Networks

Jedidiah McClurg [*] Goce Trajcevski [*] Jesse Yanutola

Dept. of Electrical Engineering and Computer Science
Northwestern University
Evanston, IL, 60208, USA
{jrm807, goce}@eecs.northwestern.edu, jesseyanutola2012@u.northwestern.edu

Abstract

Wireless Sensor Networks (WSN) which contain heterogeneous nodes and monitor multiple phenomena present a unique set of challenges in regards to efficient management of reactive behavior. The ECA (on *Event* if *Condition* then *Action*) paradigm from Active Databases offers a solution via event-based synchronization which provides reduced energy consumption compared to continuous monitoring. In our demo, we show how to utilize this approach via compilation of system-level ECA triggers to mote-specific trigger code. We also demonstrate the practicality of the approach by constructing a heterogeneous WSN of TelosB/SunSPOT motes and using our tools to implement reactive behaviors based on temperature/luminance data.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design — *wireless communication*; H.2.3 [**Database Management**]: Languages — *query languages*

General Terms

Design, Experimentation, Measurement

Keywords

Heterogeneous Nodes, Meta-trigger, Wireless Sensor Network, Reactive Behavior, Environmental Monitoring, Active Database, ECA Trigger, TelosB, SunSPOT

1 Introduction

In many applications, Wireless Sensor Networks (WSN) play the role of distributed databases, processing both instantaneous and continuous queries of interest. Users express their queries in an SQL-like declarative syntax while the (distributed) query processor executes them, transparently performing optimization tasks on subsets of the nodes. TinyDB [3] is an example of such a system which provides (1) event-based behavior, i.e. continuously evaluating a particular query only upon detection of a certain event, and (2) actuation queries, i.e. executing a particular action in response to the results of a given sampling query.

[*] Research supported by NSF-CNS: 0910952

Copyright is held by the author/owner(s).

SenSys '12, November 6–9, 2012, Toronto, Canada.
ACM 978-1-4503-1169-4/11/12

However, the energy-saving capabilities of traditional Active Database triggers conforming to the ECA (on *Event* if *Condition* then *Action*) paradigm [4] are lacking in the WSN context. An important aspect of minimizing the energy consumption when executing a given trigger is collaborative management of *composite event* detection and evaluation of conditions which may be *continuous* queries. Another key problem arises due to the fact that a user may specify a trigger in which the event and condition pertain to values of *different phenomena* in *different geographical regions* without properly considering the balance between the push-based vs. pull-based [2] communication between nodes for the purpose of firing needed actions. In many cases, the user would benefit from tools which bypass this issue by providing a declarative language for writing triggers and functionality to generate an efficient execution policy automatically.

To that end, in [5] we presented the concept of a *Meta-trigger*, i.e. a construct which offers a translation of the user's ECA trigger into a system-aware execution. Specifically, we aimed for efficient management of requests like:

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 $\geq$ 80 lumens, then switch the sensors to a fast sampling mode, and report the average readings to the sink.*

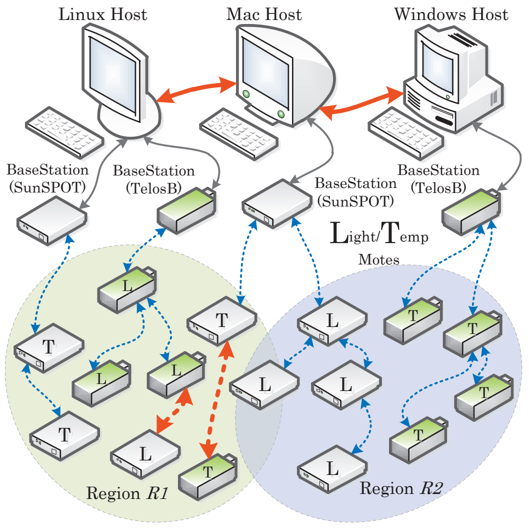


Figure 1. Example Monitoring Scenario

While our earlier work provided a proof-of-concept regarding the benefits of meta-triggers, in this demo we report some significant improvements. As shown in Figure 1, we have broadened the context to incorporate both *heterogeneous* nodes and *multiple sinks/base stations*. In addition, instead of manually programming the motes, we have developed a compiler which can generate mote-specific code from system-level triggers written in a simple query language.

2 System Architecture and Implementation

Our toolset consists of two main components: (1) a declarative interface for specifying the WSN topology and the system-level triggers, and (2) a compiler which generates the mote-specific code implementing the trigger behavior.

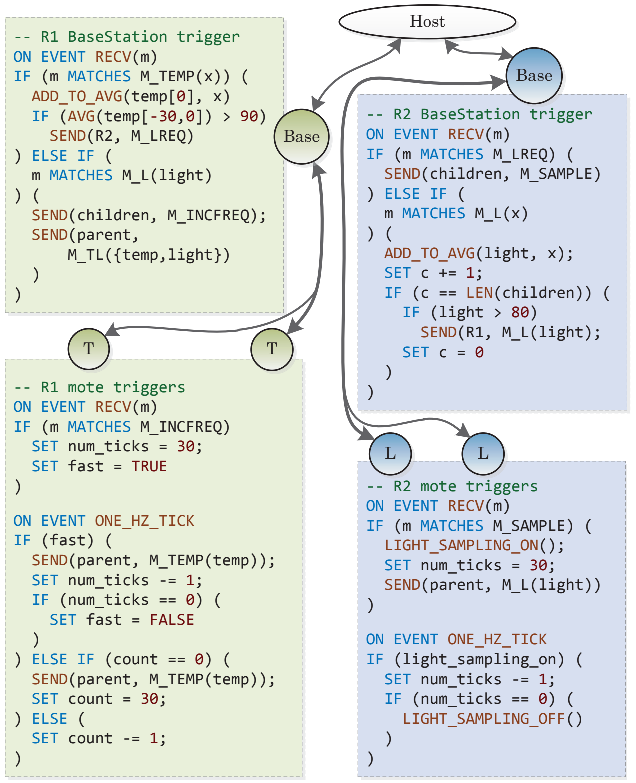
The user interface is a cross-platform Java application. It allows the user to conveniently arrange host (sink) and TelosB/SunSPOT nodes and connect them using arrows to form the desired topology. Menus allow selection of the particular phenomena to be monitored, reflecting the available sensors on a each type of mote. The interface also provides a text field for specifying the various components of the trigger (Event, Condition, and Action).

The compiler component is a cross-platform application written in OCaml. Once the user completes the query/trigger specification, the compiler is invoked, generating nesC [1] code for the TelosB motes and Java code for the SunSPOT motes. Each mote is then successively connected to the host so that its code can be built and flashed to the device.

As an example of the compilation, we refer to the request in the introduction. This system-level query Q1 can be written in our ECA query language as follows:

```
ON EVENT (AVG(R1.temp[-30,0]) > 90)
IF (AVG(R2.light) >= 80) (
  SET {R1,R2}.freq[0,30] = 1.0;
  SELECT AVG({R1.temp,R2.light}))
```

Using this, the compiler generates the corresponding mote-specific triggers which implement the overall reactive behavior (cf. Figure 2). Note that the resulting system-level implementation generated by the compiler is relatively efficient. All sensors in Region R2 can sleep until reception of an M_LREQ message from the Region R1 BaseStation, which indicates that the **event** ("average temperature over the last 30 seconds in Region R1 exceeds 90°F") has occurred. Region R2 then checks the **condition** ("average light intensity in Region R2 is $\geq$ 80 lumens"), and if true, provides R1 with average data for the **action**, namely reporting averages back to the host and temporarily increasing the sample rate.



```
-- 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()
  )
)
```

Figure 2. Compiled Triggers for Regions R1 and R2

3 Demo Overview

Our demo environment will correspond to Figure 1, with a Linux virtual machine replacing the Macintosh host since we have not yet obtained Mac hardware/software. Specifically, the setup will consist of a Linux laptop, a Windows laptop, battery-powered TelosB and SunSPOT motes (10 each), and battery-powered heat and light sources. The demo will have three main steps which outline several realistic scenarios:
Phase 1: We will first illustrate the graphical specification of the network topology, after which the system will discover the motes and help them obtain the appropriate routing information. We will then illustrate the specification of particular triggers, both for "simple" cases pertaining to a single phenomenon and a particular type of motes, as well as "composite" triggers pertaining to both luminosity, temperature and different types of motes.
Phase 2: In the second step, we will present the compilation of the triggers and download the generated code onto the individual motes participating in the respective scenarios.
Phase 3: Finally, we will monitor the actual reactive behavior in the corresponding WSNs. The host laptops (sinks) will be running our "oscilloscope" application to show the particular query results for each of the scenarios, along with the frequency of their measurement and reporting.

4 References

[1] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation*, PLDI '03, pages 1–11, New York, NY, USA, 2003. ACM.

[2] A. Hinze. Efficient filtering of composite events. In A. James, M. Younas, and B. Lings, editors, *New Horizons in Information Management*, volume 2712 of *Lecture Notes in Computer Science*, pages 164–164. Springer Berlin / Heidelberg, 2003.

[3] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TinyDB: An Acquisitional Query Processing System for Sensor Networks. *ACM Trans. Database Syst.*, 30(1):122–173, Mar. 2005.

[4] N. W. Paton, F. Schneider, and D. Gries, editors. *Active Rules in Database Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1998.

[5] G. Trajcevski, N. Valtchanov, O. Ghica, and P. Scheuermann. A Case for Meta-Triggers in Wireless Sensor Networks. In *Eighth IEEE International Symposium on Network Computing and Applications, NCA 2009*, pages 171 –178, july 2009.