Building Sensing Applications with the Owl Platform

» Presented by Richard Martin

» And more, including
» Robert Moore, Bernhard Firner,
» Yanyong Zhang, Richard Howard,
» Eitan Fenson, and many others
The Opportunity

Moore’s Law: # transistors on cost-effective chip doubles every 18 months

Bell’s Law: a new computer class emerges every 10 years

Today: 1 million transistors per $

Same fabrication technology provides CMOS radios for communication and micro-sensors
The Vision
Down the garden path of sensor networks

Programming a sensor network:

- Multi-hop
- Ad-hoc
- Aggregation and compression
- Energy conservation of whole application is paramount
- Novel operating systems, programming languages and environments
A rose by any other name

- 1999 Smart Dust
- 2000 Sensor Networks
- 2004 Internet of Things
- 2005 Ambient Intelligence
- 2009 Swarms

- ~15 years on, we still have not realized the vision.

What happened?
Problems

• Problems people talked about:
  – Energy conservation
  – Scaling number of sensors
  – Efficiency of code data size in small sensors
  – Routing

• More meaningful problems:
  – Too expensive for application domains
  – Difficult to develop applications
  – Can't re-use infrastructure
  – Not general purpose
Owl Platform

Novel constraint:
Enable application development by undergraduate level programmers

Standard languages, programming environments

Separation of concerns:
- Application developer: what is the data and app logic?
- Sensor designer: hardware and interface to aggregation layer
- System administrator: keeping the system running

Solves energy and scaling issues differently
- Move to the sensor designer level, leave the app out of it.
A Different Model: Layers

- Sensors connect to an intermediate layer that hides details.
- Solvers build higher-level representations from low-level ones.
- A uniform model of the world allows sharing.
- Applications run in standard environments in the cloud.
Layered Model – cont.

Layering allows for separation of concerns
  - Sensor designers
  - Deployment/IT staff
  - Solver/Algorithm developers
  - Application developers

Each layer exports interfaces and methods
  - libraries for different languages (Java, C++, Ruby)

Components communicate by state in the aggregator and world models.
  - components (solvers, application) use network
  - allows for proprietary solvers, open source system
Sensor examples

Chair occupancy

Door open/close

Kinect Skeleton

Coffeepot Temperature

Power Consumption

Phone Tracking
Owl sensor model

I want to add sensed data. What do I do?

An adaptation layer puts the sensor data into this format:

Physical layer, Source Sensor ID, Target ID, Time, Signal Level, Sensed Data

Sends the above over the Internet to the aggregator
World model

World server holds a model of the world
  - Shared state between applications
  - Partitioning of the name space between applications

The world is a hierarchical name space of variables
  - Similar to LDAP, Windows Registry, SNMP MIB
    - Balance of structure, open-ness

Variables have types, times, and an origin
### World Model Data Format

Enter URI filter pattern: [Enter URI filter pattern]  
Enter a new URI: [Enter a new URI]  
- [Fetch/Refresh]  
- [Create URI]

Update Attributes  
Delete URI/Attribute

<table>
<thead>
<tr>
<th>Object URI</th>
<th>Attribute Name</th>
<th>Origin</th>
<th>Data</th>
<th>Created</th>
<th>Expires</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halloween2011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>region</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test temperature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test temperature</td>
<td>creation</td>
<td>Ben</td>
<td>0x</td>
<td>1319731453082</td>
<td>0</td>
</tr>
<tr>
<td>test temperature</td>
<td>sensor.temperature</td>
<td>Ben</td>
<td>1.26</td>
<td>1319731470449</td>
<td>0</td>
</tr>
<tr>
<td>test temperature</td>
<td>location.uri</td>
<td>grail/discriminator solver</td>
<td>Halloween2011.locations.Ben's Desk</td>
<td>1319764031065</td>
<td>0</td>
</tr>
<tr>
<td>test temperature</td>
<td>temperature</td>
<td>grail/temperature solver</td>
<td>19</td>
<td>1319763017202</td>
<td>0</td>
</tr>
<tr>
<td>test temperature</td>
<td>mobility</td>
<td>grail/mobility_solver</td>
<td>0x00</td>
<td>1319763659777</td>
<td>0</td>
</tr>
<tr>
<td>test temperature</td>
<td>Add New Attribute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test uri</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
World Model Data Format – cont.

Object URI name:
  Example: edu.rutgers.owl.makefaire.keys

Attributes/Data: - similar to fields in a data structure
  Attribute is a string, data is binary
  Examples:
    • Mobility 0/1
    • Person1 : XY points of a kinect skeleton

Origin: - who or what created the data
  Could match to a public key – not done yet

Created/Expires:
  When was the data created, and how long is it valid?
Owl solver chains

Semantic Meaning

High level Solvers

Low level Solvers

Sensor Data

- Is everyone having tea?
- Where is my mug?
- Is this a good talk?
- Is my advisor here?
- Is there fresh coffee?
- Did someone leave the door open?

Gathering

Item Locator

Location Discriminator

Localization

Location Fingerprinter

Talk Attendance

Calendar Events

Room In Use

Coffee Brewing

Propped Door

Signal Statistics

Chair Switch

Power Consumption

Temperature

Door Latch

Sensor Data Solvers
Owl application patterns

SMS/Email Alerts

Status Maps
Example Owl application patterns- cont.

Reports

Physical Actuation
Deployed App: Monitoring Animal Laboratories

Short timescales: Operations (notifications)
Long timescales: Veterinarians (reports)

Temperature
Light
Doors (switch)
Humidity
Deployed App: Home Monitoring
Putting it all together: Demo Panels App
Putting it all together: Demo Panels App

Door Open/Closed
Putting it all together: Demo Panels App

- Welcome
- Standing Water Detected

[Image of a house with standing water detected nearby]
Putting it all together: Demo Panels App
Putting it all together: Demo Panels App
Putting it all together: Demo Panels App

Left without Wallet and Keys
Wallet and Keys application

When (the door changes state to open)
   Foreach item in [wallet, keys]
      If (the item has not moved in the last 10 seconds) then
         add it to the list of missing items

   If (the list is non-empty)
      send an alert with the list of missing items.
Solver chain for the Wallet and Keys panel

Demo Panels App

Wallet and Keys Solver

Binary State Solver (Door)

Mobility Solver (Keys and Wallet)
Binary State and Mobility Solvers

Binary state:
Read sensor value, put open/closed state in the world model

Mobility detection:
Read wireless received signal power over period of N seconds
If signal variance is over a threshold, change object's state to moving in the world model
Owl Lines of Code

The diagram shows the number of lines of code for various projects, categorized by programming language.

- **World Model**: C++ (6000 lines)
- **Aggregator**:
  - C++ (1500 lines)
  - Java (500 lines)
- **Demo Panels App**:
  - C++ (1000 lines)
  - Java (0 lines)
- **Wallet and Keys**: C++ (300 lines)
- **Binary state solver**: C++ (100 lines)
- **Mobility solver**: C++ (50 lines)
Other Applications

• Leak detection
  – Sense standing water, email/SMS if water detected

• Office space assignment
  – Sense door open/closes, assign new students to lightly used offices

• Fresh Coffee
  – Sense temperature of coffee pot, email/SMS if a temp spike

• Chair Stolen
  – Email/SMS if a chair is moved away from the owner's cubicle

• Loaner Bicycle Inventory
  – Count # of bicycles in a room to see if one is available.
Evaluation of ease of development

Perform a user-study in next 3 months.

Have undergraduates develop simple application in Owl and another system (Smartthings)
   Send an alert when 3 door sensors are triggered in order within a time window.

Metrics: learning time, development time, lines of code, functionality, code quality.

Experiment has passed human subject review approval process
Conclusions and Future Work:

Conclusions:

- Application development simplified
- Codebase to accomplish sensor applications surprising small

Future work:

- Leverage origin ID for security and privacy
- Continue to add applications in student seminars and projects
- Need to add actuation layers for next version
Owl Resources:

Main Developer's page:
  http://www.owlplatform.com/developers.php

World Model:
  https://github.com/OwlPlatform/world-model

Aggregator:
  https://github.com/OwlPlatform/aggregator

Makerfaire Demo Application:
  https://github.com/romoore/maker-demo

Wallet and Keys solver:
  https://github.com/romoore/wallet-and-keys

Binary state solver (switch):
  https://github.com/OwlPlatform/binary_state_solver

Signal strength solvers (mobility):
  https://github.com/OwlPlatform/signal-strength-solvers
Backup slides

Backup slides
Lines of Code

World Model (C++): 6274

Aggregator (Java): 1439

Makerfaire Demo Application (Java): 1142

Wallet and Keys solver (Java): 507

Binary state solver (switch) (C++): 273

Signal strength solver (mobility) (C++): 244
Finding Variables in the World Model

StepResponse mobilityResponse =
  this.asClient.getStreamRequest(itemIds,
      System.currentTimeMillis(), 0, mobilityAttributes);

StepResponse doorResponse =
  this.asClient.getStreamRequest(doorIds,
      System.currentTimeMillis(), 0, doorAttributes);

try {
  // Keep going until an error or a mobility update
  while (!mobilityResponse.isComplete() &&
    !mobilityResponse.isError() &&
    (!doorResponse.isComplete() && !doorResponse.isError())) {
    if (mobilityResponse.hasNext()) {
      
```
StepResponse mobilityResponse =
    this.asClient.getStreamRequest(itemIds,
        System.currentTimeMillis(), 0, mobilityAttributes);

StepResponse doorResponse =
    this.asClient.getStreamRequest(doorIds,
        System.currentTimeMillis(), 0, doorAttributes);

try {
    // Keep going until an error or a mobility update
    while (!mobilityResponse.isComplete() &&
        !mobilityResponse.isError() &&
        (!doorResponse.isComplete() && !doorResponse.isError())) {
        if (mobilityResponse.hasNext()) {

Example aggregator code

/** Connection to the aggregator. using "poll" */

SolverAggregatorConnection agg = new SolverAggregatorConnection();

if (!this.agg.connect(10000)) {
    System.err.println("Unable to connect");
    return false;
}

Example aggregator code

/** Connection to the aggregator. using "poll" */

SolverAggregatorConnection agg = new SolverAggregatorConnection();

if (!this.agg.connect(10000)) {
    System.err.println("Unable to connect");
    return false;
}

SubscriptionRequestRule everythingRule = new SubscriptionRequestRule();
    everythingRule.setUpdateInterval(0l);

    everythingRule.setPhysicalLayer(SampleMessage.PHYSICAL_LAYER_ALL);
    this.agg.addRule(everythingRule);

    SampleMessage sample = null;
    while ((sample = this.agg.getNextSample()) != null) {
        Attribute attr = new Attribute();
        // If the RSSI value is above threshold, say it's "nearby".
        if (sample.getRssi() > RSSI_THRESHOLD) {
            System.println("Chair is nearby");
        }
    }
Sensor simplicity

• Sensor node cost is a limitation for many applications
  – Applications enabled at sensor cost of $100, $10, $1, 10¢, 1¢?
• Cost assumptions based on scaling Moore's law real omit real constraints
  – 15 years show these constraints are fundamental

• Cost is driven by the number and type of components, not Moore's law!

• TO reduces costs by several factors
  – enough to expand the application space ($80->$10)
• Marginal costs will only go down if there is a true single-chip sensor
  – But high fixed costs remain a barrier for a true single chip solution!
Two wireless sensor boards

**Classic**

**Transmit-Only**
TO-PIP(2013)
Component counts

Sensor Node Component Count

- IC count
- Other Components
- Total

Graph showing component counts for RoboMote, TelosB, Mica, EPIC Core, G-PIP4, and TO-PIP.