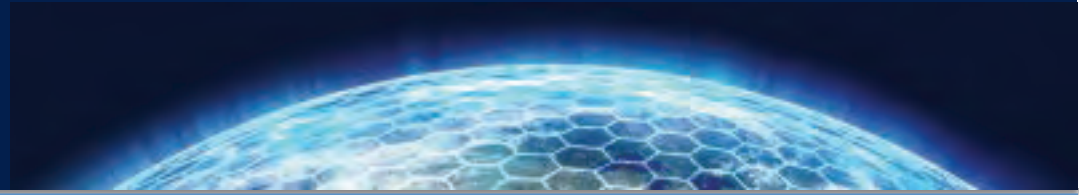




iCyPhy



# Accessors

## A Software Architecture for IoT

*Edward A. Lee and Christopher Brooks*

*EECS, UC Berkeley*

*Tutorial on Tools for Real-Time Systems (TuToR)*

*In conjunction with the Real-Time Systems Symposium (RTSS)*

*December 5, 2017 – Paris, France*



University of California at Berkeley

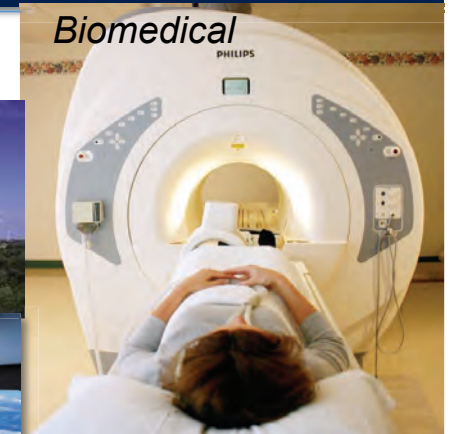
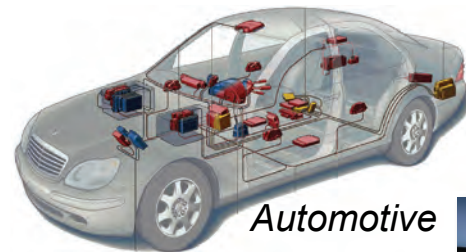


# Cyber-Physical Systems

Focus on the Internet of *Important* Things

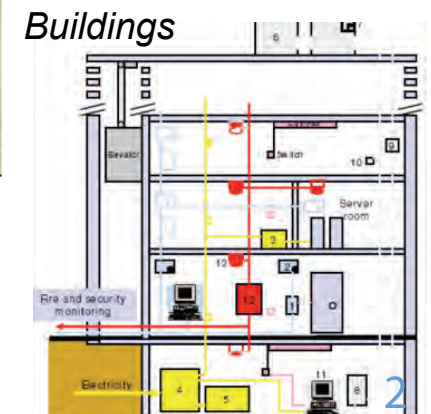
## Not just information technology:

- Cyber + Physical
- Computation + Dynamics
- Security + Safety



## Properties:

- Highly dynamic
- Safety critical
- Uncertain environment
- Physically distributed
- Sporadic connectivity
- Resource constrained



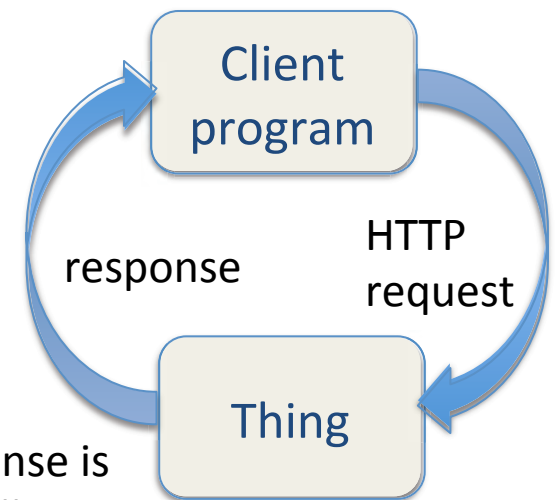
We need engineering **models** and **methodologies** for dependable cyber-physical systems.



# A Common IoT Design Pattern: REST with AACs

A RESTful service [Fielding & Taylor 2002] is accessed using a design pattern common on the web that we call *Asynchronous Atomic Callbacks* (AAC) (also called the *Reactor Pattern*).

In the Web, AAC is widely used. It is central to many popular internet programming frameworks such as Node.js & Vert.x, and to CPS frameworks such as TinyOS.



Response is typically asynchronous to avoid blocking the client program.

Response handler executes atomically.

URL encodes all state info (credentials, commands, etc.)



# Example in JavaScript

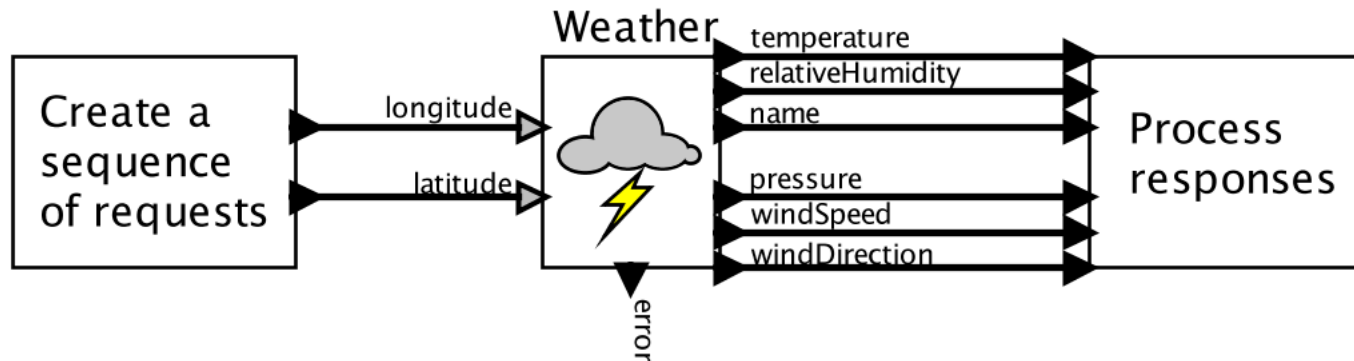
```
// Import a module providing network services
var http = require("http");
// Construct a URL encoding a request
var url = "http://foo.com/deviceID/...";
// Issue the request and provide a callback
http.get(url, function(response) {
    // ... handle the response ...
});
```

The callback function will be called atomically some time later when the server response arrives.



# Another Common Design Pattern: *Actors*

Streaming requests:



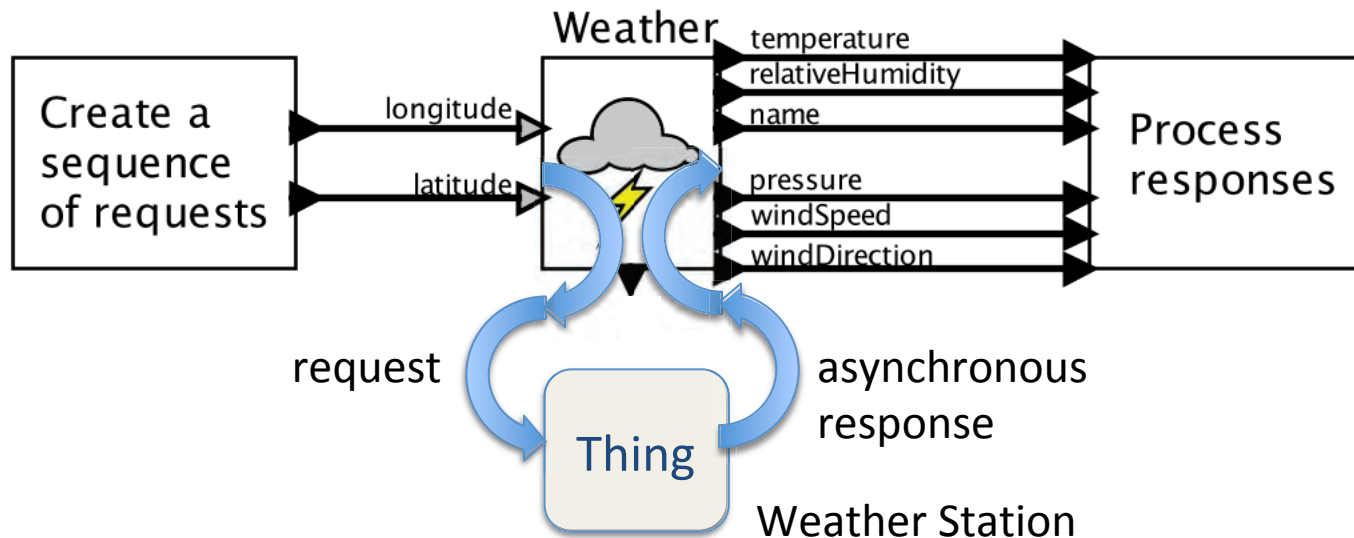
Sequence of requests for a service (a stream) triggers a sequence of responses.

**Actors embrace concurrency and scale well.**



# Actors and AAC

Streaming requests:

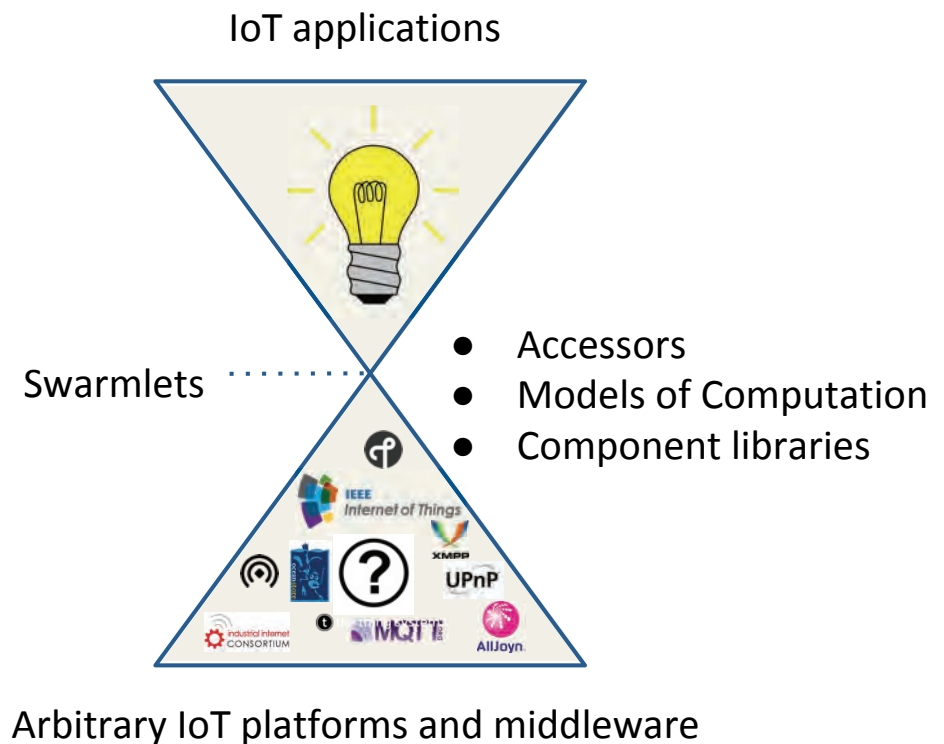


This is a key property of accessors, a design pattern for IoT that embraces concurrency, asynchrony, and atomicity.



# Principled Composition of Devices and Services

- A **Swarmlet host** is a platform for component-based design (analogous to a **browser**, but for things, not people).
- A **Swarmlet host** can instantiate **accessors**, which serve as local proxies for devices and services.
- **Accessors** can be provided by device vendors or third parties.
- **Accessors** enable interoperability of **independently designed** components.

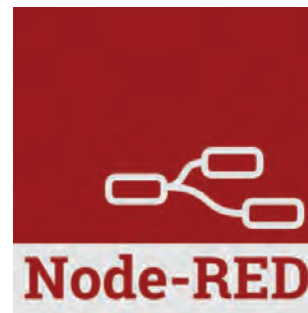




# We are not alone pursuing this approach

## Notable efforts:

- Node Red (IBM)
- Calvin (Ericsson)



From: "Home Automation with Node Red, JeeNodes and Open Energy Monitor," Dom Bramley's Blog of Maximo and the 'Internet of Things', IBM Developer Works, Dec., 2013.

The image displays a Node-RED interface for home automation. The main workspace shows a flow for 'RFM12 Inbound' and 'RFM12 Outbound'. The 'RFM12 Inbound' flow starts with a 'Serial In' node connected to '/dev/ttyUSB0:57600', followed by a 'Parse RF12B' node, a 'Serial In Debug' node, and a 'Publish RFM12B' node. The 'RFM12 Outbound' flow starts with an 'RFM12B Sender' node, followed by a 'Send to Node' node, a 'Delay 5 millis' node, a 'sender debug' node, and a 'devttyUSB0:57600' node. Below the main flow, there are two 'Node Mapping' sections: 'Inbound Node Mapping' and 'Outbound Node Mapping'. The 'Inbound Node Mapping' section shows nodes for 'RFM12 Node 8 IN', 'RFM12 Node 12 IN', 'RFM12 Node 14 IN', 'RFM12 Node 19 IN', and 'RFM12 Node 20 IN' connected to 'Shed Node Raw', 'Loft Node Raw', 'Garage Node Raw', 'Bedroom Node Raw', and 'Energy Monitor LCD Raw' respectively. The 'Outbound Node Mapping' section shows nodes for 'Ambient Light Lounge 1', 'Ambient Light Lounge 2', and 'LCD Time' connected to 'RFM12 Node 6 OUT', 'RFM12 Node 6 OUT', and 'RFM12 Node 20 Out' respectively. On the left, there is a 'DESCRIBE' section with a document icon and a 'CONNECT' section with a network icon. At the bottom, there is a 'MANAGE' section with a network icon and a 'DEPLOY' section with a mobile phone icon. The right side of the interface shows a 'debug' console with a list of messages, including 'OK 19 176 0 0 0 41 2 31 0', '17.6', '55.3', '3.1', 'ok', 'OK 46 25 2 194 10', 'ack', '5.37', '2.754', 'ok', and '? 31 100 238 71 239 27 118 90 198 235 141 24 139 230 250 138 98 124 208 80 223'.



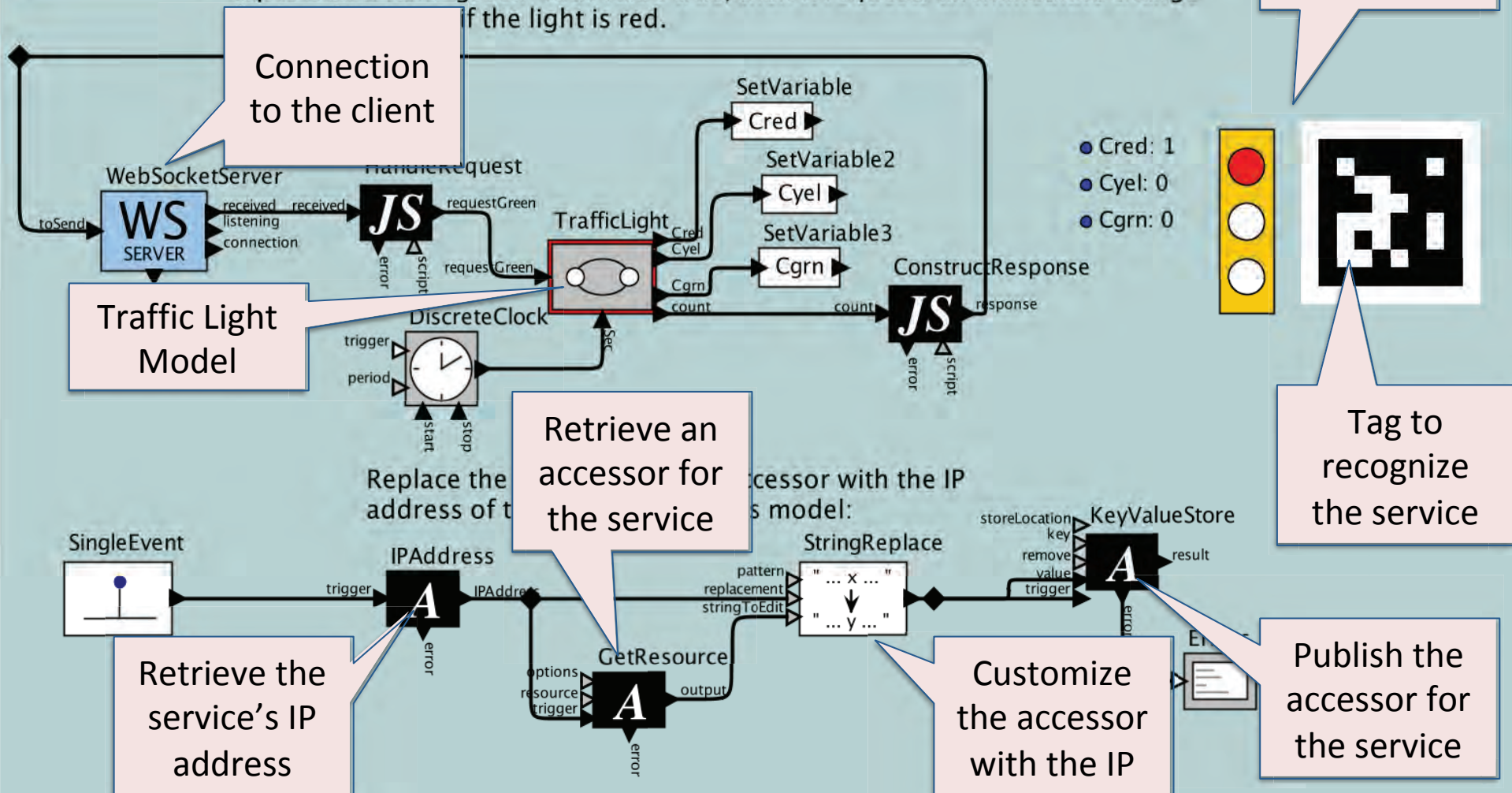


# Demo: A Service

DE Director

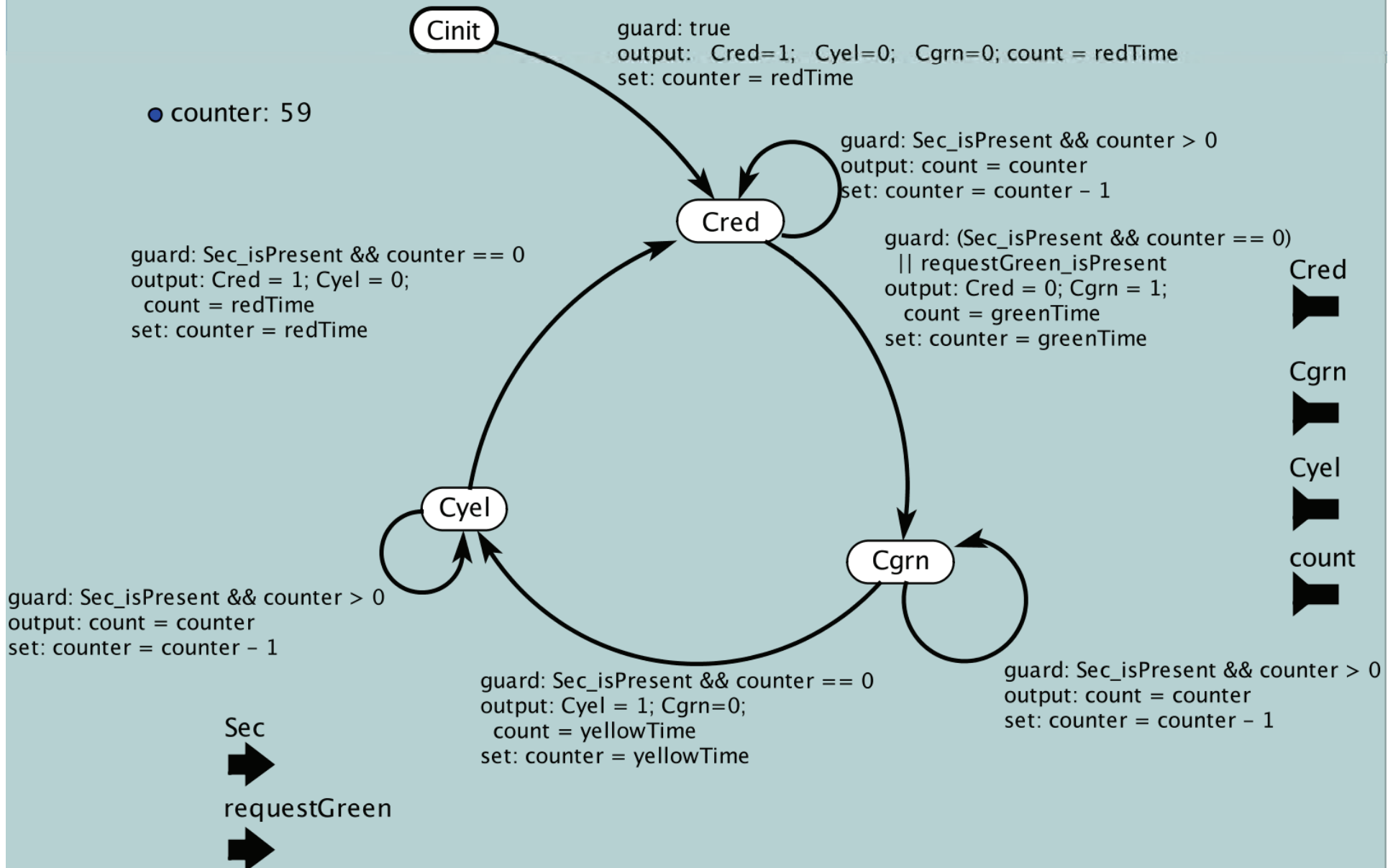


This model realizes a traffic light. It starts a web socket server and sends on it the time remaining until the next change in the light. If it receives a JSON input with a field "green" with value true, then it requests an immediate change of the light if the light is red.



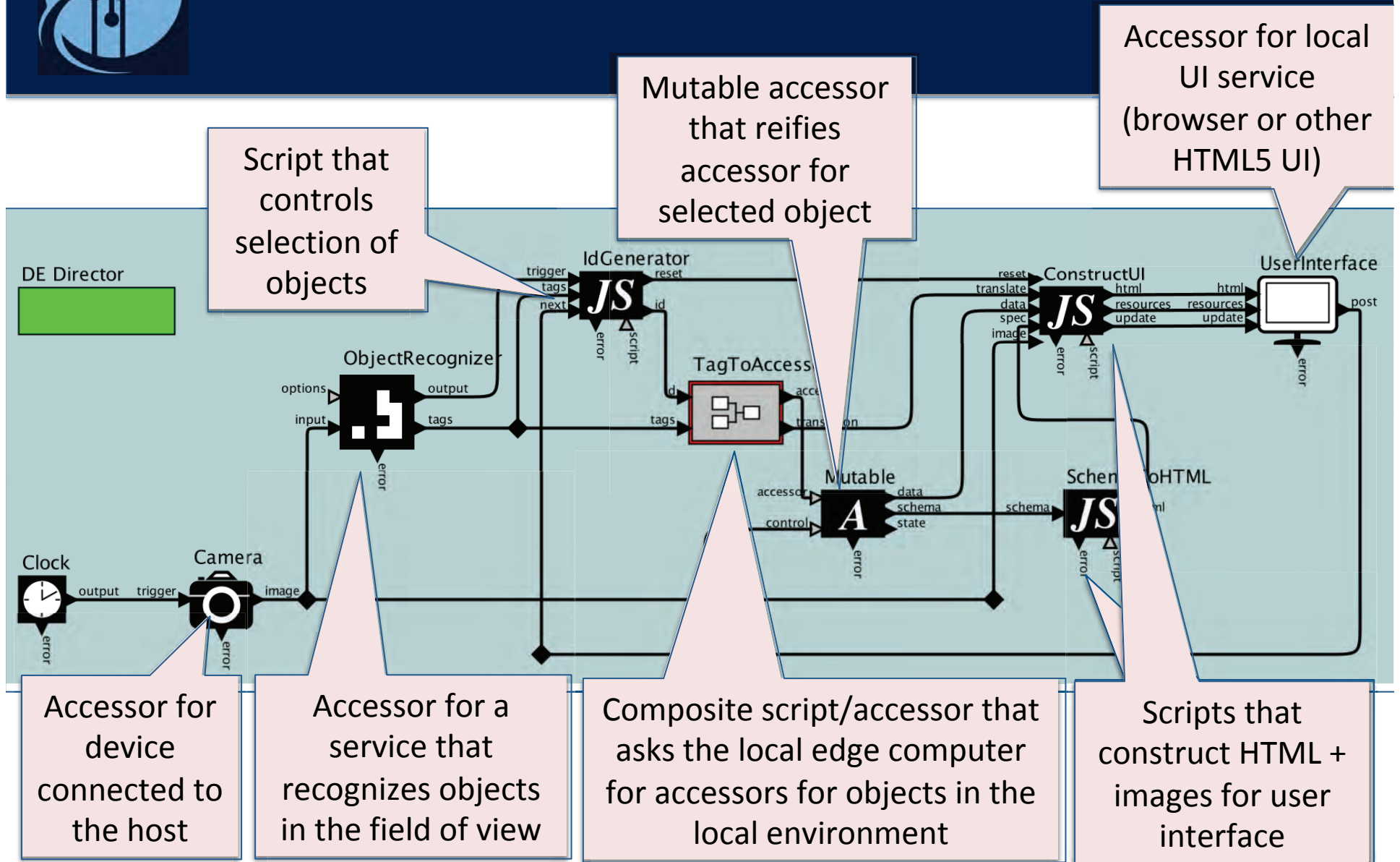


# Traffic Light Model





# Demo – Augmented Reality





# Example UI for Augmented Reality

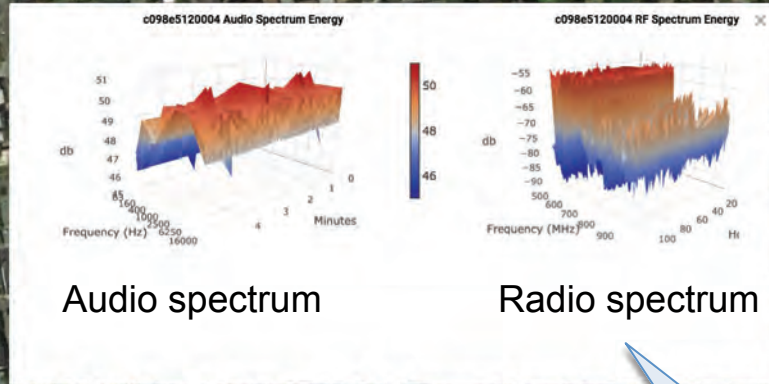


Exactly the same model being used at the TerraSwarm annual meeting to access a variety of sensors and actuators.

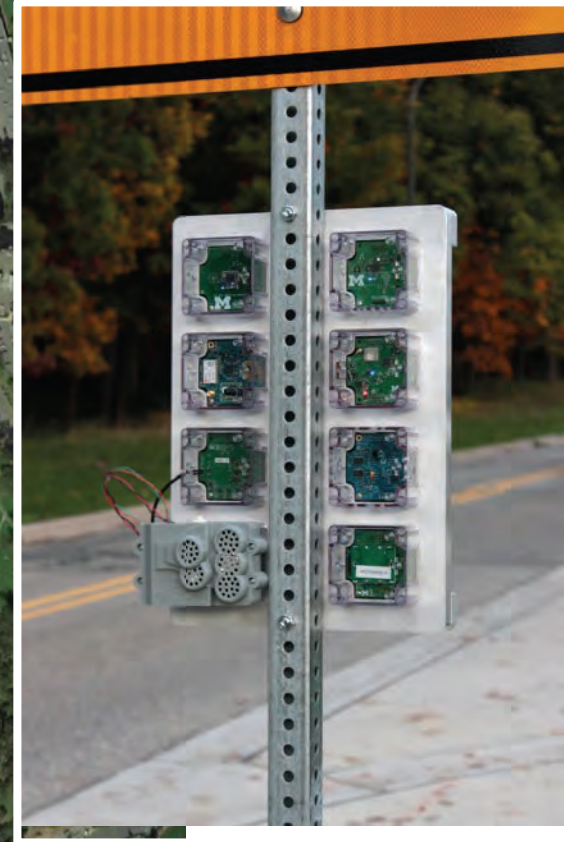
device	'BLEES'
pressure_pascals	100543.3
humidity_percent	54.69
temperature_celcius	20.96
light_lux	102
acceleration_advertisement	false
acceleration_interval	false
sequence_number	27938
id	'c098e530005a'
_meta	{ received_time: '2017-10-11T16:26:02.781Z', device_id: 'c098e530005a', receiver: 'ble-gateway', gateway_id: 'c0:98:e5:c0:00:1c' }



# Better User Interfaces



User interface is based on HTML 5. Supports extensive design.

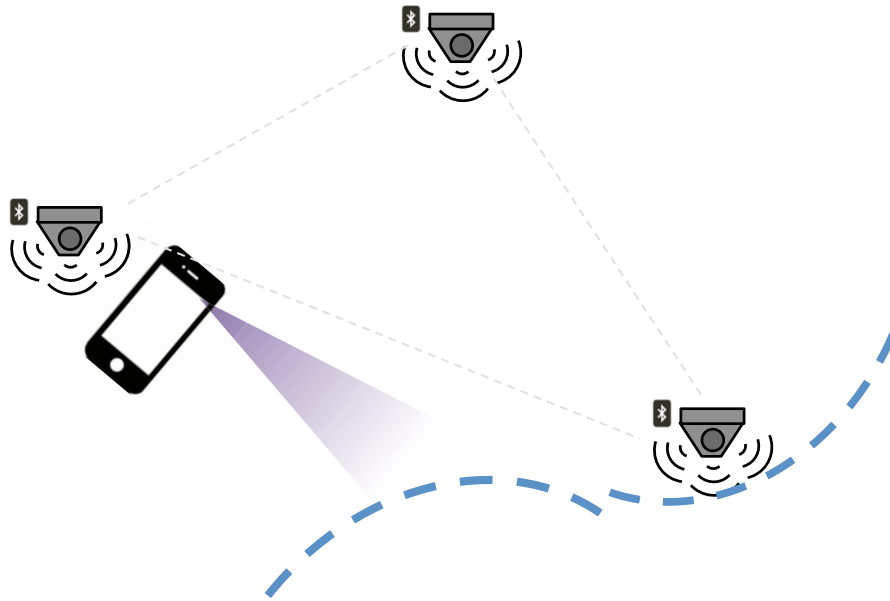


Visualization interface by Beth Osyk



# Tagless Odometry

Using object recognition, GPS, inertial sensors, WiFi fingerprinting, etc., we can hope to get rid of the tags.





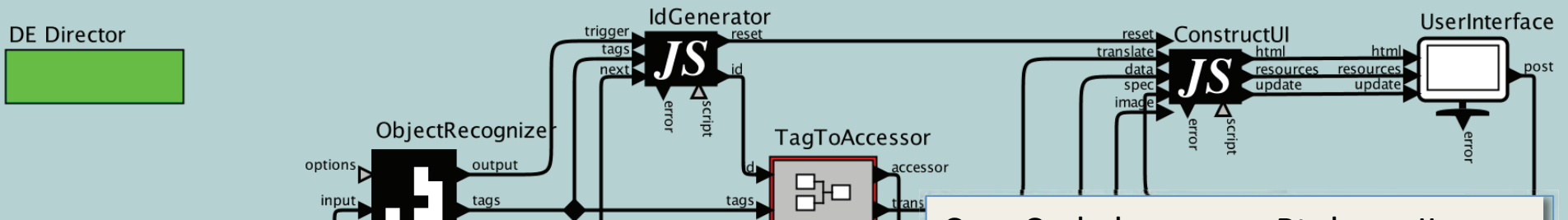
# Local Devices/Services

## Demonstrated at TerraSwarm Meeting

- BLEE sensors via MQTT (local comm.)
- BLEE sensors via GDP (global comm.)
- Powerblade sensors via MQTT (local comm.)
- Powerblade sensors via GDP (global comm.)
- Hue lightbulb via HTTP (local comm.)
- Robot via ROS Bridge (local comm.)
- Audio service via WebSocket (local comm.)
- MessageBoard via HTTP (local comm.)



# CapeCode: Swarmlet Development Our Programming Framework for the IoT



CapeCode leverages Ptolemy II, which provides a visual editor, strong type system, and lots of other infrastructure.

file:/ptll/ptolemy/configs/capecode/intro.htm

File Help



## CapeCode, Version 0.1

*The modular Internet of Things, based on Ptolemy II.*

- [Demos](#)
- [Documentation](#)
- [Authors](#)
- [Copyright](#)

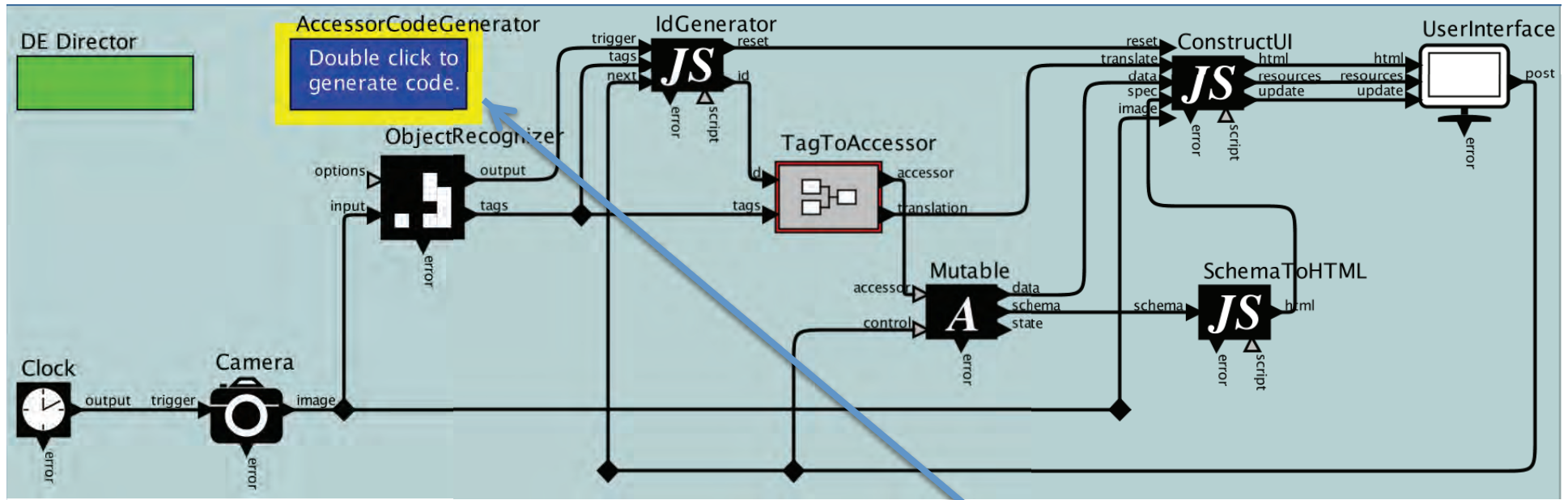
To create a CapeCode model, select File -> New -> Cape Code Model from the menu bar. See the [Ptolemy Project web page](#) for more information about the Ptolemy II.

(source: [Wikipedia](#))





# Deploying Swarmlets

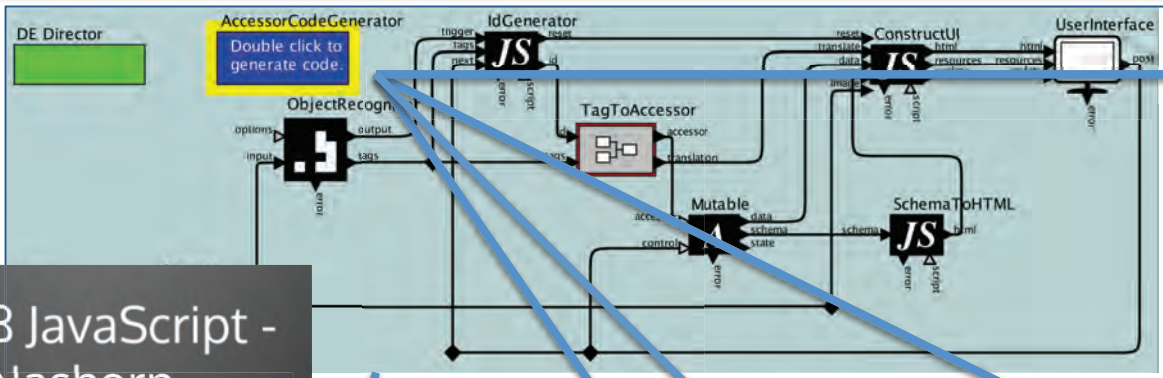


Code generator produces JavaScript files that can run on a lightweight JavaScript host.

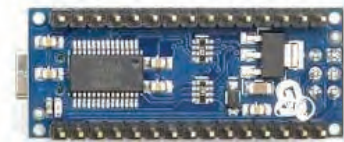
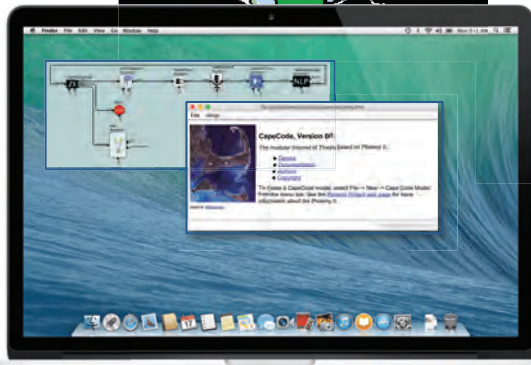
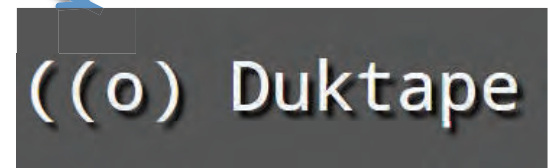


# Deploying Swarmlets

## Write Once, Run Everywhere?



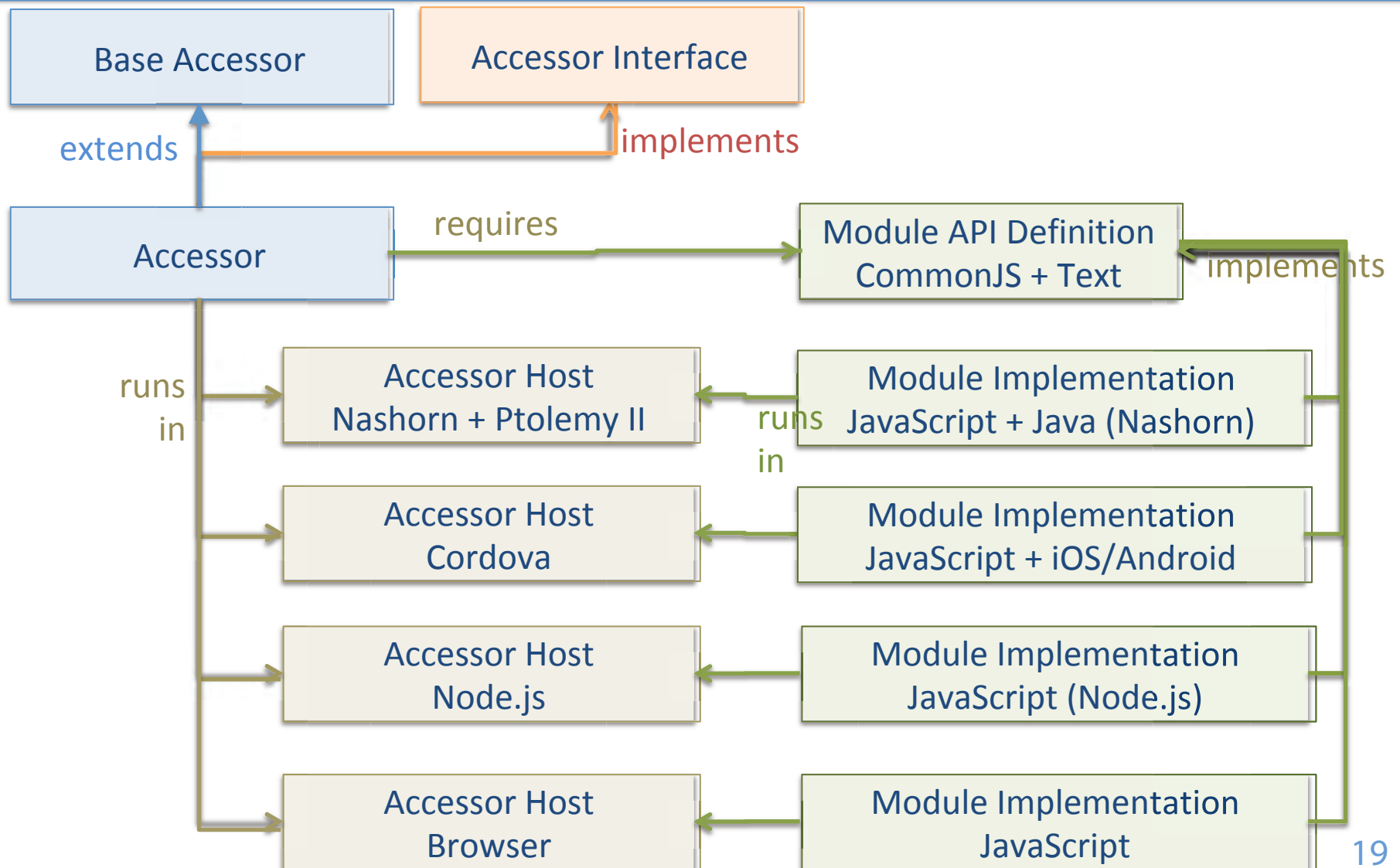
Java 8 JavaScript - Nashorn





# Accessor Architecture Version 1.0

<http://accessors.org>





# Browser Host

Firefox File Edit View History Bookmarks Tools Window Help 18 83% Mon 3:34 PM

localhost:8088/hosta/browser/test/testAccessorDirectory.html

## Accessors

- audio
- cameras
- devices
- gdp
- image
- localization
- net
- robotics
- RosPublisher
- RosSubscriber
- LocationRosPublisher
- services
  - GeoCoder
  - StockTick
  - Weather
- signals
- test
- utilities

### Accessor class: services/GeoCoder.js reveal code

**Modules required:** httpClient (Not supported by this host), querystring (Not supported by this host)

Retrieve a location given an address. The location is given as an object with two numeric fields, "latitude" and "longitude". For example, {"latitude": 37.85, "longitude": -122.26} is the location of Berkeley, California.

This accessor requires a "key" for the Google Geocoding API, which you can obtain for free at <https://developers.google.com/maps/documentation/geocoding/intro>.

This accessor does not block waiting for the response, but if any additional *address* input is received before a pending request has received a response or timed out, then the new request will be queued and sent out only after the pending request has completed. This strategy ensures that outputs are produced in the same order as the input requests.

**Author:** Edward A. Lee

**Version:** \$\$Id: GeoCoder.js 342 2015-10-31 15:48:43Z cxh \$#

#### Parameters

Name	Type	Value	Documentation
timeout	int	5000	No description found
outputCompleteResponseOnly	boolean	true	No description found
key	string	Enter Key Here	No description found

#### Inputs react to inputs

Name	Type	Value	Documentation
address			No description found

#### Outputs

Name	Type	Value	Documentation
response			No description found
location			No description found

Key challenge:  
Many accessors  
require modules  
that cannot be  
supported in a  
browser due to  
security  
constraints.



# Node.js Host

<https://accessors.org/wiki/Main/NodeHost>

## Install and invoke:

```
svn co https://repo.eecs.berkeley.edu/svn-anon/projects/terraswarm/accessors/trunk/accessors
cd accessors/web/hosts/node
node nodeHostShell.js
```

## Run the Node.js host:

```
Welcome to the Node swarmlet host (nsh). Type exit to exit, help for help.
nsh> var a = instantiate('a', 'test/TestGain')
Reading accessor at: /ptII/org/terraswarm/accessor/accessors/web/test/TestGain.js
Instantiated accessor a with class test/TestGain
undefined
nsh> a.initialize()
undefined
nsh> a.setParameter('gain', 4)
undefined
nsh> a.provideInput('input', 5)
undefined
nsh> a.react()
TestGain: inputHandler: input: 5 gain: 4
undefined
nsh> a.latestOutput('scaled')
20
nsh> exit
exit
```



# Challenges

- Multi-platform strategy
  - Maintaining compatible modules across hosts
- Regression tests
  - Need dummy devices and services
- Brittle designs
  - APIs for remote services change
  - Services and devices disappear
- Safety and security
  - Authorization (even without network connectivity)
- Privacy
  - Keep data local whenever possible
- Timing
  - Best-effort timing is not good enough
  - Cloud variability is too much for many applications.



# iCyPhy: The Home for this Research

## *Industrial Cyber-Physical Systems Center*

iCyPhy is a university-industry partnership to pursue pre-competitive research on design, modeling, and analysis techniques for cyber-physical systems, with emphasis on industrial applications. Topics:

- Hardware and software architectures
- Model-based design for CPS
- Verification, validation, and certification
- Highly dynamic networked systems
- The Internet of things (IoT)
- Safety, privacy, and security
- Synthesis and learning
- Localization and location-aware services
- Learning and optimization
- Safety-critical systems
- Human-in-the-loop systems.
- Systems-of-systems design
- Semantics of timed systems

<http://icyphy.org>

