A Web of Things Application Architecture - Integrating the Real-World into the Web

Dominique Guinard
Ph.D. Defense, ETH Zurich
Motivation

Why should we bring the Web and real-world devices together?

[flickr.com/photos/moragcasey]
An Increasing Number of Connected Smart Things…

A very large ecosystem of smart things, complex application development

From machines & home appliances…

and tagged objects.

to sensor networks…
Need for a Common Internet of Things Application Architecture

- Application development with smart things:
  - Requires expert knowledge:
    - Hardware/software heterogeneity
    - Lack of common application protocols
  - WSN [Mot2011]
  - RFID [Sch2008]

- Hypothesis: The Web (application archi. of the Internet) can be the application architecture of smart things as well.

- Research Question: «How can the Web be leveraged to ease the development of Internet of Things applications and bring it closer to non-specialists?»

References:


Contributions

- A Web of Things Application Architecture:
  - Adapt and leverage protocols, services and tools of the Web ecosystem
  - Foster a participatory application development:
    - Easier for specialists
    - Closer to Web developers (Web languages), tech-savvies (mashups) and end-users (browsers)
  - Evaluated for WSN and RFID:
    - Simplifies the development and deployment of applications
Web of Things Application Architecture

Simplifying Application Development in the Internet of Things
Web of Things Application Architecture

Layers

- Applications
- Accessibility
- Findability
- Sharing
- Composition

Web Cloud

Developers

End Users
- Tech-Savies
- Web Developers
- Specialists

Applications

- Energy
- Mobile Mash
- EPC Dashboard
- EnergieVisible
- Friends & Things
- EPC Find
- AmbientMeter
- EPCIS Webadapter
- Mobile Tag Pusher
- Web Sun SPOT
Device Accessibility Layer

- How do we make smart things accessible on the Web?
- Generic design process[^Gui2010] for smart things as Web resources:
  - REST[^Fie2000] and Resource Oriented Architectures[^Ric2007]


[^Ric2007]: Richardson, L., & Ruby, S. RESTful web services, O'Reilly Media.
GET, PUT
GET, DELETE
http://<DOMAIN>:<PORT>/genericNodes
/node1/sensors/temperature

GET, PUT
GET

http://<DOMAIN>:<PORT>/genericNodes

[{
   "name": "spot 1", "location": "kitchen", "id": 1299,
   "name": "spot 2", "location": "living room", "id": 1288,
   "name": "spot 3", "location": "...", "id": 1812
}]

Currently available Spots:

Spot 1 (in the Kitchen)
Spot 2 (in the Living room)
Spot 3 (...)

iGoogle
Resource Design -> Representations Design -> Interface Design -> Implementation Strategy

Web Server & RESTful Web API

- WebSockets
- Atom(Pub) Server
- REST Application Framework (JAX-RS / RESTlet, etc.)
- Gateway Core
- Discovery Unit
- Search Service
- Translation Service
- Storage Service

Core Services (left)

- BT Profile
- DPWS
- OPC-UA
- Bluetooth
- IEEE 802.15.4

Pluggable Services (right)

- App Drivers
- Transport Drivers

Device with native Internet support

IP Connectivity (6LoWPAN, Low-power WiFi, etc.)

Embedded Web Server & RESTful Web API

Smart Gateway

Clients

Direct Web Integration
Findability Layer

- Once smart things are accessible on the Web, how do we enable users to find the right service for their application?
- Enabling Smart Things to be indexed by search engines (lightweight metadata) [Gui2011]
- Local lookup and discovery infrastructure [Gui2010a, May2011]

[Gui2011] Guinard, D., Trifa, V., Mattern, F., & Wilde, E. From the Internet of Things to the Web of Things. Architecting the Internet of Things (pp. 97-129)


Once smart things are accessible and findable on the Web, how do we share them?

Social Web of Things [Gui2010b]
Social Access Controller (SAC)

- **Existing systems:**
  - Require dedicated access control lists (e.g., HTTP Digest or Basic Authentication)

- **Leverage social graphs of social networks:**
  - Are walled-gardens [Ber2009]
  - Allow sharing data, not services

- **Social Access Controller as proxy between clients and smart things**

Social Access Controller (SAC)

1) Authenticate with Social Networks
2) Retrieve trusted connections
3) Give Smart Thing / Gateway credentials
4) Crawl & find resources
5) Share resources
6) Advertise shared resources

Social Networks

Smart Gateways
- HTTPS
- HTTPS
- HTTPS
- HTTPS

Embedded Devices
Sharing in Friends and Things

Here you can see all Resources that you have shared with your friends or you can share new Resources. For existing Shares, you can display usage statistics in order to see whether it was worth sharing that Resource.

**Gateway:**
Select a gateway.

```
vswot.inf.ethz.ch:8081
```

**URL:**
Select a Resource to be shared. Loading all available resources might take some time, please be patient.

```
Temperature (/sunspots/Spot1/sensors/temperature)
```

**Social Network:**
Select a Social Network to display friends.

```
Facebook
```

**User:**
Select a friend to share a resource with.

```
Véronique Guinard
```

http://vswot.inf.ethz.ch:8091
/gateways/vswot.inf.ethz.ch:8081
/resources/sunspots/spot1/sensors/temperature
Once smart things are accessible, findable, shareable on the Web, how do we enable their easy composition by non-specialists, into new applications?

Physical Mashups [Gui2010, Gui2010c]


[Guinard2010c] Guinard, D. Mashing up your web-enabled home. ICWE 2010
From Web 2.0 Mashups to Physical Mashups

- **Web 2.0 Mashups:**
  - “Web applications generated by combining [...] disparate Web sources [...] to create useful new services” [Yu2008]
  - Ad-hoc applications accessible to a larger public

- **Physical Mashups:**
  - Composite Web applications involving smart things and virtual Web services
  - Three development approaches

---

Energie Visible: An Energy-Aware Mashup

- **Developers:**
  - Smart Meters as an RESTful Web API:
  - Mashup with any language supporting HTTP

- **Users:**
  - Used by several families around the world (Energie Visible)
EPC Mashup Dashboard: RFID Business Intelligence

- **Developers:**
  - RFID Readers & Data in a black-board approach
  - Wizard-based creation of Widgets
  - Merging Web data and real-world RFID data

- **Users:**
  - Simple Web page providing real-time business intelligence
  - Deployed at the SAP future store
Electronic Article Surveillance as a Physical Mashup


[Naef2009] Naef, L. *ClickScript a visual programming language in the browser*. Master Thesis, ETH Zurich
Selected Evaluations

Smart Gateways, Social Access Control & Developers’ Experiences

[flickr.com/photos/myfwc]
Smart Gateway vs Direct Web Integration

- **End-to-end HTTP**
  - 7000 (sequential) requests:
    - Avg.: 205 ms, SD: 127.8 ms
    - Min.: 96 ms, Max.: 8.5 sec

- **Smart Gateway**
  - 7000 (sequential) requests:
    - Avg.: 4.2 ms, SD: 3.7 ms
    - Min.: 2 ms, Max.: 111 sec

![Round-Trip-Time Depending on the Access Strategy](image-url)
Using Social Access Control

- 1000 requests on a Sun SPOT (Smart Gateway):
- Direct Access:
  - Avg.: 9 ms, SD: 2 ms
  - Max.: 40 ms, Min.: 6 ms
- Through SAC (Facebook), RTT:
  - Avg.: 218 ms, SD: 24 ms
  - Min: 204 ms, Max: 830 ms
  - Most of the RTT (140ms) due to social network login => caching
Ease of Use? Assessing the Developers’ Experience

- WS-* (WSDL, SOAP, etc.) as one of the most comprehensive alternatives (DPWS, DNLA, etc.)
- Performances were compared[^Yaz2009], not ease of use
- Study with 69 computer science students
- 2 applications:
  1. Android phone accessing a Sun SPOT featuring a WS-* API
  2. Android phone accessing a Sun SPOT featuring a RESTful Web API

Ease of Learning

- **RESTful Web API:**
  - 70% easy to very easy to learn
  - 63% fast to very fast to learn
Suitability by Use-Case and Guidelines

- REST and Web more adapted to foster adoption by non-specialists
- WS-* more adapted for high QoS/security requirements
Conclusions, Limitations and Outlook

What did we contribute? What are the current limitations?

[flickr.com/photos/brapke]
Contributions & Learnings

- **WoT Application Architecture & evaluation in WSN + RFID**
  - The Web can be leveraged and adapted as a smart thing application architecture
  - Eases the development & brings it closer to non-specialists
  - Unveils integration possibilities:
    - Browser, search engines, social networks, Web languages, mashups, etc.
Limitations & Outlook

- Not the best approach for every use-case:
  - Real-time, high QoS requirements, battery-life

- Pushing Internet and Web standards forward:
  - Lower foot-print (6LoWPAN)
  - Web push (HTML5 WebSockets, etc.)
  - Metadata for smart things

- Real-world evaluations:
  - Larger deployments, industrial trials (IPSO alliance)
  - Comparisons with other alternatives
  - Evaluating the mashups with more end-users
Thanks a lot for your attention

- Dominique Guinard
  - Contact details: [www.guinard.org](http://www.guinard.org)
  - Blog: [www.webofthings.com](http://www.webofthings.com)
  - Software: [www.webofthings.com/projects](http://www.webofthings.com/projects)
  - Publications [www.webofthings.com/publications](http://www.webofthings.com/publications)
Backup

Wait! There is a little more…
Complete Web of Things Application Architecture

Layers

- Applications
- Composition
- Sharing
- Findability
- Accessibility

Building-blocks

- Domain Specific Mashup Editors
- Physical Mashup Engine
- Widgets-Based Mashups
- Social Access Controller
- Physical Feeds Aggregator
- REST Crawler
- Translation Service
- Description Model
- Web Push
- RESTful Web API 4 Device

Developers

- End Users
- Tech-Savvies

Applications

- Web Developers
- Embedded Syst. Dev.
- Clickscript WoT
- Energy Mobile Mash
- EPC Dashboard
- EnergieVisible
- Friends & Things
- EPC Find
- AmbientMeter
- EPCIS Webadapter
- Mobile Tag Pusher
- Webenergy
- Sun SPOT Sync Driver
- HTTP Sun SPOT
Case-Studies

- Wireless Sensor Networks:
  - General Purpose Sensing Platform (Sun SPOTs)
  - Smart Metering Platform (Ploggs)

- Tagged Objects:
  - RFID global network:
    - EPC Network
Web-Enabling Smart Things in 4 Steps

- Based on REST [Fiel2000] and Resource Oriented Architecture [Rich2007].
- Applied it to smart things in [Gui2010].

**Resource Design**

**Representation Design**

**Interface Design**

**Implementation Strategy**

**Web of Things - Resource Temperature**

- Home
- Parent
- Refresh
- Atom Feed

GET, POST, PUT, DELETE, OPTIONS

Content Negotiation, Status Codes


[Richardson2007] Richardson, L., & Ruby, S. RESTful web services, O'ReillyMedia.
Identify Resources:
- Any component of an application that needs to be used and addressed.
- Link resources together
Smart things should offer different representations:
- HTML for browsability
- JSON for mashups
- XML for interoperability

Web of Things - Resource Temperature

- Home
- Parent
- Refresh
- Atom Feed

```json
{"resource":
   {"methods":["GET"],
    "name":"Temperature",
    "links":["/feed", "/rules"],
    "content":
    [{"description":"Current Ambient Temperature",
      "name":"Current Ambient Temperature",
      "value":"24.0",
      "unit": "celsius"}]
   }
}```
Leverage content negotiation:
  - Accept: application/json

Use the HTTP Verbs extensively:
  - GET, PUT, POST, OPTIONS, DELETE
  - GET /genericNodes/2/sensors/temperature
  - PUT /genericNodes/2/actuators/led/1

Map status codes:
  - 200 OK, 201 Created, 400 Bad Request, etc.

The presented design process can be automated
Automating the Design Process

- The design process can be semi-automated through an editor
- Auto-WoT [May2010a] project:
  - Generates the RESTful Web API
  - Wraps it in an OSGi module, loadable as a device driver in a Smart Gateway

Smart Gateway Mediated Interaction

Client ➔ EmbeddedWebServer
- HTTP Request: GET abstract-nodes/1/sensors/light
- getDataForURI(abstract-nodes/1/sensors/light)

EmbeddedWebServer ➔ GatewayCore
- doGetLight()
- wrappedLightValue
- HTTP Response: wrappedLightValue

GatewayCore ➔ DeviceDriver
- getLight()
- wrapAndSend(lightValue)

DeviceDriver ➔ AbstractNode1
- using protocol of abstract-node1
- lightValue

AbstractNode1 ➔ GatewayCore

15.08.2011
Pushing Data From Smart Things

Diagram:

- **GET /temperature**
  - 200 OK (35 celsius)
  - GET /temperature
  - 304 Not Modified
  - GET /temperature
  - 304 Not Modified
  - [...]
  - [...]
  - GET /temperature
  - 200 OK (135 celsius)

- **POST /temperature/subscription**
  - 200 OK, subscribed
  - work on something else
  - work on something else
  - [...]
  - [...]
  - new temperature 55
  - new temperature 135

- **WebClient**
- **SmartThing**
Pushing From Web Sockets: t-Pusher Service

**Diagram:**
- **WebClient** to **SmartThing**:
  - GET /temperature
  - 200 OK (35 celsius)
  - GET /temperature
  - 304 Not Modified
  - GET /temperature
  - 304 Not Modified
  - [...]
  - GET /temperature
  - 200 OK (135 celsius)

**Pusher Service**
- POST http://<SmartThing.URI>/topic/temperature
- ["Temperature Getter","value":"35.5"]
- POST http://<Callback-URI>
- Connection: Upgrade
- Upgrade: WebSocket
- Challenge Response
- push to WebSocket
- push over WebSocket
- ["Temperature Getter","value":"35.5"]
Performance Evaluation

- 7000 (sequential) requests:
  1. End-to-end HTTP
     - Avg.: 205 ms, SD: 127.8 ms
     - Max.: 8.5 sec (single request)
     - Min.: 96 ms
  2. Syn-based Smart Gateway:
     - Avg.: 4.2 ms, SD: 3.7 ms
     - Max.: 111 sec
     - Min.: 2 ms
Scalability / Concurrency Evaluation

- Smart Gateway sync-based approach
  - Scales better:
    - Strongly depends on the Web server implementation
  - Provides more aged data.
**Discovery by Crawling**

- Thanks to the Accessibility Layer (REST), smart things can be crawled
- **Shortcomings:**
  - Rough descriptions (UIs)
  - Does not enable automated mashup integration

```java
crawl(Link currentLink) {
    new Resource() r;
    r.setUri = currentLink.getURI();
    r.setShortDescription = currentLink.text();
    r.setLongDescription = currentLink.invokeVerb(GET).extractDescriptionFromResults();
    r.setOperations = currentLink.invokeVerb(OPTIONS).getVerbs();
    foreach (Format formats: currentFormat) {
        r.setAcceptedFormats = currentLink.invokeVerb(GET).setAcceptHeader(currentFormat);
    }
    if (currentLink.hasNext())
        crawl(currentLink.getNext());
}
```
Metadata Description: Smart Things Description Model

- Description model for:
  - Findability through existing search engines
  - Automatic integration into mashups
- Crawling algorithm gathers the minimal information:
  - Read URLs
  - GET & OPTIONS on resource
  - Content-negociation
- Compound of microformats: hProduct, hCard, geo, hReview
Local Lookup and Discovery Units (LLDUs)

- Embedded Web Server & REST API
  - REST Application Framework (JAX-RS / RESTlet, etc.)
  - Query Web User Interface
  - Registry Service
  - Lookup Service
  - Infrastructure Service
  - Discovery Service

- OSGi (Declarative Services)
  - BT Profile
  - DPWS
  - [...]
  - Device Drivers
    - Bluetooth
    - IEEE 802.15.4
    - [...]

- LLDU Services
- Query Augmentation
- STD Model Translation Service

- Findability

Dominique Guinard
LLDU Deployment
Infrastructure Performance Evaluation

- LLDU tree:
  - Depth of 6
  - 11 virtual LLDUs
  - 1 Physical LLDU (LLDU + Smart Gateway)
    - 2 Sun SPOTs
    - 61 virtual services

- LLDU on a PC (2.4 GHz, 2 GB of RAM):
  - 10000 keywords queries ("light")
  - Avg: 619 ms
  - Min: 12 ms
  - Max: 3735 ms
Query Extension Evaluation

- 17 neutral developers:
  - Each describes one device and at least two services (based on the STD model)
  - RFID readers, robots, smart meters, etc.
- Search keywords provided by 7 IT people:
  - No augmentation: 70%
  - Wikipedia: ~90%
  - Yahoo Web Search: 95-100%
  - Optimum: Yahoo with 5-10 added keywords (95%).
Social Access Controller (SAC) Architecture
Friends and Things: User Interface

Here you can see all Resources that have been shared with you. You can either open a resource in a new browser window, display it directly in friends & things or you can make custom RESTful HTTP requests to it. Further, you can register Feeds in order to send updates to it in regular time periods.

GET /gateways/localhost:8082/resources/EnergyMonitor

POST /gateways/localhost:8082/resources/EnergyMonitor
Request Data: URL-encoded data to be sent to the resource, e.g.: key1=value1

Here you can see all Resources that you have shared with your friends or you can share new Resources. For existing Shares, you can display usage statistics in order to see whether it was worth sharing that Resource.

Gateway:
Select a gateway: localhost:8082

URL:
Select a Resource to be shared. Loading all available resources might take some time, please be patient.

EnergyMonitor (/EnergyMonitor)
Overall load of the current place (0-100) (/EnergyMonitor/load.html)
List all Ploggs (/EnergyMonitor/ploggs.html)
Kettle (/EnergyMonitor/ploggs/Kettle.html)
EnergyMonitor/ploggs/Kettle/status.html
All Ploggs (/EnergyMonitor/ploggs/all)

Social Network:
Select a Social Network to display friends.
Accessing Shared Smart Things: WSN

1) Use resource URI
2) Authenticate with Social Networks
3) Access smart thing
4) Redirect results

Social Networks

Smart Gateways

Smart Things

HTTPS

Accessing Shared Smart Things: RFID

From Web 2.0 Mashups to...

- **Web 2.0 Mashups:**
  - “Web applications generated by combining [...] disparate Web sources [...] to create useful new applications or services” [Yu2008]

- Composite applications with:
  - Lightweightness and simplicity
  - Accessibility to larger public
  - Prototypical or opportunistic nature

Manual Mashups with Energie Visible
Adapting a Mashup Editor

$.ajax({
  url: "http://" + ip + "/sunspots/" +
  name + "/sensors/temperature",
  type: "GET", dataType: "json",
  success: function(result){
    var temperature =
      result.resource.getters[0].value
    state.outputs.item(0).setValue(temp)
    component.finishAsync();
  } [...]})

[Naef2009] Naef, L. ClickScript a visual programming language in the browser. Master Thesis, ETH Zurich
Building Mashup Editors: Physical Mashups Framework

- Requirements:
  - Support for event-based mashups
  - Support for dynamic building-blocks
  - Support for non-desktop platforms
  - Support for application specific editors
Physical Mashup Lab Deployment & Mobile Apps

- Webcam
- Mobile Webapp
- Mobile Phone + QR Code
- RFID Gate Reader

End-User Development with Mashup Editors
Mobile Energy Mashup Editor

- Android-based mashup editor for mashable homes
- Uses the Physical Mashup Framework RESTful Web API
- Findability Layer for automatic building-blocks generation
Mobile Energy Mashup Editor cont’d

- Lets end-users create simple rules to optimize their energy consumption:
  - Turn the heating on only when I am driving home and temp < 18 deg.
  - Turn off the lights when sun light is strong enough.
- Android-based mashup editor.
- Uses the Physical Mashup Framework RESTful Web API.
- Uses the Findability Layer for automatic building-blocks generation.
Mobile Energy Mashup Editor cont’d

- JSON reflection to generate adapted UIs.
REST vs WS-*: Guidelines

<table>
<thead>
<tr>
<th>Requirement</th>
<th>REST</th>
<th>WS-*</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mobile &amp; Embedded</td>
<td>+</td>
<td>-</td>
<td>Lightweight, IP/HTTP support</td>
</tr>
<tr>
<td>Ease of use</td>
<td>++</td>
<td>-</td>
<td>Easy to learn</td>
</tr>
<tr>
<td>Foster third-party adoption</td>
<td>++</td>
<td>-</td>
<td>Easy to prototype</td>
</tr>
<tr>
<td>Scalability</td>
<td>++</td>
<td>+</td>
<td>Web mechanisms</td>
</tr>
<tr>
<td>Web integration</td>
<td>+++</td>
<td>+</td>
<td>Web is RESTful</td>
</tr>
<tr>
<td>Business</td>
<td>+</td>
<td>++</td>
<td>QoS &amp; security</td>
</tr>
<tr>
<td>Service contracts</td>
<td>+</td>
<td>++</td>
<td>WSDL</td>
</tr>
<tr>
<td>Adv. security</td>
<td>-</td>
<td>+++</td>
<td>WS-Security</td>
</tr>
</tbody>
</table>