Internet of Things (IoT): Middleware

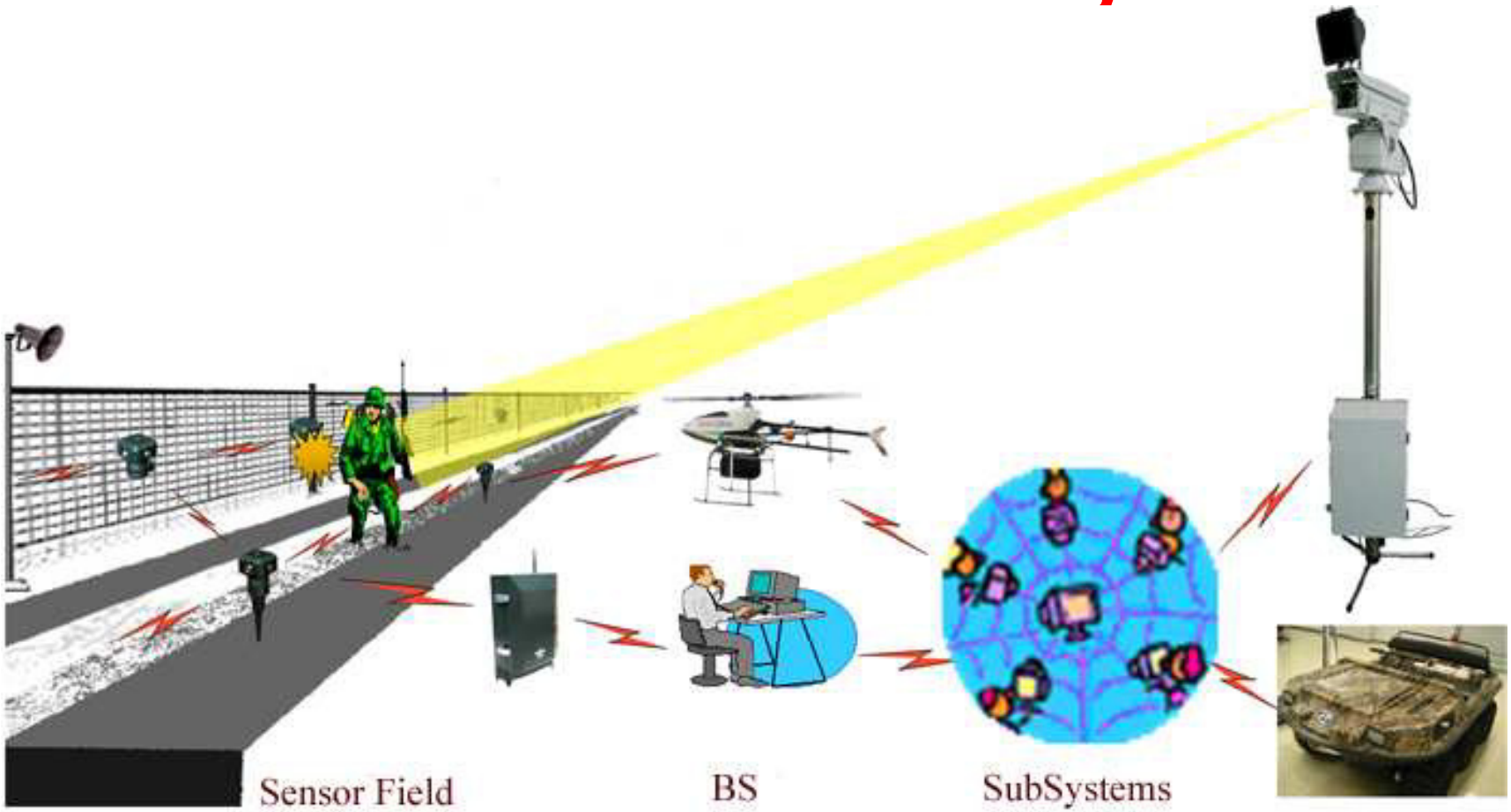
Roch Glitho, PhD

Associate Professor and Canada Research Chair

My URL - http://users.encs.concordia.ca/~glitho/

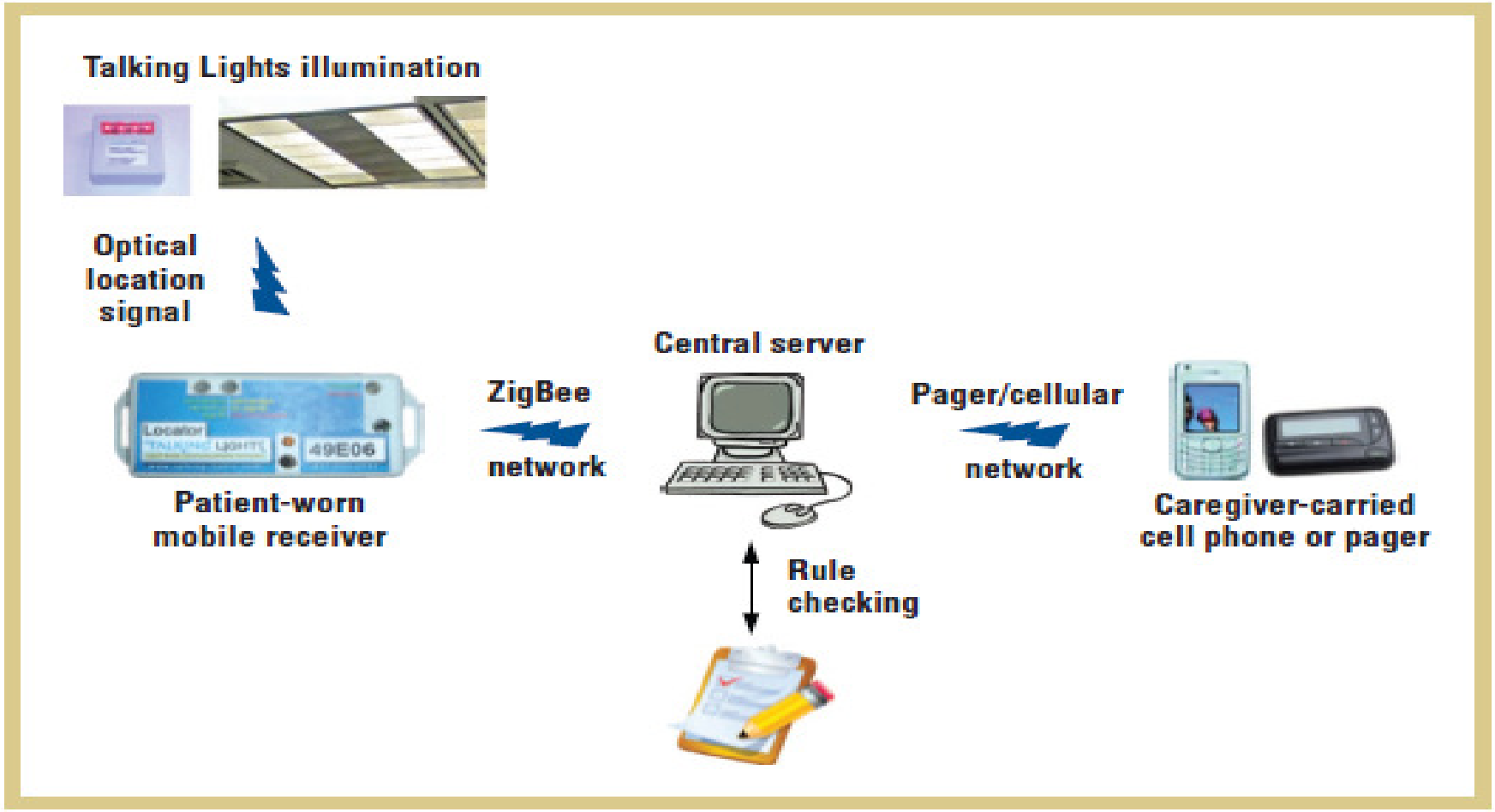
Concordia University
Engineering and Computer Science
Concordia Institute for Information Systems Engineering

A Fence Surveillance System



Y. Kim et al, Autonomics '08 Proceedings of the 2nd International Conference on Autonomic Computing and Communication Systems

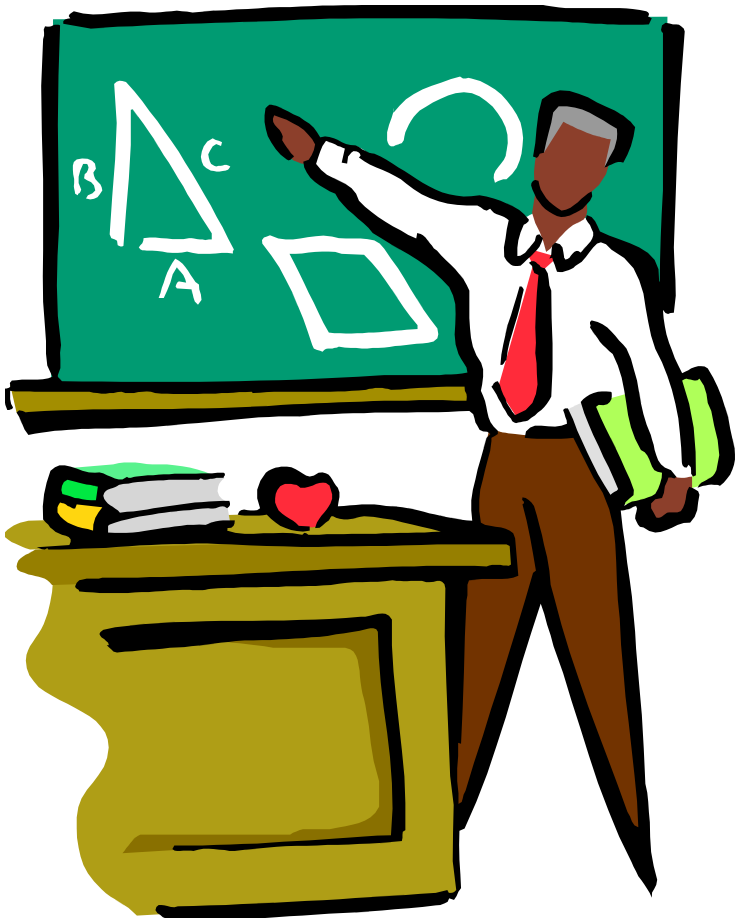
Escort: Safety Monitoring for People Living with Alzheimer



D.M Taub et al, The Escort System: A Safety Monitor for People Living with Alzheimer's Disease, IEEE Pervasive Computing, April- June 2011

The Prototype (A screen shot of an intrusion)

Introduction to Middleware

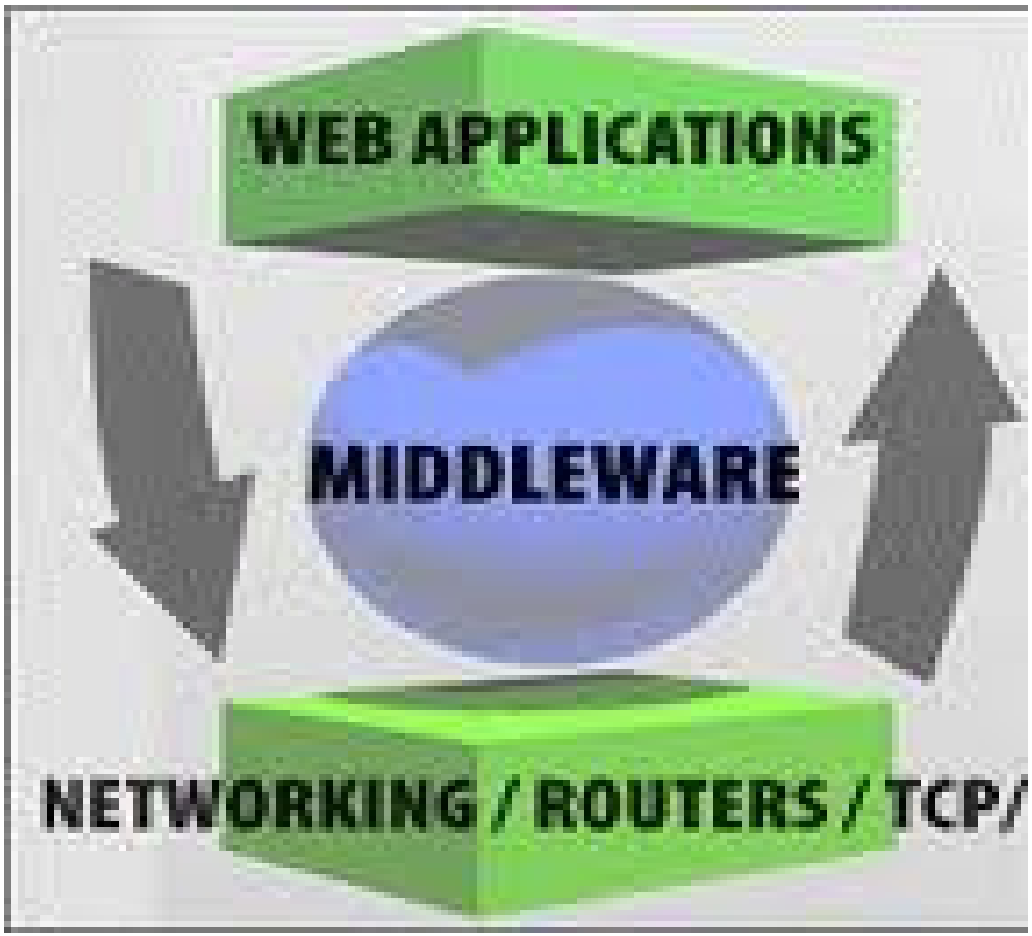


- Informal definition and motivations
- An illustration: The middleware for the application planned for this workshop

Middleware


useful applications
noise
useful applications
noise
useful applications
noise
2400_1
MICAZ Mote

Middleware



Why do We need Middleware?

Rapid application development

- No need to know the low level specific of wireless sensors
- Generic services that span applications

Easy integration with existing applications

- Requires the use of technologies such as Web services

Possibility to easily support of a wide range of applications (horizontal approach)

- Will prevent silo approach

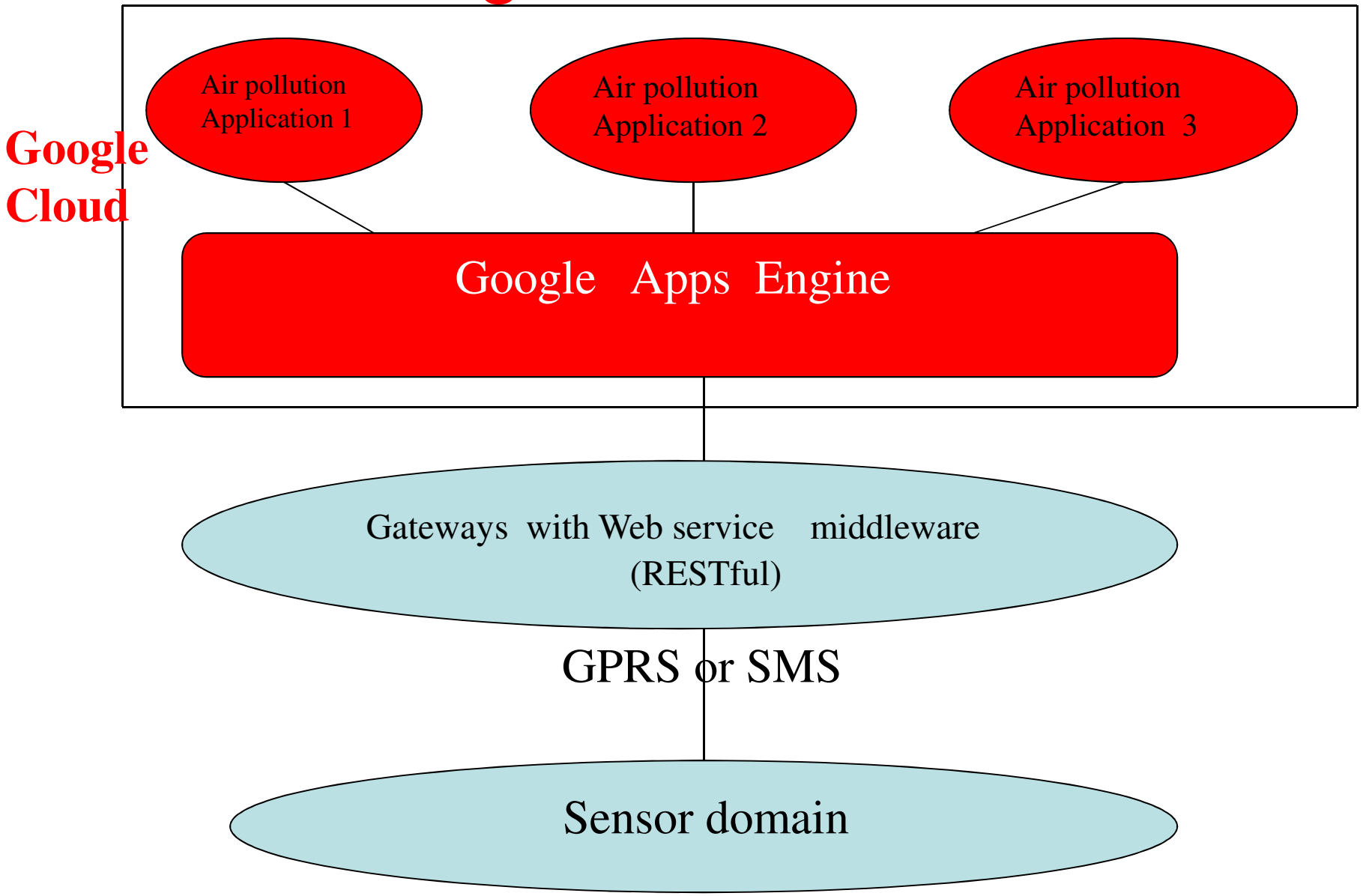
The application (Minimal expectations)

1. A Web page accessible from anywhere, anytime and which displays raw pollution data from a fixed place in Porto Novo and at a given periodicity
    - End-users will have no control over the Web page
    - Place and periodicity will be hard coded

Notes

- Flexibility will be added after the minimal requirements are met, e.g., ability for users to select on the page
    - Places
    - Periodicity
    - Data format

The High Level Architecture



Implementation assumptions and principles

Assumptions

- Gateway implemented on a centralized server with 2 interfaces (SMS/GPRS and Internet)
- There is an Internet connection where the gateway is deployed
- There is no Internet connection at the specific sites where the sensors are deployed

Principles

- A simple pull mechanism – It is highly inefficient but much easier to implement
  - A push mechanism will be considered after the implementation of the pull mechanism

Required sub-set of functionality

Application side

- A RESTful client application in Google Apps Engine that can send a request to a server located outside the Google world (i.e. a server located on the Gateway), process the response, then post the data on a Web site. The application will be accessible as SaaS
- The challenges
  - How to develop the application using the APIs offered by Google Apps Engine
  - How to make the application located on Google Apps Engine communicate with a functional entity located outside Google Apps Engine
  - How to publish the application as SaaS using Google Apps Engine

Required sub-set of functionality

Gateway side

- A client/server application
  - Acts as server towards Google Apps Engine and acts as client towards the actual wireless sensors.
- The challenges
  - How to map the REST request from Google Apps Engine onto the proprietary command sets supported by the sensors
  - How to communicate with the sensors using the appropriate GPRS/SMS APIs
  - How to communicate with Google Apps Engine

Required sub-set of functionality

Wireless sensor side

- We assume that the software will be provided by Marco

Work to be done by the 2 students prior to the workshop

1. Each student should select (or be assigned) the portion on which he should focus (Application part vs. gateway part)
2. Get familiar with the appropriate tool kits, i.e.
   - Google Apps Engine + RESTFUL Web service tool kit for the one focusing on application
   - RESTFul Web service tool kit + APIs for GPRS/SMS for the one who will work on the gateway side.

   Note: Preston will email the specification of the RESTFul Web services to the student during the week.

Cloud Middleware

- On Cloud Computing
- The State of Art
- Vision and Research Directions

Cloud Computing

Cloud Computing

Cloud computing: Easy-to-access pool of virtualized resources (e.g., servers, applications, development platforms).

Cloud Computing

- Resource utilization is adjusted to variable load through dynamic.
- Customers pay per usage.

Cloud Computing

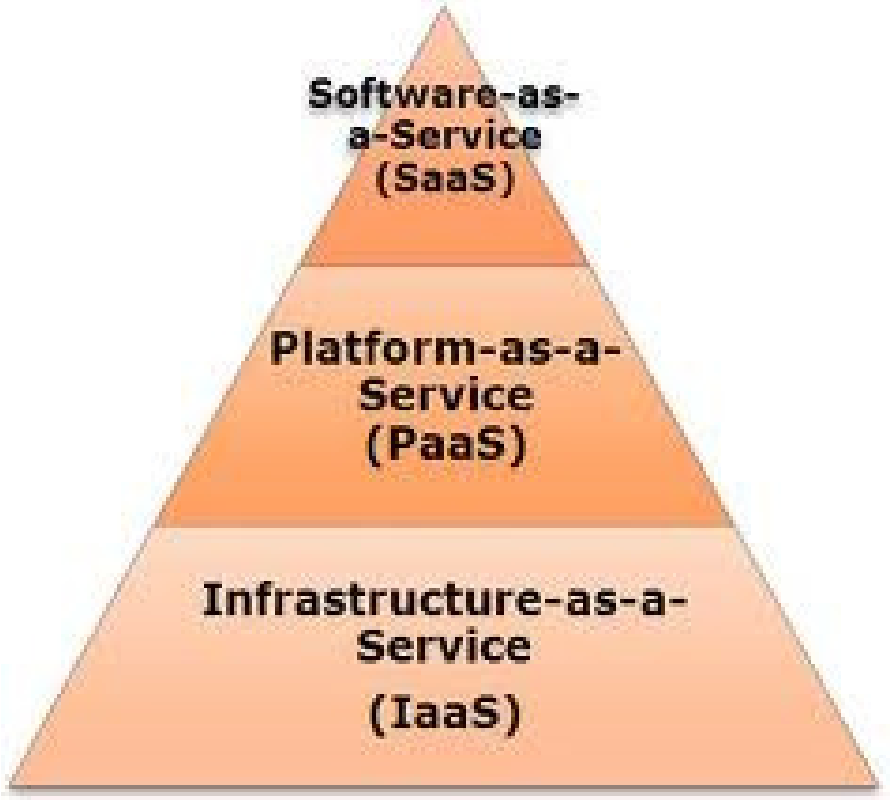
Examples:

- Data base in clouds

- Telecommunication base station in clouds

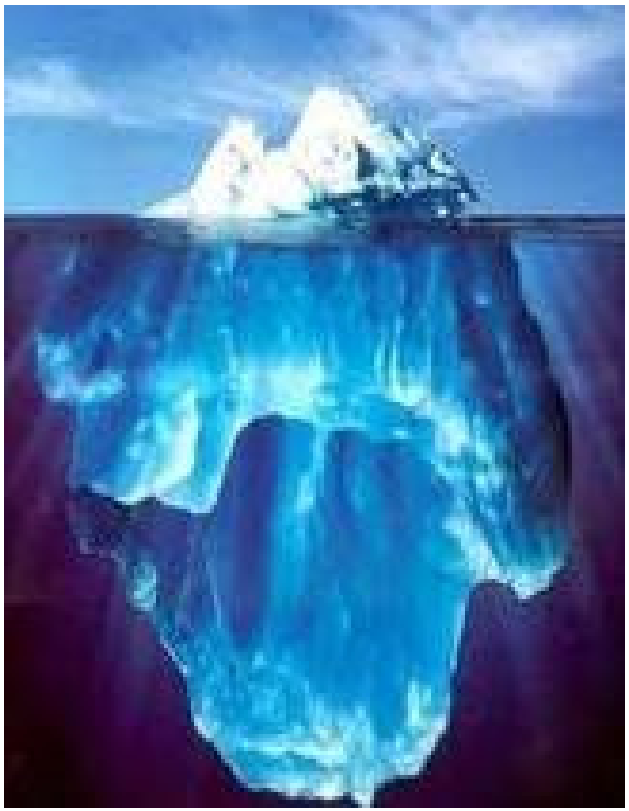
On Cloud Computing

Layers



Layers

Software as Services (SaaS): the tip of the iceberg (End-user perspective)



Layers

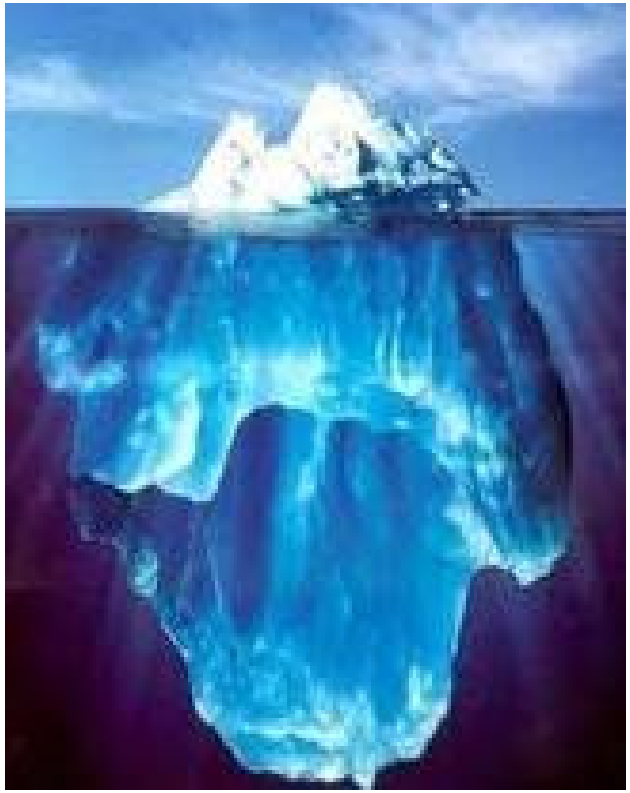
Software as Services (SaaS): the tip of the iceberg (End-user perspective)

Applications offered by service providers and residing in the cloud

- Pay per use basis
- Accessible by end-users (and eventually other applications)
- Relatively simple applications so far
- An example of a more complex application:
- Remedyforce for IT helpdesk management (SalesForce Portfolio)

Layers

Platforms as a Service (PaaS): immersed part I (Service provider perspective)



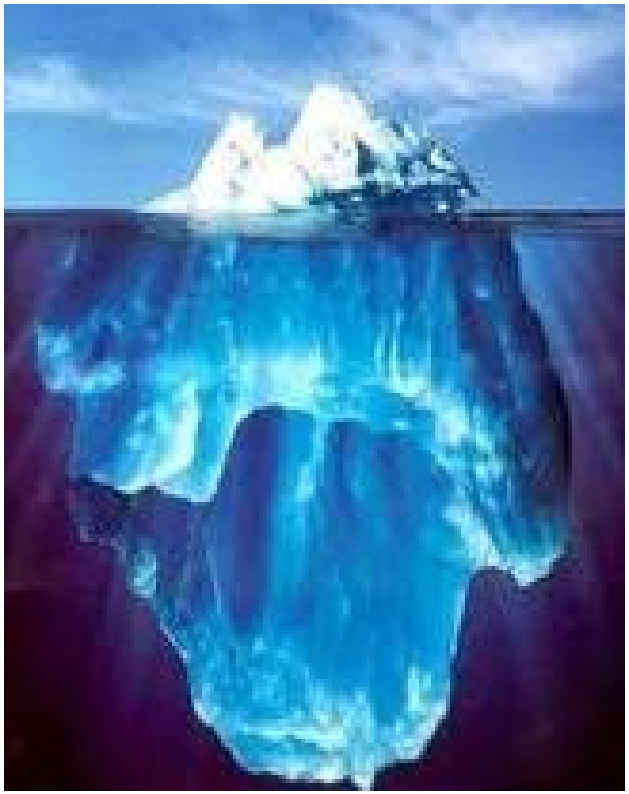
Layers

Platforms as a Service (PaaS): immersed part I (Service provider perspective)

- Platforms used for the development and management of the applications offered as SaaS to end-users (and other applications)
  - Examples:
  - Google Apps Engine (Freeware)
  - Microsoft Azure (Microsoft)
  - Cloud Foundry (open source)
    - Can be extended (currently being extended in my lab for telecoms, IoT and others)

Layers

Infrastructure as a Service (IaaS): immersed part II: Infrastructure provider perspective)



Layers

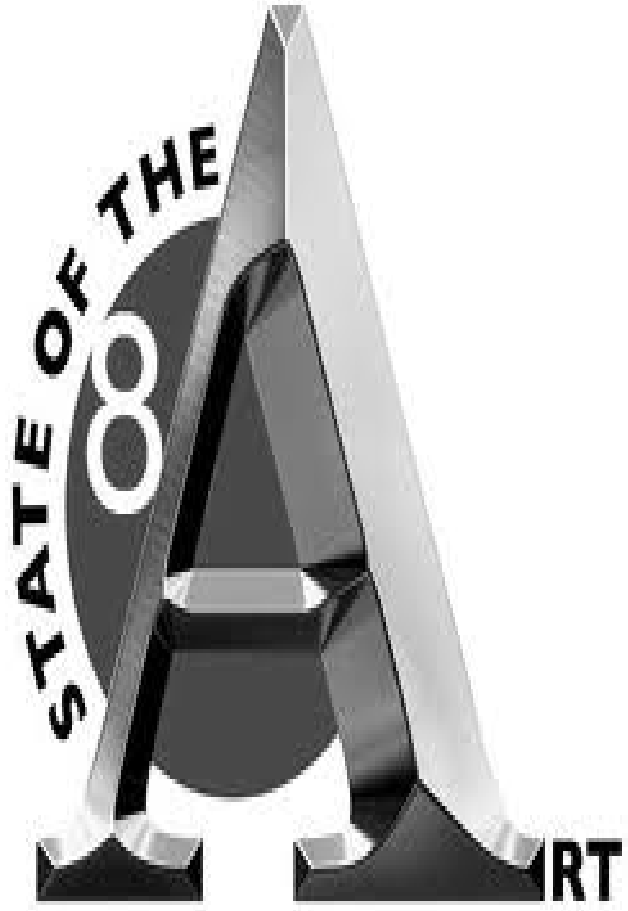
Infrastructure as a Service (IaaS): immersed part II: Infrastructure provider perspective)

Virtualized resources (CPU, memory, storage and eventually service substrates) used (on a pay per use basis) by applications

- Examples
- IBM Blue Cloud
- Amazon EC2

- Note: Focus so far in the literature: CPU, memory, storage –

Clouds as IoT applications Enablers: The State of the Art



- Use of cloud processing/computation power
- (Straightforward)

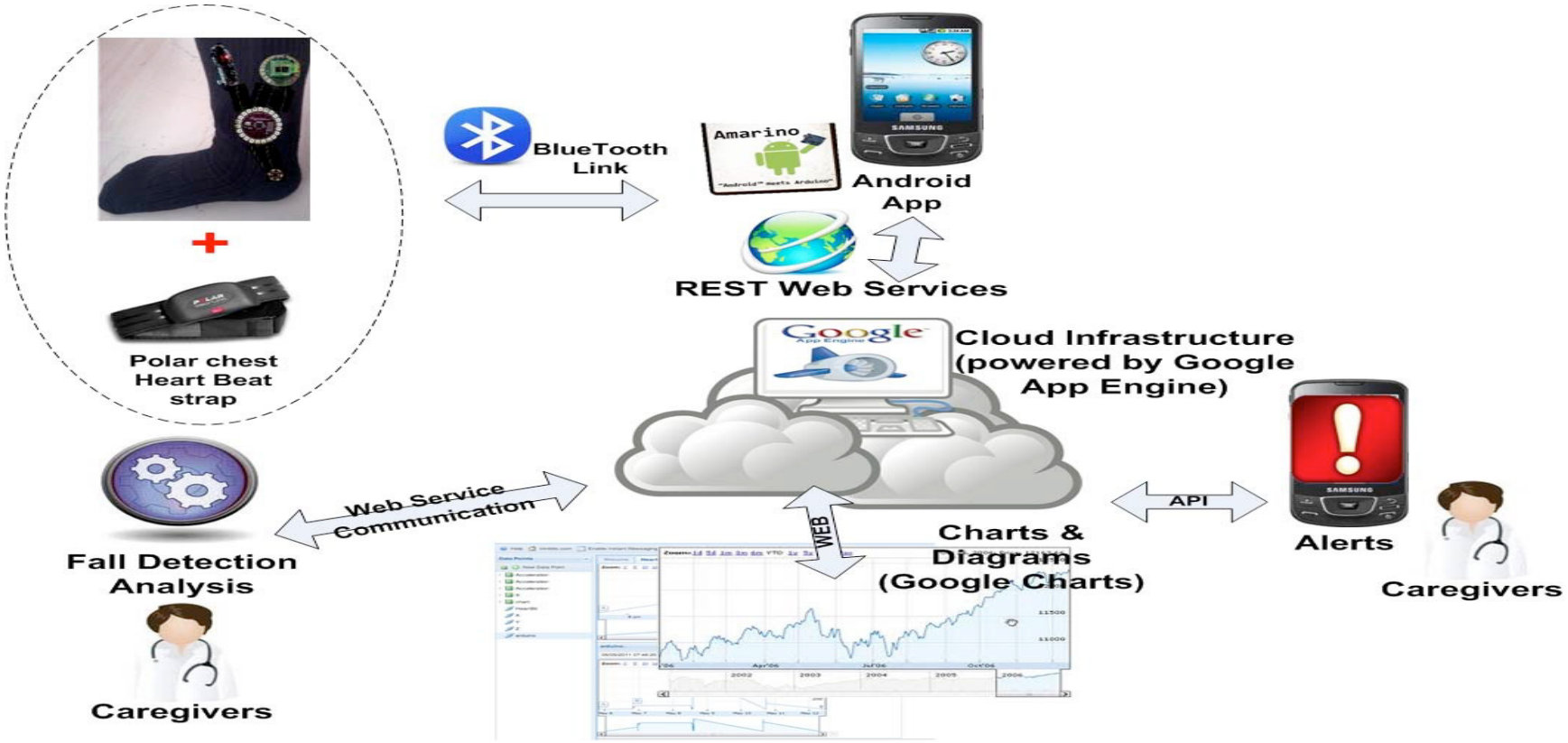
- Applying cloud fundamentals to IoT applications provisioning
- (More complex)

Use of Cloud Processing and Storage Power

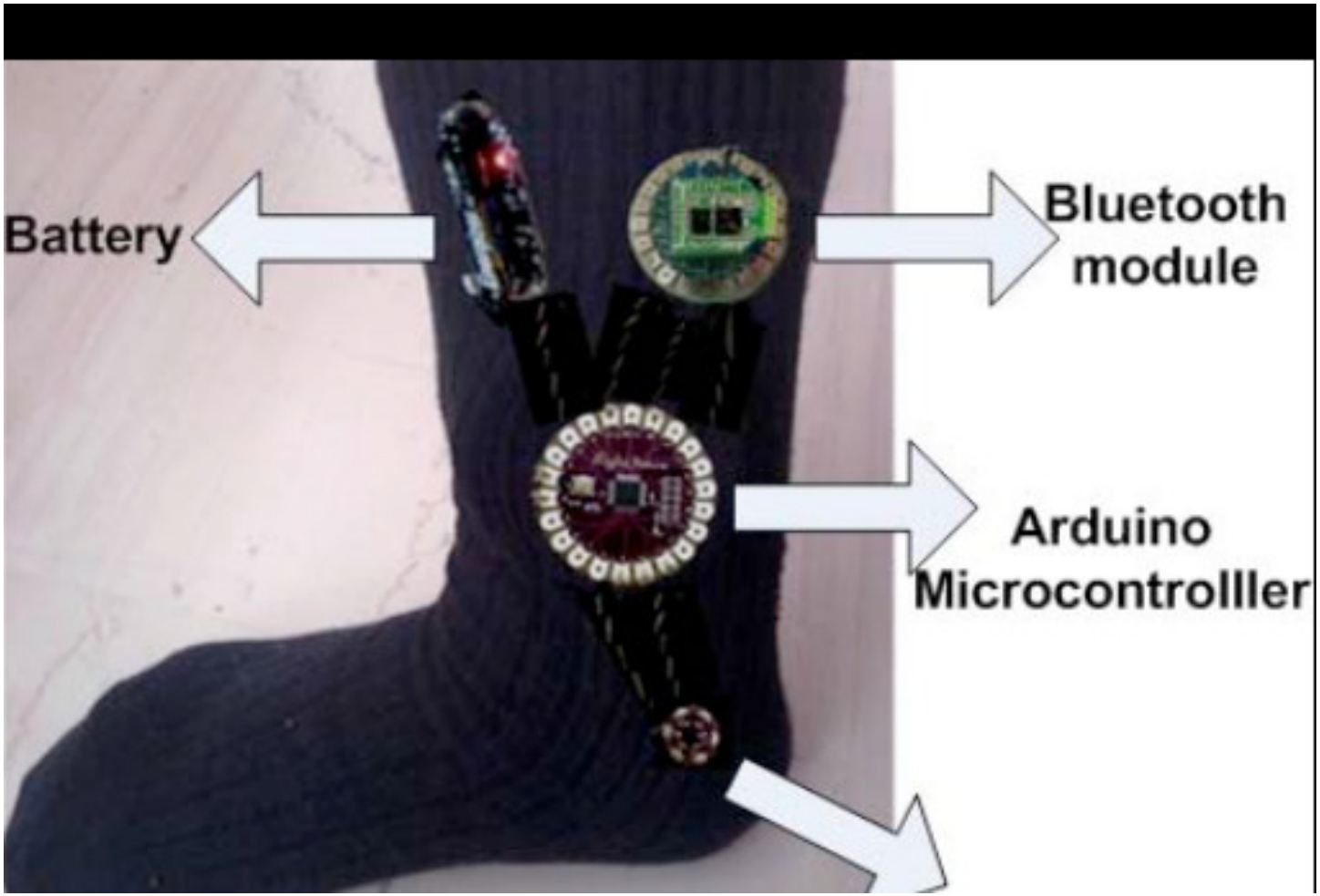
Managing Wearable Data Through Cloud

C. Doukas and I. Maglogiannis, Managing Wearable Sensor Data through Cloud Computing, 2011 Third International Conference on Cloud Computing Technology and Science

Managing Wearable Data Through Cloud



Managing Wearable Data Through Cloud



Managing Wearable Data Through Cloud

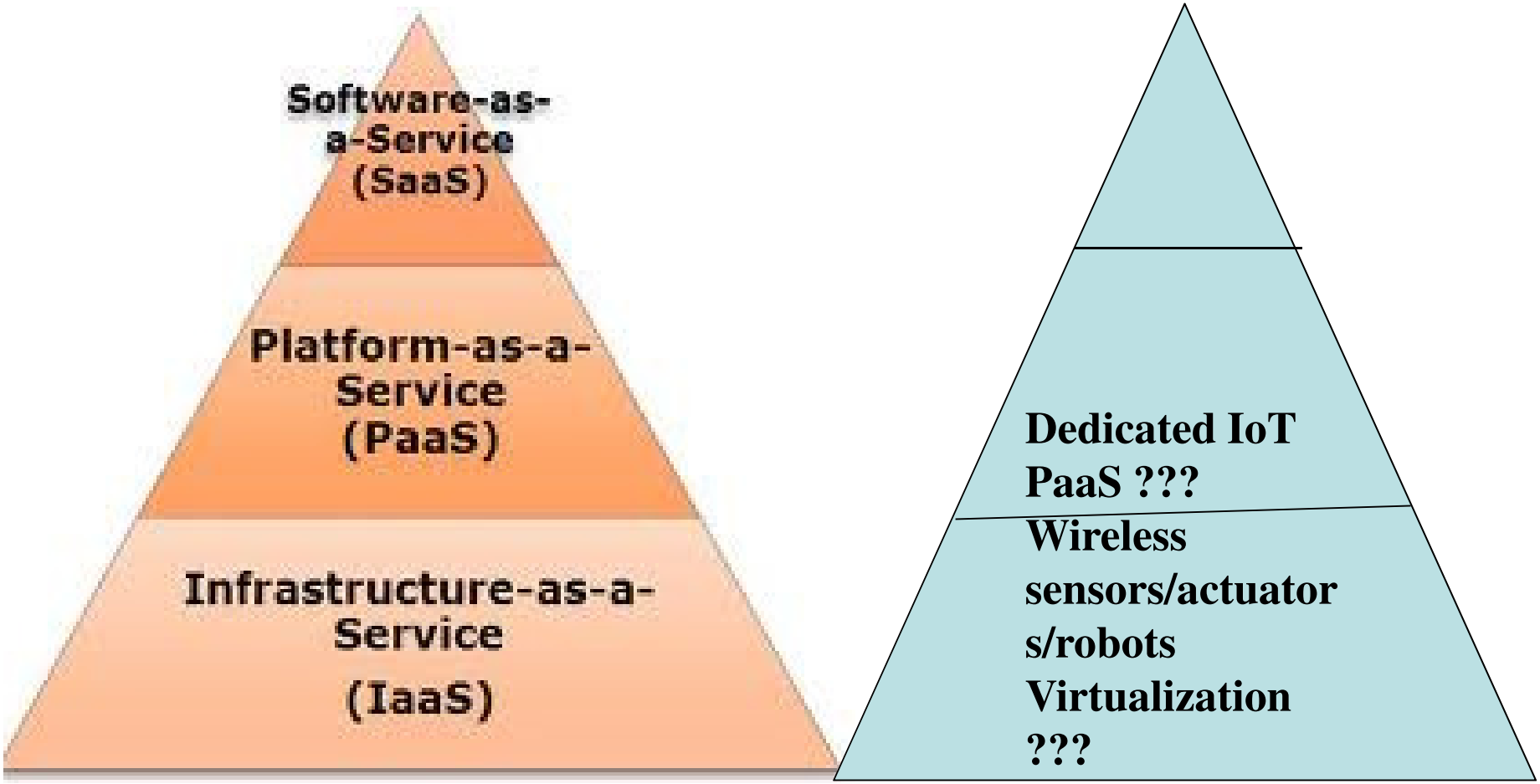
Managing Wearable Data Through Cloud

Some of the shortcomings:

- No true efficient usage of WSN resource through virtualization
- PaaS not geared towards WSN applications

- No general framework for re-using third party applications

Applying Cloud Fundamentals to IoT



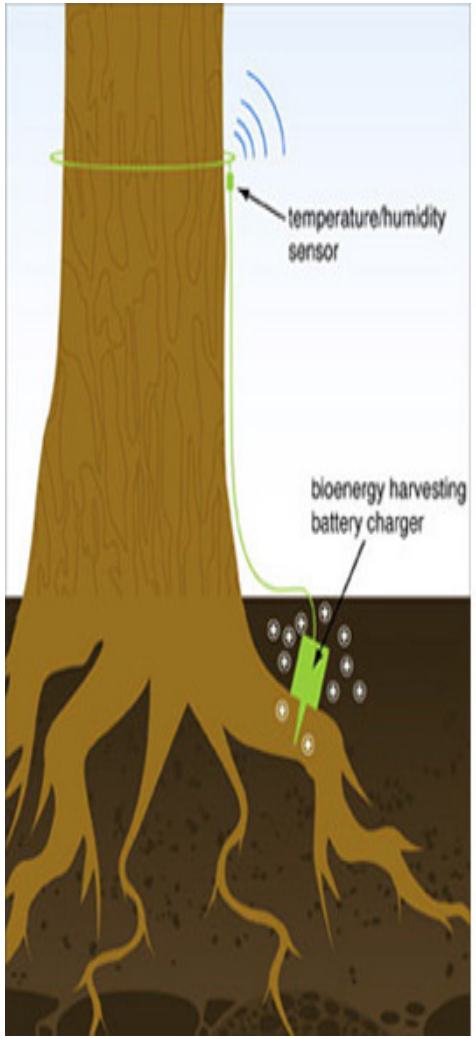
Our Own Work

- We are currently exploring the specific case of wireless sensors in a project with CISCO systems entitled

“ Towards Cost Efficient Applications and Services Provisioning in IP Wireless Sensor Networks”

- 2 part time post doctoral students
- 3 PhD students
- 2 Master students

Use Case: Smart Forests



Short Storyline

Actors

- Forest protection agency
- Wild fire management department
- Environment aware campers

In short

1. The forest protection agency deploys wireless sensors in a forest to collect environmental data to build forest environment monitoring applications that are offered as SaaS in a cloud (available anywhere/anytime and might re-use software modules (e.g. statistics packages that reside in different clouds)

2. Sometime later, the wild fire management department decides to use robot fire fighters to suppress wild fire. It decides to re-use the wireless sensor infrastructure already deployed by the forest protection agency. It build applications that are automatically notified when fire erupts, then dispatches automatically robot fire fighters. These applications will reside in the wild fire management department cloud,

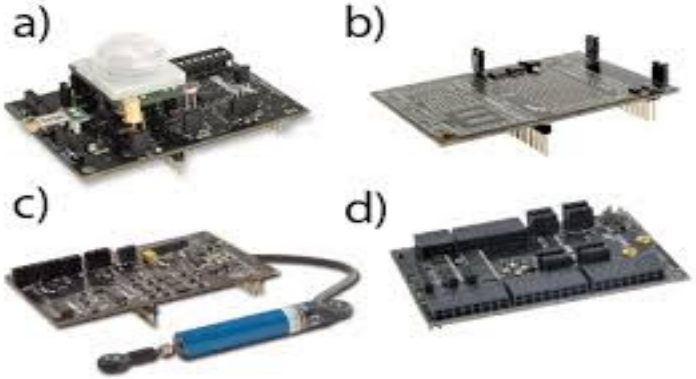
In short

3. The forest monitoring applications offered as SaaS by the forest protection agency became quite popular. However, some environment aware campers wanted more than what the applications could offer. The forest monitoring agency then decides to offer to these campers a PaaS with a very high level abstraction. With this PaaS they can either customize the applications offered as SaaS, or even develop and deploy their own applications that will re-use the very same wireless sensor environment.

The details

Infrastructure

- **Heterogeneous environmental wireless sensors**
  - Monitor CO2, CO, temperature humidity in the forest
    - Owned, deployed in the forest, and used by forest protection agency



- **Note: Thanks to node level / network level virtualization this infrastructure will be shared with other actors (i.e. wildfire management department, environmental aware campers)**

Infrastructure

- **In addition to the environment wireless sensors ..**
  - Heterogeneous fire fighter robots
    - Owned by wild fire management department (or eventually third parties) and located at different places
      - Some might fetch retardants
      - Others might do the actual fire extinguishment

- **Thanks to node level / network level virtualization this infrastructure will be used in an efficient way (i.e. Optimal coalition formation)**

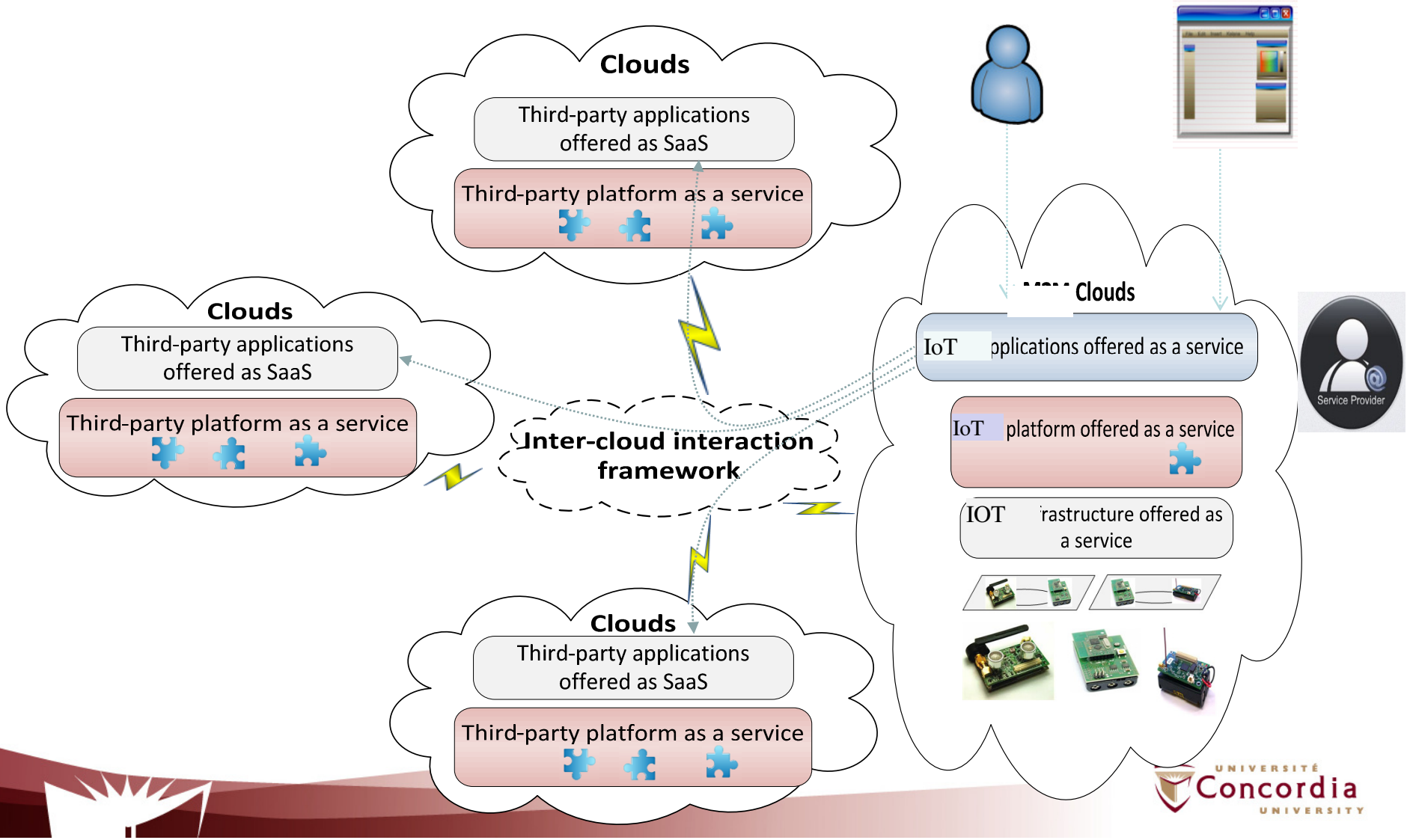
Examples of Applications that can run on the infrastructure

- **1. Environment changes monitoring applications**
  - Rolled out by forest protection agency with the deployment of environment wireless sensor
  - Developed by specialized software house
  - Show maps of the forest with potential drought / deforestation areas
- **2. Wild fire fighting application**
  - Rolled out by wildfire management agency
    - Re-use environment wireless sensor deployed by forest monitoring agency
    - Use robots
  - Developed by specialized software house
  - Is notified when fire erupts and use robots to extinguish the fire

Examples of Applications that can run on the infrastructure

- **3. Notifications when CO2 reaches given threshold**
  - Developed by non specialized programmers
  - Re-use the environmental wireless sensors deployed by forest monitoring agency
    - Measurement of impact on environment
      - Alarms when CO2 reaches thresholds during camp fires
    - Safety
      - Alarms when CO2 reaches thresholds in camping areas.

Towards A Comprehensive Service Delivery Platform for IoT



Towards a Comprehensive Architecture

Research issues / challenges

- Inter-cloud interactions in the specific case of IoT service delivery platform
  - Overlays might be used
- IoT applications offered as SaaS
- Platforms for the development and management of IoT applications in cloud settings
- Cloud infrastructure for IoTdevices / networks
  - IoT virtualization
    - Node level virtualization
    - Network level virtualization

Middleware for wireless sensor networks: Web Services based

- RESTFul Web Services based

RESTFul Web services middleware for M2M

RESTFul Web services in General

Introduction

- What about using the Web's basic technologies (e.g. HTTP) as a platform for distributed services?
  - This is what is REST about.

Introduction

- REST was first coined by Roy Fielding in his Ph.D. dissertation in 2000
- It is a network architectural style for distributed hypermedia systems.
- It is not an architecture, but a set of design criteria that can be used to asses an architecture

Introduction

- REST is a way to reunite the programmable web with the human web.

- It is simple
  - Uses existing web standards
  - The necessary infrastructure has already become pervasive
  - RESTFull web services are lightweight
  - HTTP traverse firewall

Introduction

- RESTFul web services are easy for clients to use
- Relies on HTTP and inherits its advantages, mainly
  - Statelessness
  - Addressability
  - Unified interface

REST Model

- Resources

- Clients make standard HTTP requests against resources, either an ag-gregate or a specic resource.

The format of the URL is:

- • endpoint/version/namespace/resource[?query parameters]
- • endpoint/version/namespace/resource/resource id

REST Operations

- GET queries for a list of a class of resources or the details of a specifc resource.
- POST creates a new resource instance and will provide either a job or a resource instance in the response body.
- PUT updates an existing resource with the specied parameters.
- DELETE removes or terminates or deactivates a resource.
- HEAD provides response headers, including a count of match-ing resource

REST Response Codes (Examples)

- 200 { request was successful and details about the response can be found in the body of the response.
- 201 { request POST was successful and an object was created in the system.
- 202 { requested operation has been accepted and the body contains further information.

Examples of tool kits

- RestLet
- Jersey

Examples of RESTful Web Services

- Examples of existing RESTful web services include:
  - Amazon's Simple Storage Service (S3) (*http://aws.amazon.com/s3*)
  - Services that expose the Atom Publishing Protocol (*http://www.ietf.org/html.charters/atompub-charter.html*) and its variants such as GData (*http://code.google.com/apis/gdata/*)
  - Most of Yahoo!'s web services (*http://developer.yahoo.com/*)
  - Twitter is a popular blogging site that uses RESTful Web services extensively.
  - Most other read-only web services that don't use SOAP
  - Static web sites
  - Many web applications, especially read-only ones like search engines

RESTFul Web services for M2M: Constrained Environments

Z. Shelby, Embedded Web Services, IEEE Wireless Communications, December 2010

IETF Constrained Application Protocol (CoAP)

- **Constrained Application Protocol (CoAP)**:
  - Realizes a minimal subset of REST along with resource discovery, subscription/notification, and the use of appropriate security measures

CoAP features

- **Compact Header**: binary header (4 bytes) + extensible options , and total header 10-20 bytes for typical requests
- **Methods and URIs**: like HTTP (GET, PUT, POST, DELETE)
- Content types: Can indicate content type of the payload in the header

CoAP features

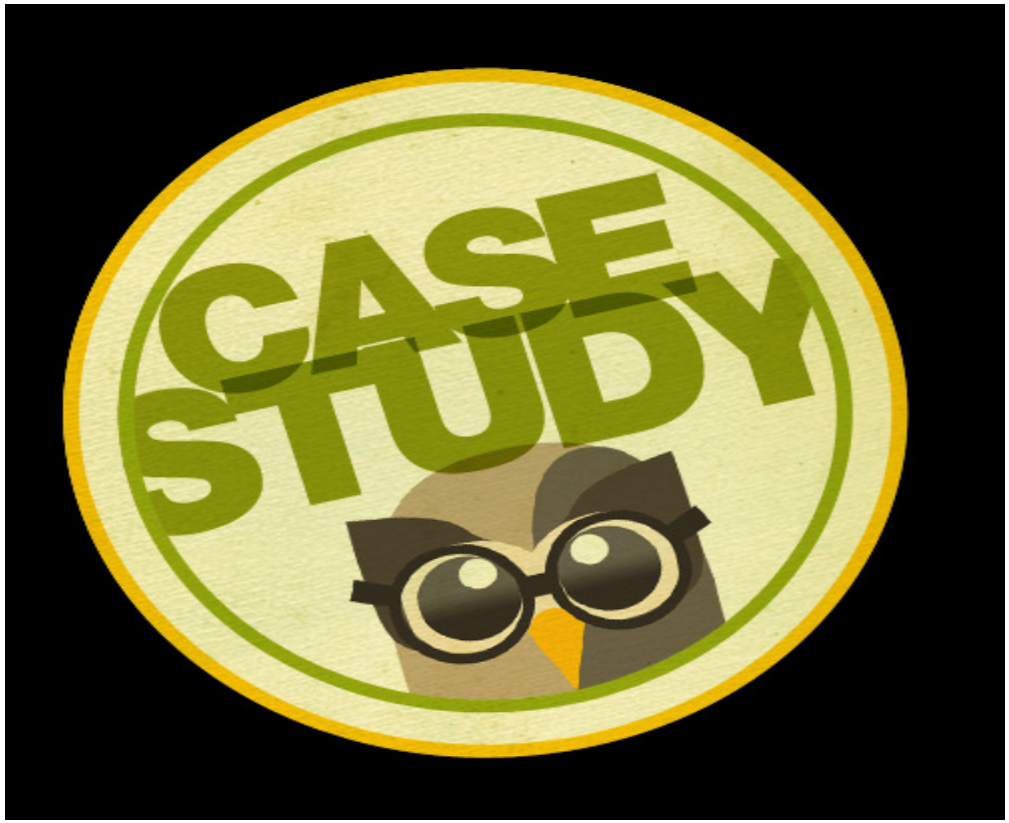
- **Transport binding**: UDP + simple stop-and-wait reliability mechanism. Optional security is supported using Datagram Transport Layer Security (DTLS)

CoAP features

- **Resource Discovery**: to discover the list of resources offered by a device, or for a device to advertise or post its resources to a directory service.

- **Subscription**: an asynchronous approach to support the push of information from servers to clients using subscriptions

Case Study : Integrating Wireless Sensor Networks with the Web



W. Colitti et al,
http://hinrg.cs.jhu.edu/joomla/images/stories/IPSN_2011_koliti.pdf

Design and Development of an End to End Architecture

- CoAP over 6LoWPAN
- Contiki based WSN
- Access of WSN data directly from a browser

Design and Development of an End to End Architecture

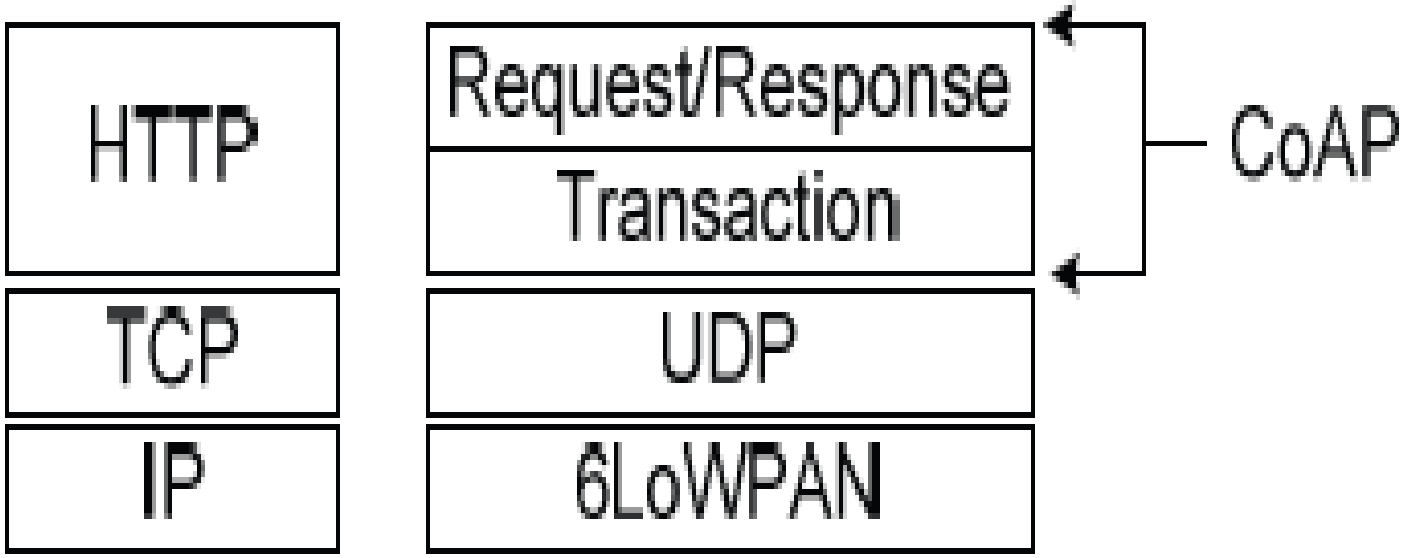


Figure 1. HTTP and CoAP protocol stacks

Design and Development of an End to End Architecture

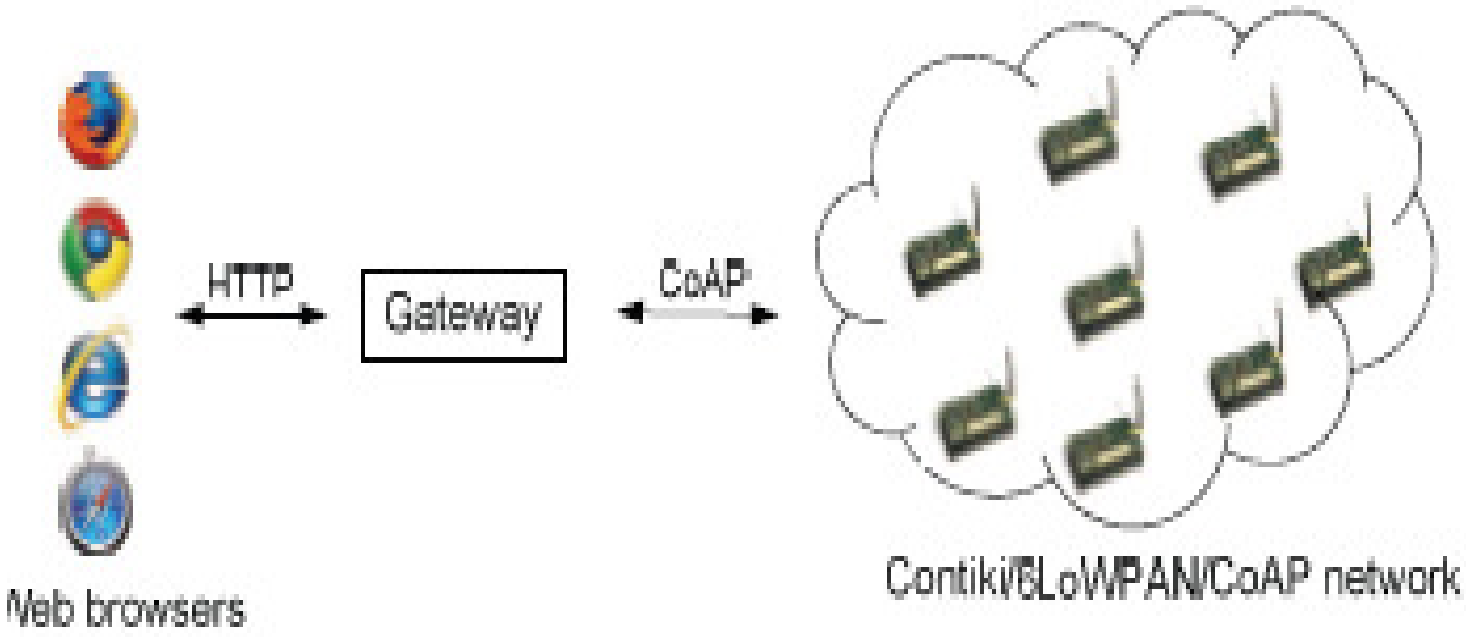
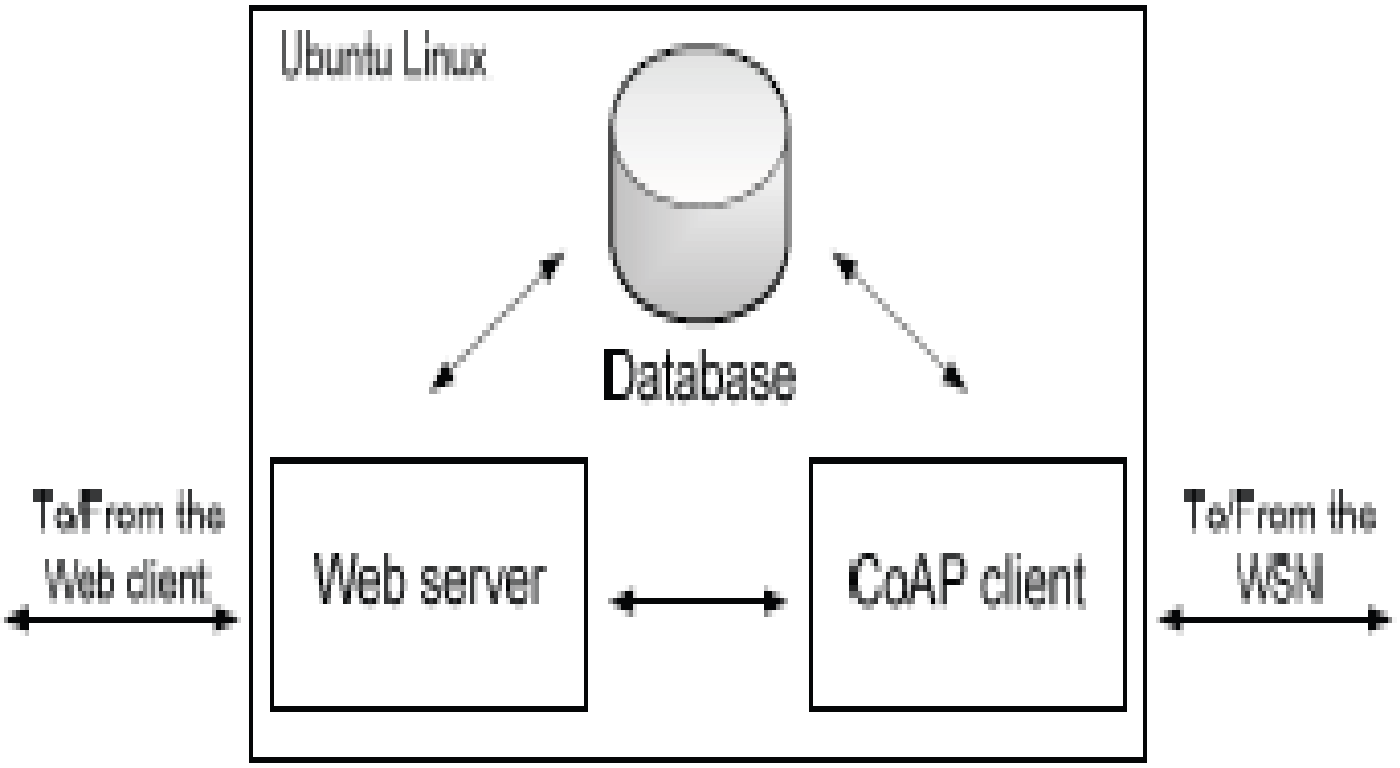


Figure 2. Integration between WSNs and the Web

Design and Development of an End to End Architecture




THE END

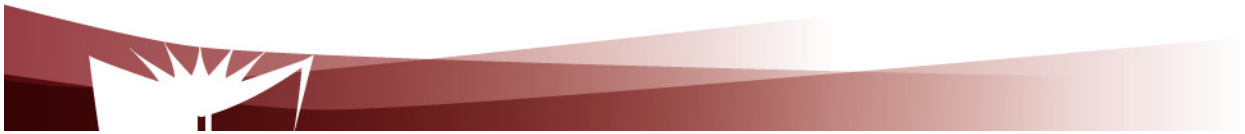

The End

The Proof Of Concept Project

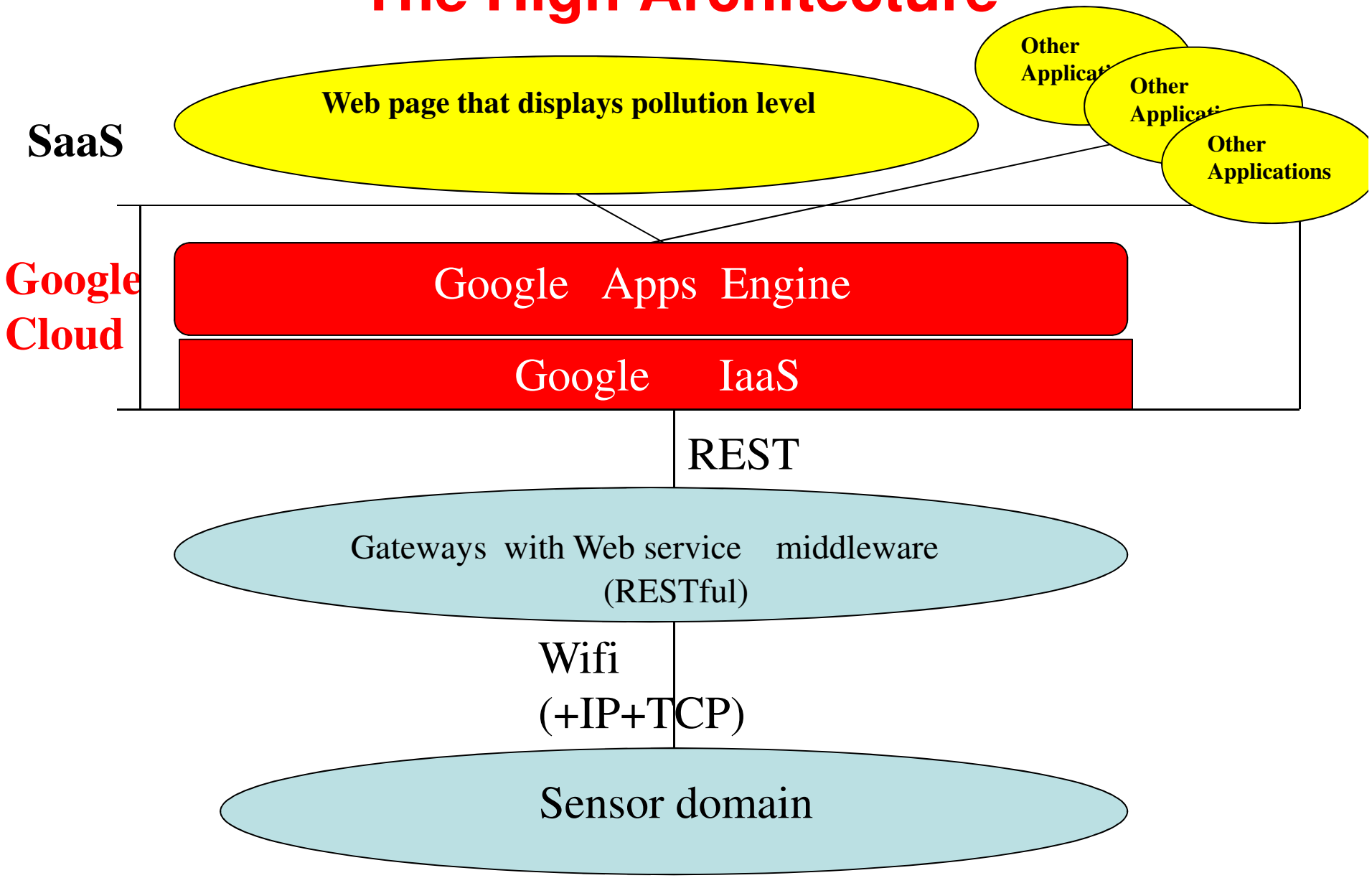
Roch Glitho, PhD

Associate Professor and Canada Research Chair

My URL - http://users.encs.concordia.ca/~glitho/



The High Architecture



REST Resource Model

Sensors http://benin-project.com/

All Sensors http://benin-project.com/sensors/

Sensors in specific location

http://benin-project.com/sensors/Location

Individual sensors:

http://benin-project.com/sensors/Location/Sensor-id

Individual sensors data:

http://benin-project.com/sensors/Location/Sensor-id/data

How simple could it be if we have a middleware?

Examples of requests from application to middleware:

HTTP GET:

http://benin-project.com/sensors/Dangbo/{*}/data

HTTP GET:

http://benin-project.com/sensors/Dangbo/sensor-id/data