SMART SANTANDER

SmartSantander Tutorial - Part 1

Overall purpose of the tutorial

- Gain a deeper understanding of experimental IoT research and how to conduct it on a real testbed
- Learn about the SmartSantander facility, its architecture, composition at different testbeds sites and use cases
- Get familiar with a typical experimentation life-cycle and how SmartSantander tools can help you during it
- Become productive on the different SmartSantander testbed sites and underlying HW platforms (hands-on)
- Understand in more detail the SW components of the testbed backend and how to setup your own testbed with it

Structure of tutorial

Session Title
Part 1: An overview of SmartSantander facility
Part 2: Experimentation with SmartSantander
Part 3: Hands-on Programming the different testbed sites - Santander/Belgrade - Guildford - Luebeck
Part 4: Setting up your own testbed

An overview of SmartSantander facility

PART 1

Learning goals - Part 1

- Understand why experimental evaluation is important
- Know the EU Future Internet Experimental Research (FIRE) landscape and how the SmartSantander facility relates to it
- Understand the main purpose of the SmartSantander facility and for what it is useful
- Know the architecture of SmartSantander and the purpose of its components
- Learn about the composition of different SmartSantander testbed sites and their characteristics
- Examine examples of use cases for experimentation

Motivation (1/2)

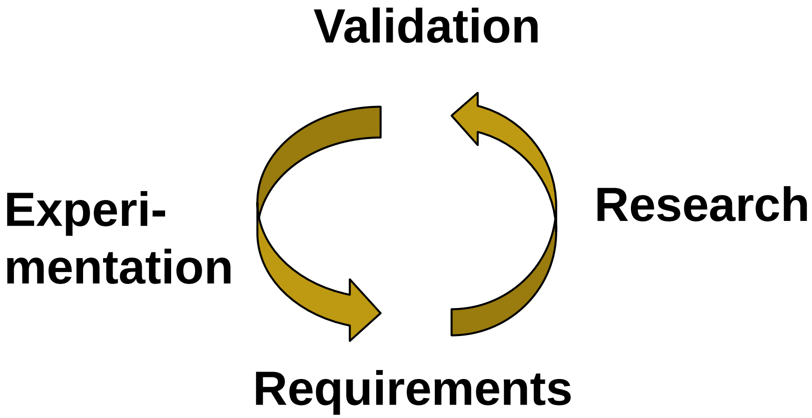
- Trends supporting need for real world experimentation
  - Demand for shorter cycles between research to market
  - High-tech solutions more often to be deployed “in the wild”
  - Increasing need for real end user involvement

Motivation (2/2)

- Top tier conferences in IoT, WSN and pervasive computing require experimental evaluation for credibility
  - ACM SenSys 2012 – 21 from 22 papers use experimental evaluation!
    - Indoor testbeds (4)
    - Custom HW testbed setups (7)
    - Mobile phones (5)
    - Mobiles phones with custom HW setup (4)
    - Traces from real world deployment (1)
    - Simulations using TOSSIM (1)
  - IEEE Percom 2013
    - Simulations with real world traces (9)
    - HW emulation with real world traces (1)
    - Experiments with infrastructure prototype (5)
    - Experiments with Android phone (6)
    - Experiments with both Android and infrastructure (1)
    - Small real world experiment, large scale simulation with traces (2)
    - PC based evaluation of prototype (2)

Benefits of experimental research

- Facilitate bridging of theory and reality
  - Difficult to model reality in all its detail
    - Idealistic/wrong assumptions
    - Implementation constraints
    - Limited evaluation conditions



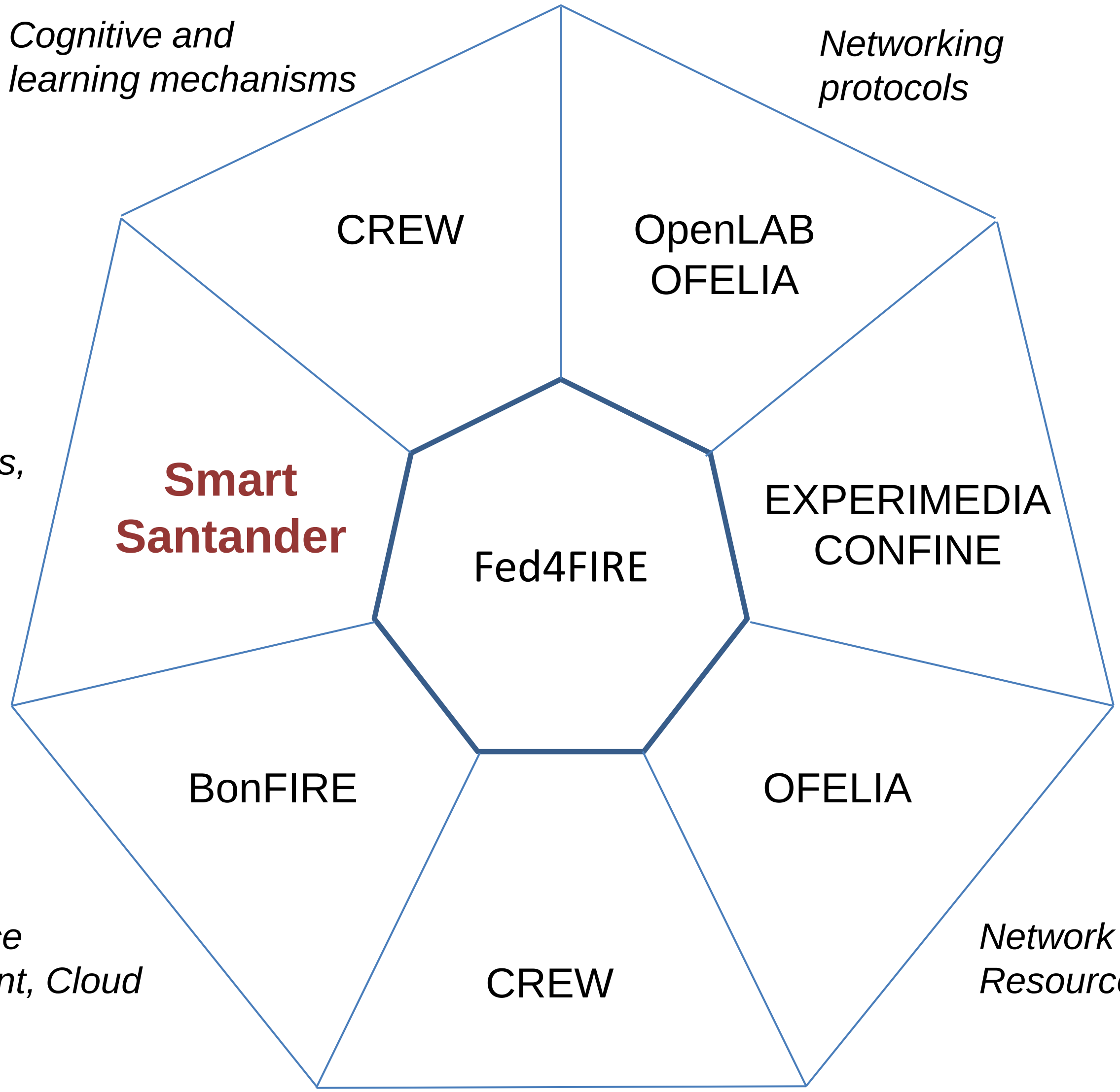
- Improved impact of results
  - Credibility
  - Easier adoption of outcomes
  - Faster path to market

Introduction to FIRE

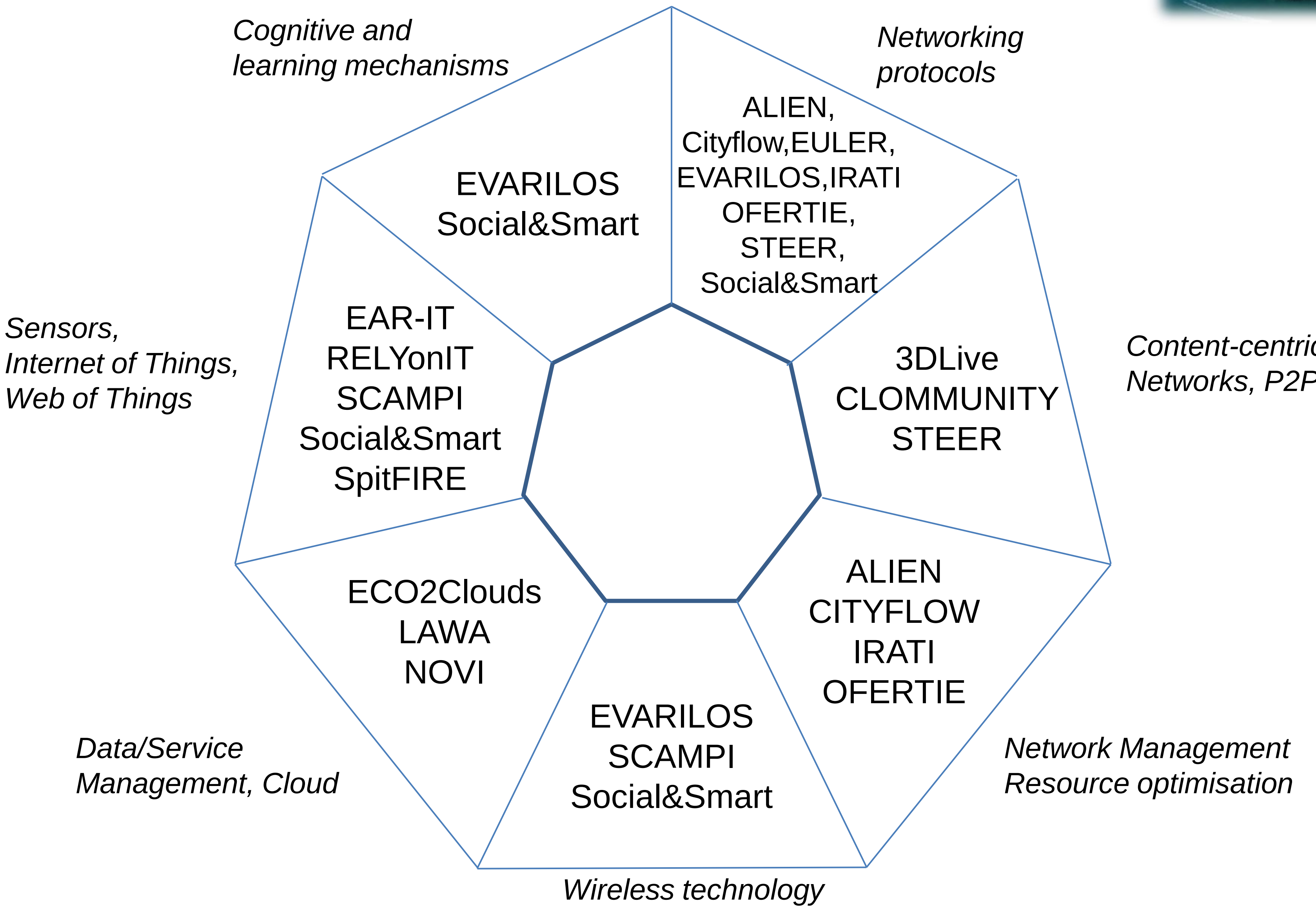
- Provide **large-scale experimental facilities** for Future Internet Research and Experimentation
  - New networking and service architectures and paradigms addressing all levels
  - Validate disruptive technologies for real and understand potential migration paths
  - Assess socio-economic implications

- Approach
  - Build experimental facilities need by community
  - Facilitate the sharing of research facilities
  - Support collaborative experimental research

FIRE facilities



FIRE research projects



IoT experimentation needs

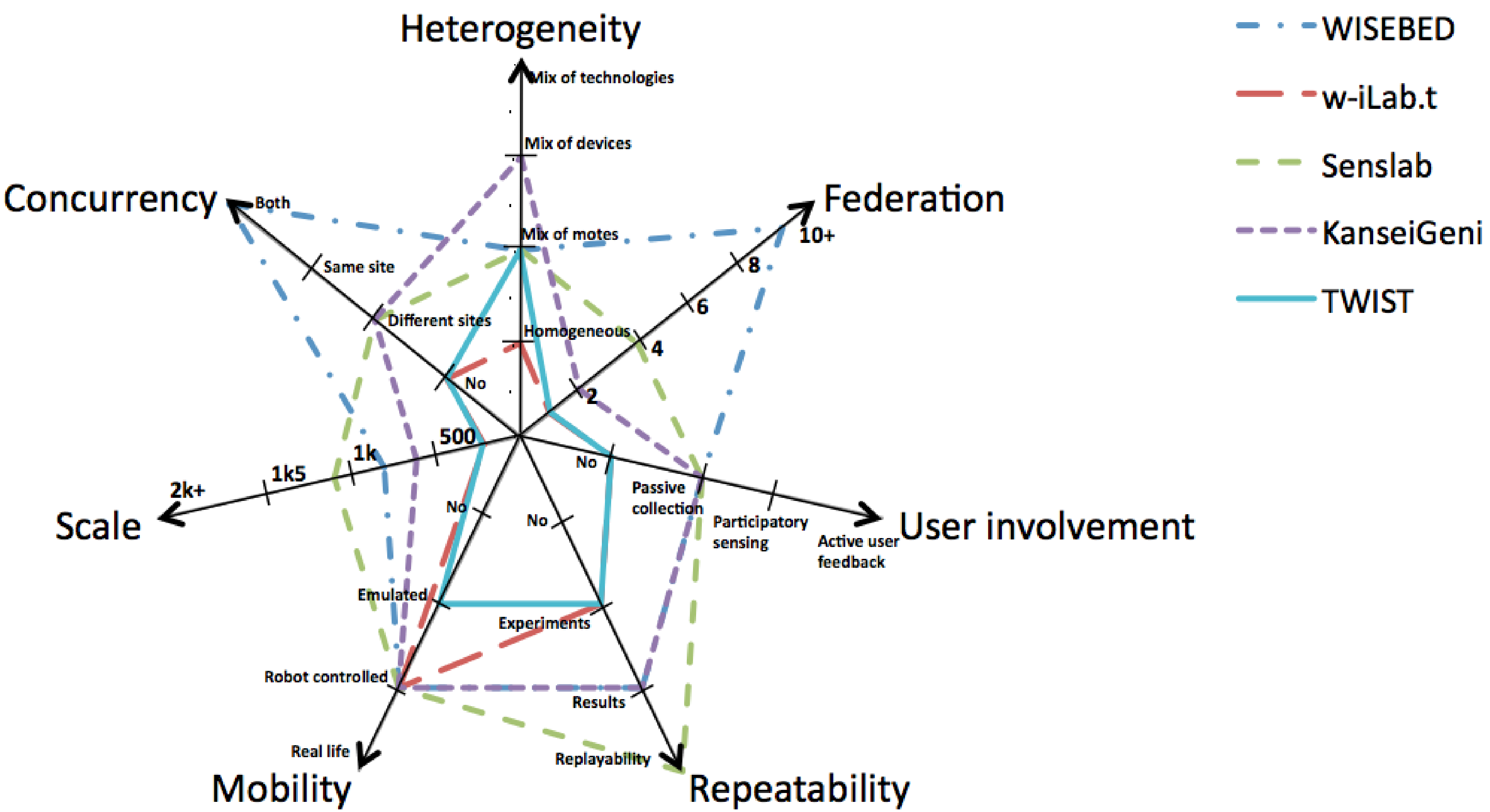
- IoT research and experimentation facility require
  - Realism of experimentation environment
  - Heterogeneity of IoT devices
  - Adequate scale
  - Mobility support from controlled to realistic
  - Concurrency
  - Repeatability and replayability
  - Real end-user involvement in the experimentation cycle
  - Federation with other Internet research facilities

Gluhak, A.; Krco, S.; Nati, M.; Pfisterer, D.; Mitton, N.; Razafindralambo, T., "**A survey on facilities for experimental internet of things research**," *Communications Magazine, IEEE* , vol.49, no.11, pp.58,67, November 2011

Examples of WSN/IoT testbeds

	Environment	Devices and scale
KANSEI-GENI	Indoor, lab	576 motes (96 XSM, 384 TelosB, 96 iMote2) attached to Stargate GW, 2 sites
SENSELAB	Indoor, lab	1024 WSN430 (521 with 802.15.4 MAC, 512 with free MAC layer), 4 sites
w-ilab.t	Indoor, lab	200 TMoteSky motes
TWIST	Indoor, lab	204 motes (102 TelosB, 102 eyesIFX)
WISEBED	Indoor, lab and outdoor	750 motes (200 iSense, 143 TelosB, 108 G-Node, 100, MSB-A2, 44 SunSPOT, 60 pacemate, 24 Tnode), 9 different sites

Comparison of key facilities



SmartSantander overview

- City-scale EU facility for the research and experimentation of Internet of Things (IoT) technologies and services/applications in a Smart-City context

Smart Santander Highlights

Targeting:

- Researchers
- Service providers
- End users

Duration: 36 months

Consortium:

- 15 organizations
- 8 EU countries + AU

Budget / Funding

- 8.67 M€ /6.00 M€

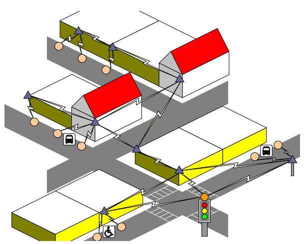
Why IoT testbed in a city

- Cities are excellent catalyst for IoT technology innovation
  - Dense social eco-systems heavily relying on technology
  - Necessary critical mass of experimental businesses, local governments and citizens as end-users
  - Initial impact of the IoT will be most visible to European citizens

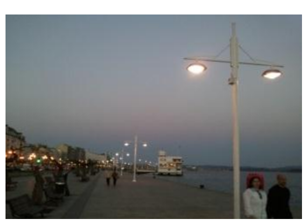
- Crosscutting applications, covering multiple dimensions of smartness
  - Smart-Economy (competitiveness), Smart-People (social and human capital), Smart-Governance (participation), Smart-Mobility (transport and ICT), Smart-Environments (natural resources), Smart-Living (quality of life)

SmartSantander facility

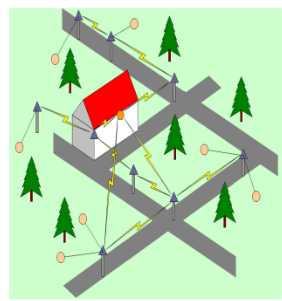
Smart traffic & parking



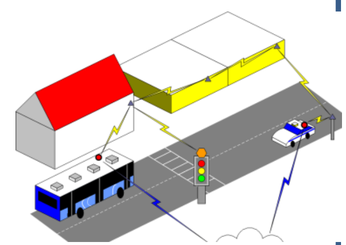
Smart lighting & environment



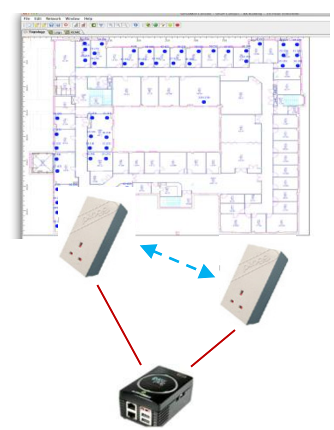
Smart parks



Mobile env. monitoring



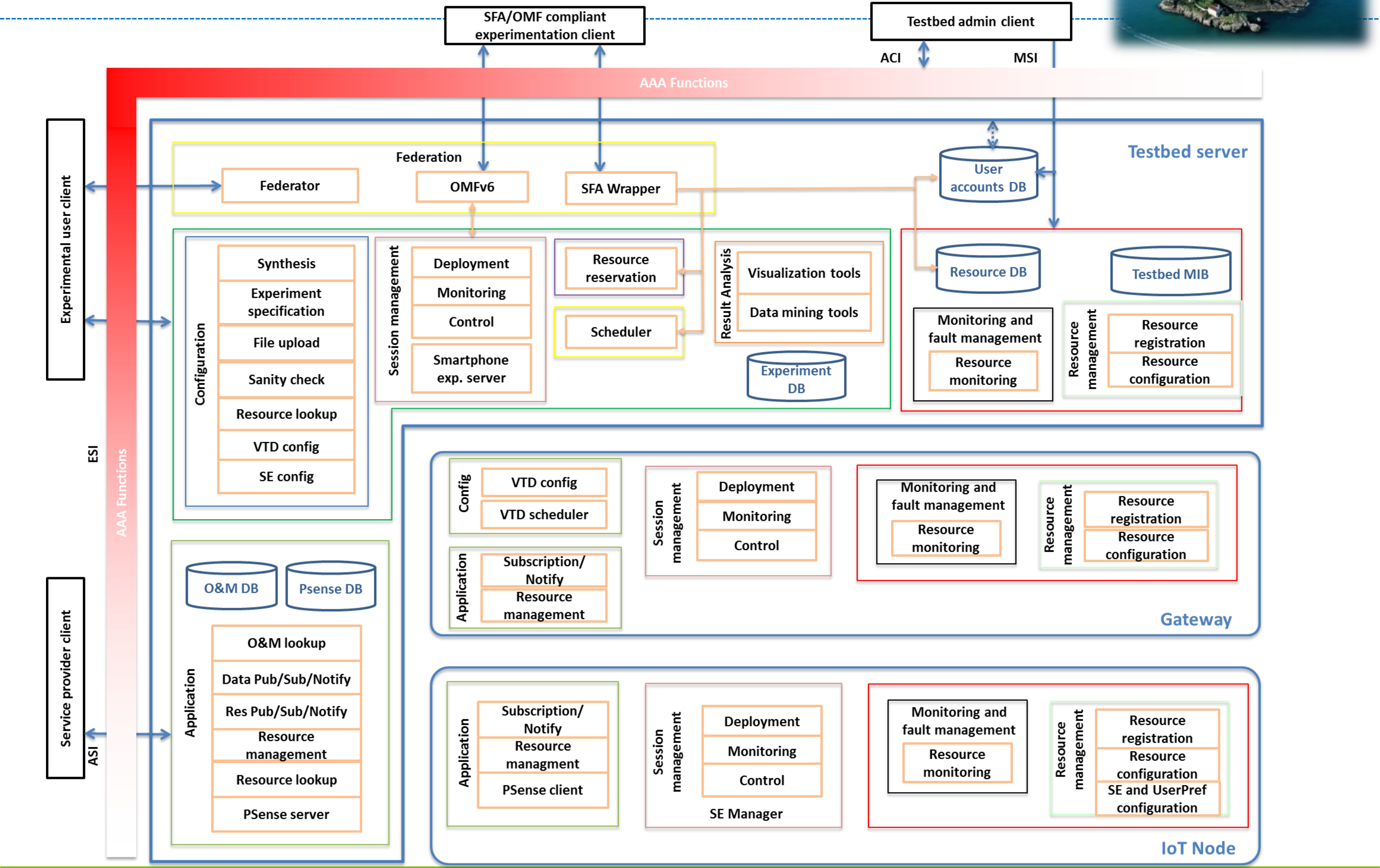
Smart buildings & energy



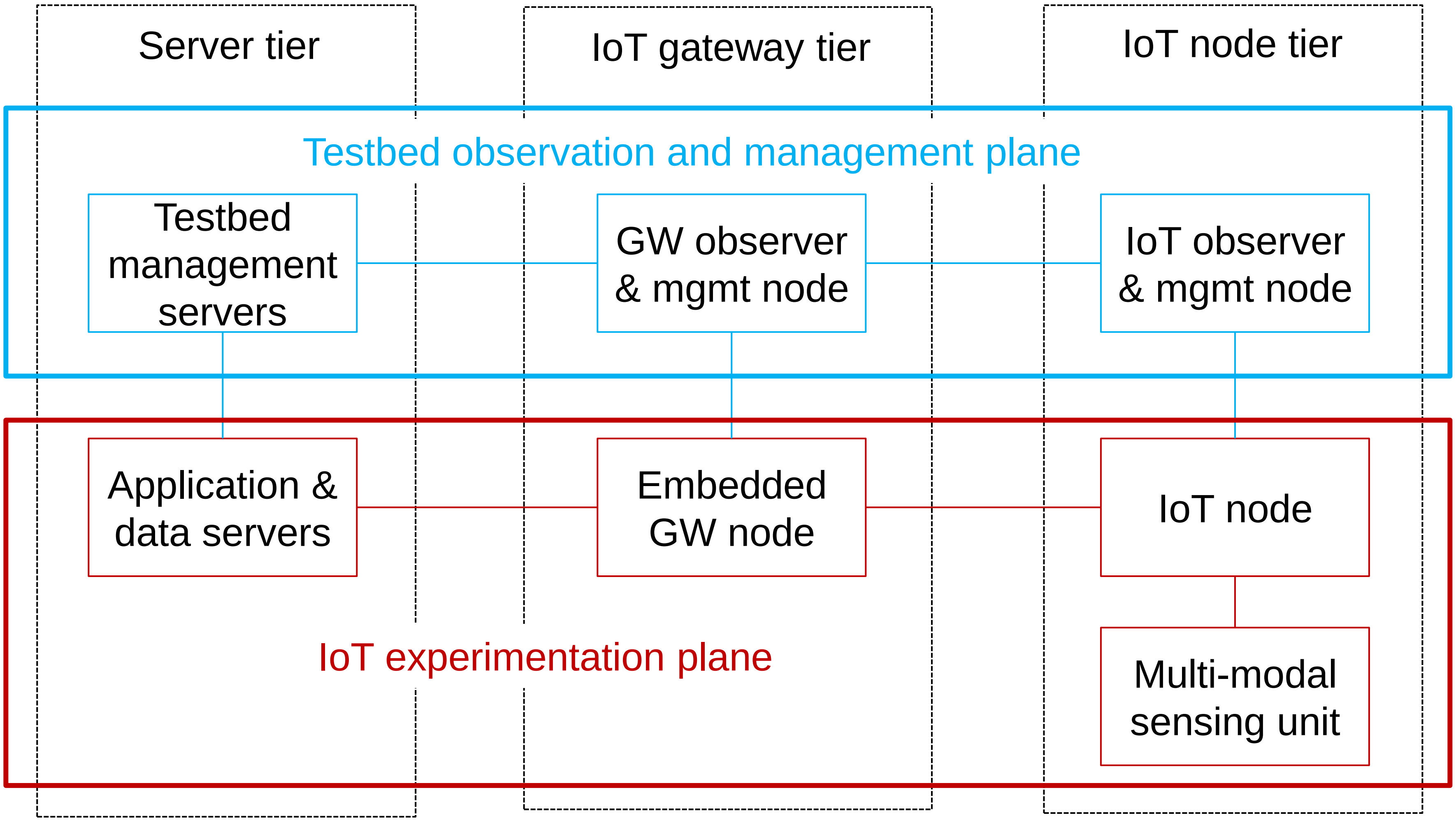
Key features

- Heterogeneity of HW and environments
- Deep societal penetration
- Mobility diversity
- Scale
- Live-services for public
- Federability with other facilities
- Diversity of supported experimentation
  - IoT protocol research
  - IoT services
  - User centric experimentation

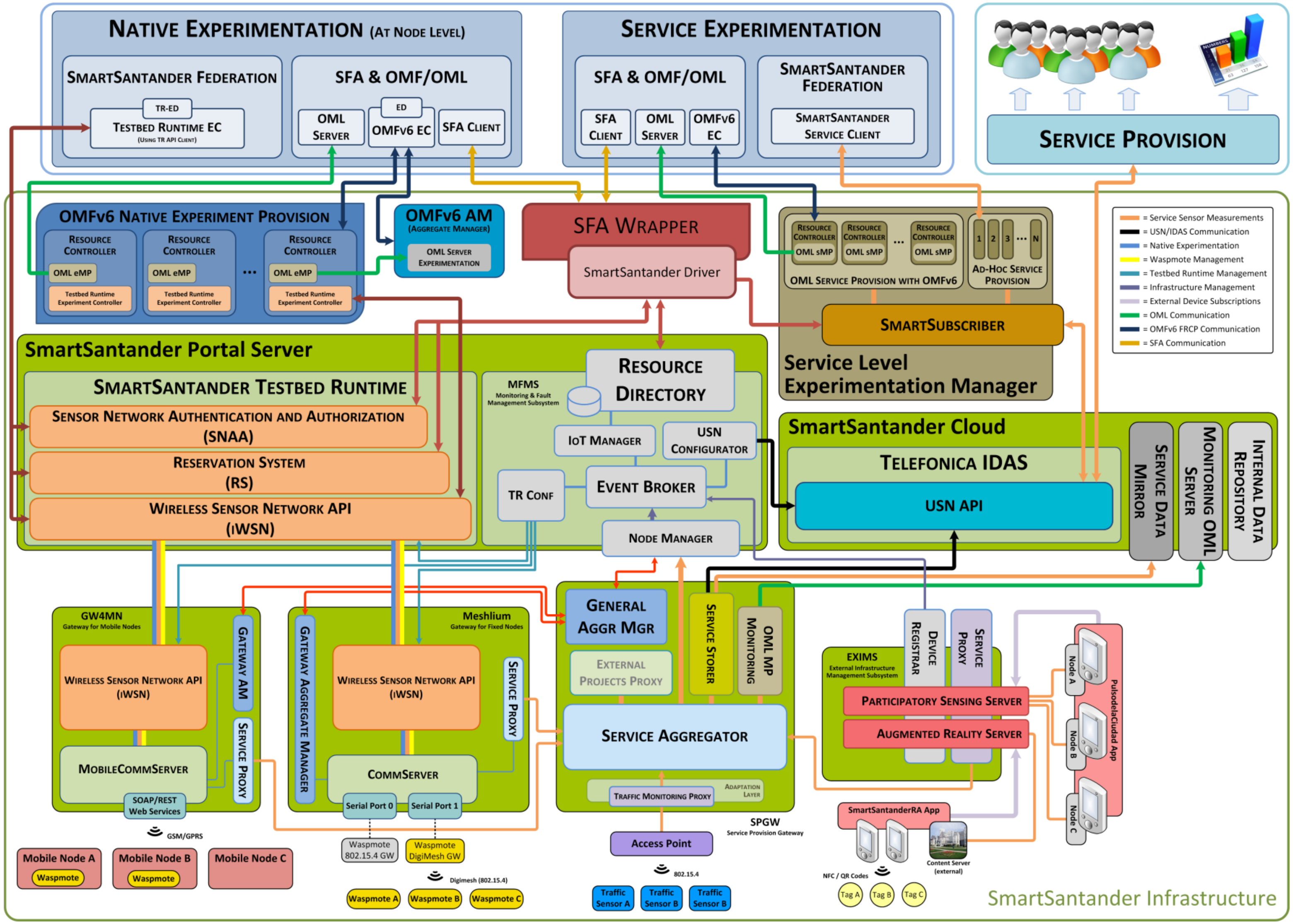
SmartSantander architecture



HW architecture



Implementation architecture

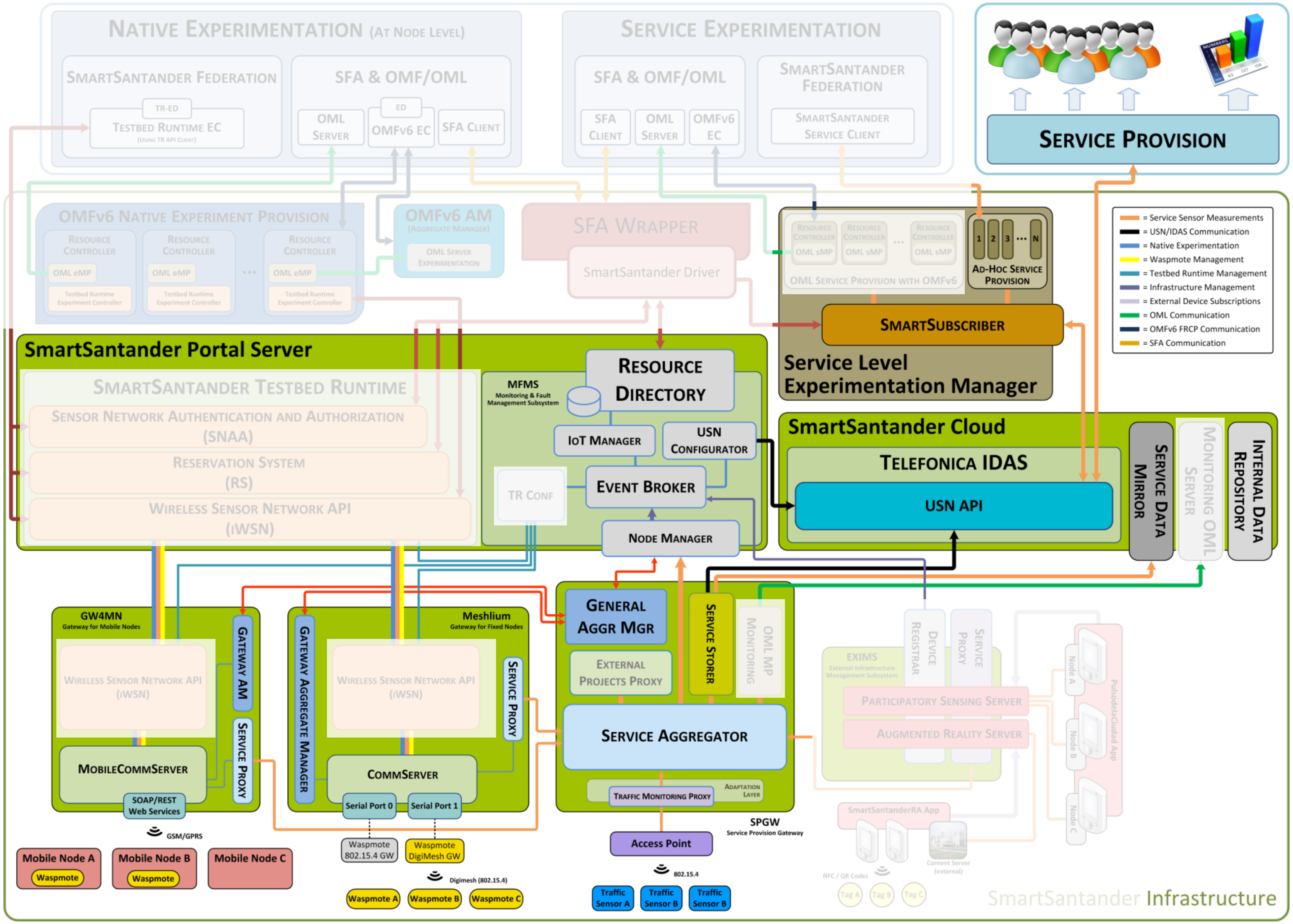


SmartSantander Architecture

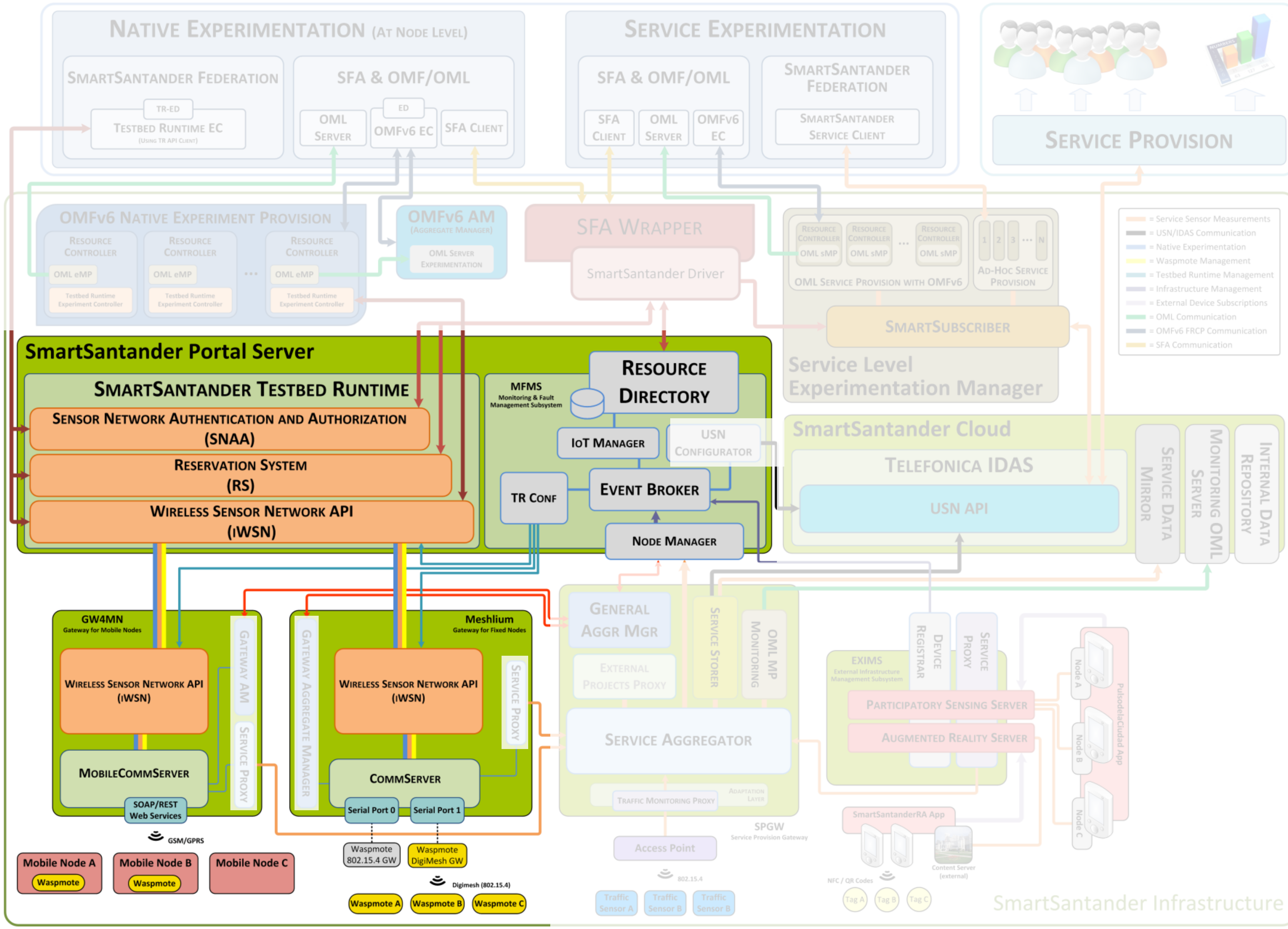
The SmartSantander architecture has 3 different experimentation planes:

1. Service-Level Experimentation (data)
2. Node-Level Experimentation (system)
3. Social Experimentation (people)

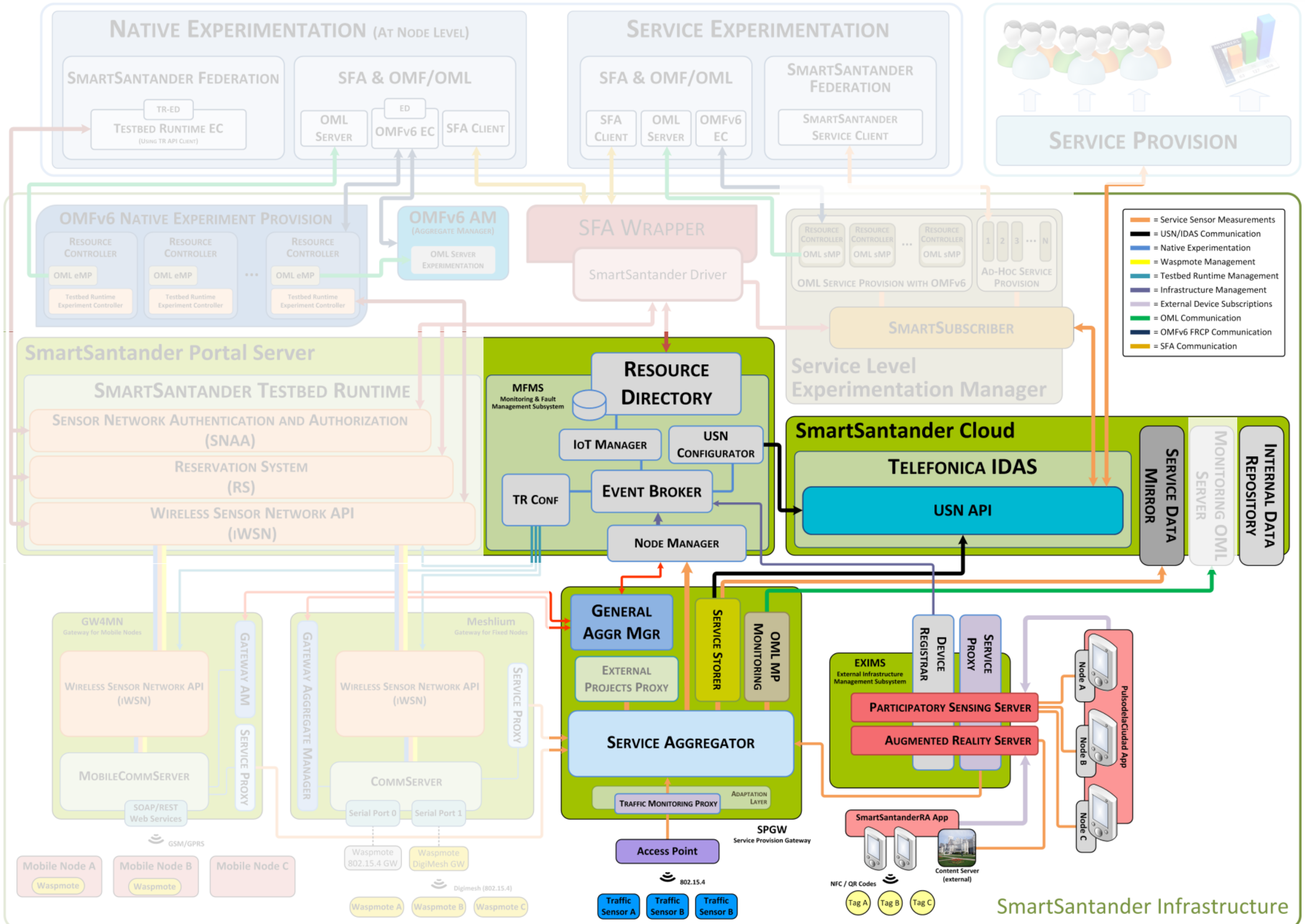
1. Service-Level Experimentation



2. Node-Level Experimentation



3. Social Experimentation



The SmartSantander Experimental Facility

SANTANDER DEPLOYMENT

Outline

- Phased roll-out
- City-scale deployment
- Conclusions

How is SmartSantander becoming a reality

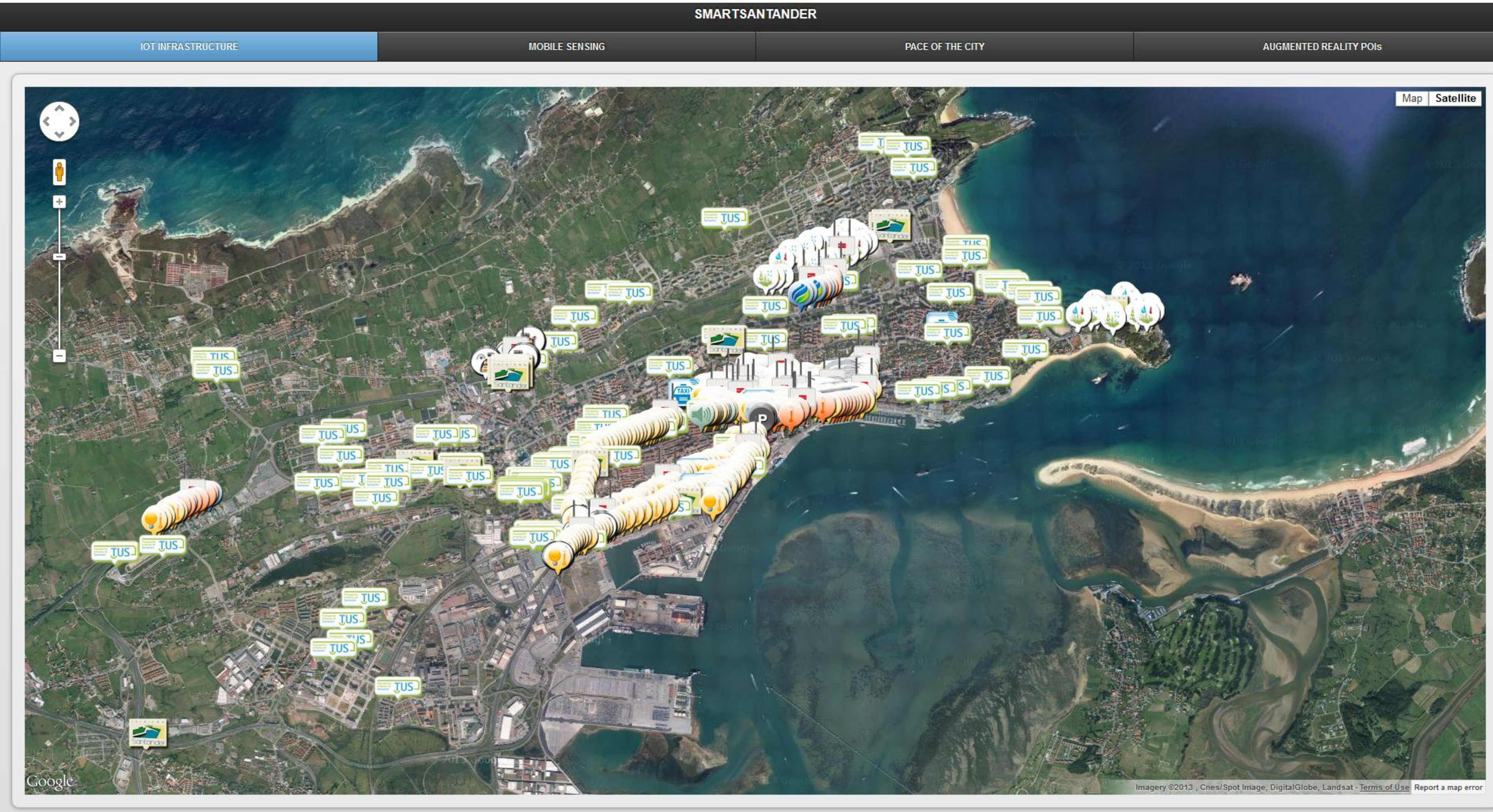
- Phased roll-out and deployment:

	Phase 1	Phase 2	Phase 3
Time	November 2011	November 2012	August 2013
Scale	2.000 IoT devices	5.000 IoT devices	20.000 IoT devices
Resources	Mainly WSN nodes and GWs	More heterogeneity WSNs, RFID, GW	Federated with other FIRE facilities
Facility services	Basic experimentation support	Advanced tools for experimentation	Advanced cross-testbed tools
Application domains	Transport, metering, environment	Traffic, environment, Participatory sensing, Augmented Reality	Parking, environment

Basis for 1st call experiments **Basis for 2nd call experiments**

City-scale deployment

- Better to see it on-line



City-scale deployment

- Phase 0 and 1 deployment
  - 2 use cases:
    - (2.1) Environmental monitoring
    - (2.2) Outdoor parking management

Node Type		Amount	Sensors	Radio I/F
Gateway		23	N/A	IEEE 802.15.4, IEEE 802.11, Digimesh, GPRS/UMTS
Repeater	Temperature	74	Temperature, Acceleration	IEEE 802.15.4, Digimesh
	Light	553	Light, Temperature, Acceleration	
	Noise	58	Noise, Acceleration	
	Gases	13	Temperature, CO, Acceleration	
Parking Sensor		373	Occupancy	Digimesh
	Total:	**23 GW 1,071 Nodes**	**2,322 sensors**	

City-scale deployment

- Phase 2 deployment
  - Extended with 6 new use cases:
    - (2.1) Traffic monitoring
    - (2.2) Mobile environmental monitoring
    - (2.3) Precision irrigation
    - (2.4) Guidance to parking lots
    - (2.5) Participatory sensing
    - (2.6) Augmented reality

City-scale deployment

- Phase 2 deployment

Node Type		Amount	Sensors	Radio I/F
Gateway	Irrigation	3	N/A	IEEE 802.15.4, IEEE 802.11, Digimesh, GPRS/UMTS
	Traffic	2		IEEE 802.15.4, GPRS/UMTS
Repeater	Traffic	9	N/A	IEEE 802.15.4
	Weather	3	Temperature, Relative Humidity, Soil Moisture, Solar Radiation, Rainfall, Windspeed, Atmospheric Pressure, Acceleration	IEEE 802.15.4, Digimesh
	Irrigation	23	Temperature, Relative Humidity, Soil Moisture, Soil Temperature, Acceleration	IEEE 802.15.4, Digimesh
	Water Flow	2	Water Flow, Acceleration	IEEE 802.15.4, Digimesh
	Agriculture	19	Temperature, Relative Humidity, Acceleration	IEEE 802.15.4, Digimesh
Mobile node	Bus (w. CAN-BUS)	2	CO, Particles, NO_2, Ozone, Temperature, Relative Humidity, Speed, Course, Odometer, Location, CAN	IEEE 802.15.4, GPRS
	Bus	68	CO, Particles, NO_2, Ozone, Temperature, Relative Humidity, Speed, Course, Odometer, Location	IEEE 802.15.4, GPRS
	Car	80	CO, Particles, NO_2, Ozone, Temperature, Relative Humidity, Speed, Course, Odometer, Location	GPRS
Traffic Sensor		59	Road Occupancy, Vehicle Count, Vehicle Speed	IEEE 802.15.4
Augmented Reality Tag		2,500	Presence (+ metadata)	NFC
Participatory Sensing Smartphone		6,500	Multiple	IEEE 802.11, GPRS/UMTS
Augmented Reality Smartphone		~14,000	Presence (+ metadata)	IEEE 802.11, GPRS/UMTS
Total:		**5 GW 115 Fixed Nodes 150 Mobile Nodes 2,500 Tags 10,000+ Smartphones**	**377 fixed sensors 1,500+ mobile sensors 20,000+ smartphone sensors**	

City-scale deployment

- Phase 3 deployment
  - Reinforce existing use cases
    - (3.1) Mobile environmental monitoring
    - (3.2) Outdoor parking management

Node Type		Amount	Sensors	Radio I/F
Gateway		3	N/A	Proprietary, GPRS/UMTS
Repeater		37	N/A	Proprietary
Mobile node	Bus (w. CAN-BUS)	10	CO, Particles, NO_2, Ozone, Temperature, Relative Humidity, Speed, Course, Odometer, Location, CAN	IEEE 802.11, GPRS
	Bus	15	CO, Particles, NO_2, Ozone, Temperature, Relative Humidity, Speed, Course, Odometer, Location	IEEE 802.15.4, GPRS, IEEE 802.11
Parking Sensor		330	Occupancy	Proprietary
Parking Tag		30	Authorization	Proprietary
Total:		**3 GW 330 Fixed Nodes 25 Mobile Nodes 30 Tags**	**330 fixed sensors 250+ mobile sensors**	

City-scale deployment

- Deployment summary
  - Heterogeneous mix of sensors and devices
  - Efficient use of budget → Maximizing experimentation possibilities
  - Mobile devices
  - End-users involvement
    - Not only passive influence of technology but also proactive involvement through participatory sensing

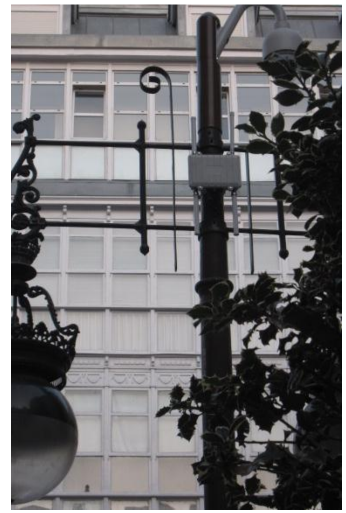
Phase 1:	**23 GW, 1,071 Fixed Nodes**	**2,322 fixed sensors**
Phase 2:	**5 GW, 115 Fixed Nodes, 150 Mobile Nodes, 2,500 Tags, 10,000+ Smartphones**	**377 fixed sensors, 1,500+ mobile sensors 20,000+ smartphone sensors**
Phase 3:	**3 GW, 330 Fixed Nodes, 25 Mobile Nodes, 30 Tags**	**330 fixed sensors, 250+ mobile sensors**
Total:	**31 GWs 1,516 Fixed Nodes 175 Mobile Nodes 2,500 Tags**	**3,029 fixed sensors 1,750+ mobile sensors 20,000+ smartphone sensors**

City-scale deployment

- Massive generation of information
  - 139,370 environmental monitoring observations per day
  - 8,365 irrigation monitoring observations per day
  - 82,726 mobile environmenal monitoring observations per day
  - 13,489 parking occupancy observations per day
  - 54,720 traffic management observations per day
  - 6,352 participatory sensing observations per day
- 450 Mbytes in one year

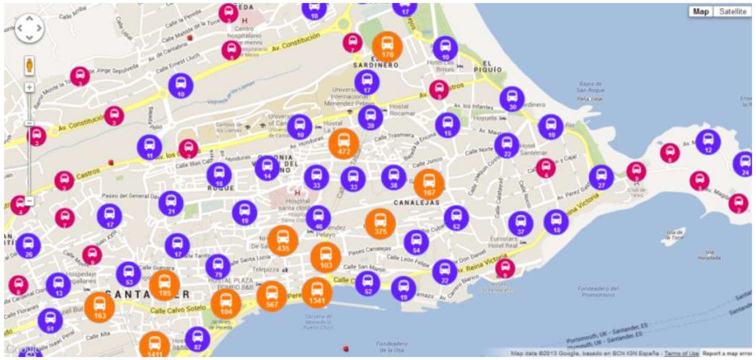
City-scale deployment

From the lab to the hostile outdoor scenario!!



Beyond City-scale deployment

From the lab to the hostile outdoor scenario!!

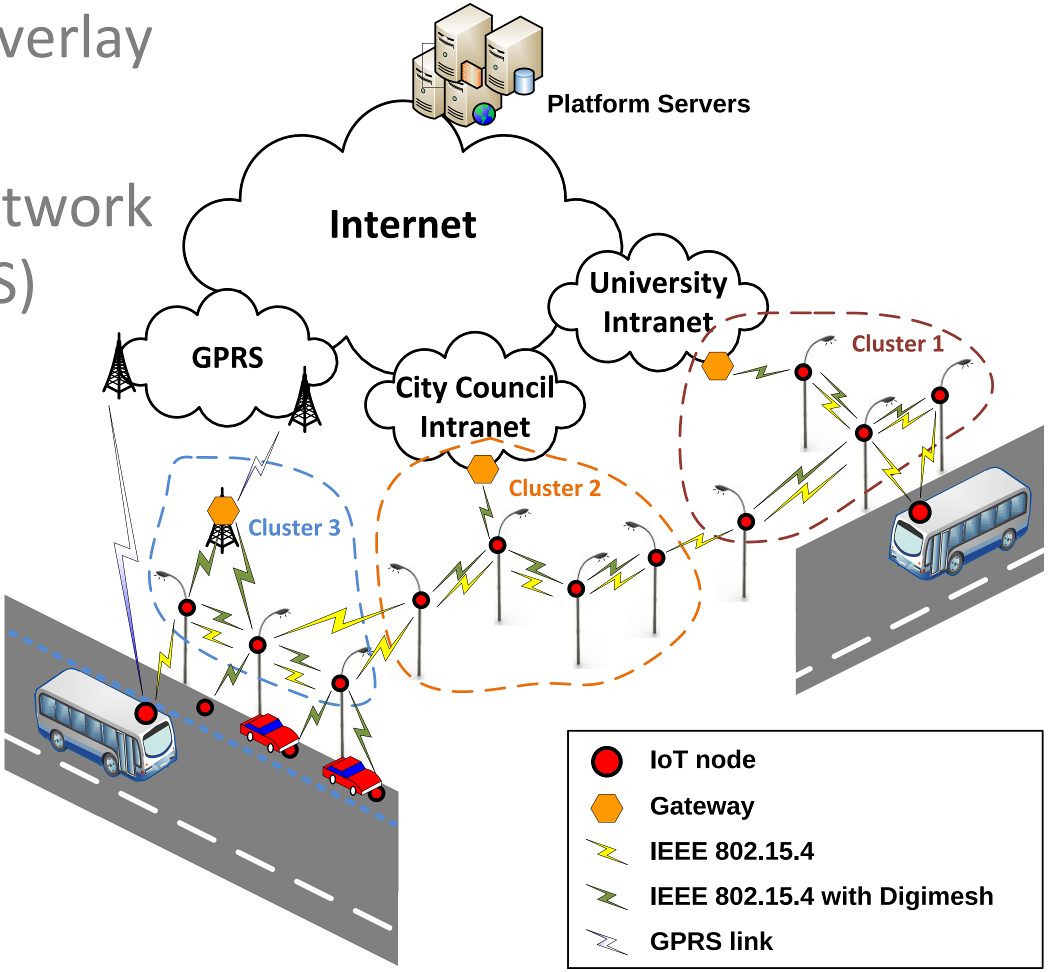


Experimentation support and tools

- Smart City for experimentation
  - Wide possibilities for experimentation
    - WSN management
    - services and applications
    - data and knowledge engineering
    - protocols experimentation
- Smart City for services
  - Several services already provided
  - Platform for Sensing as a Service paradigm
  - Open and flexible

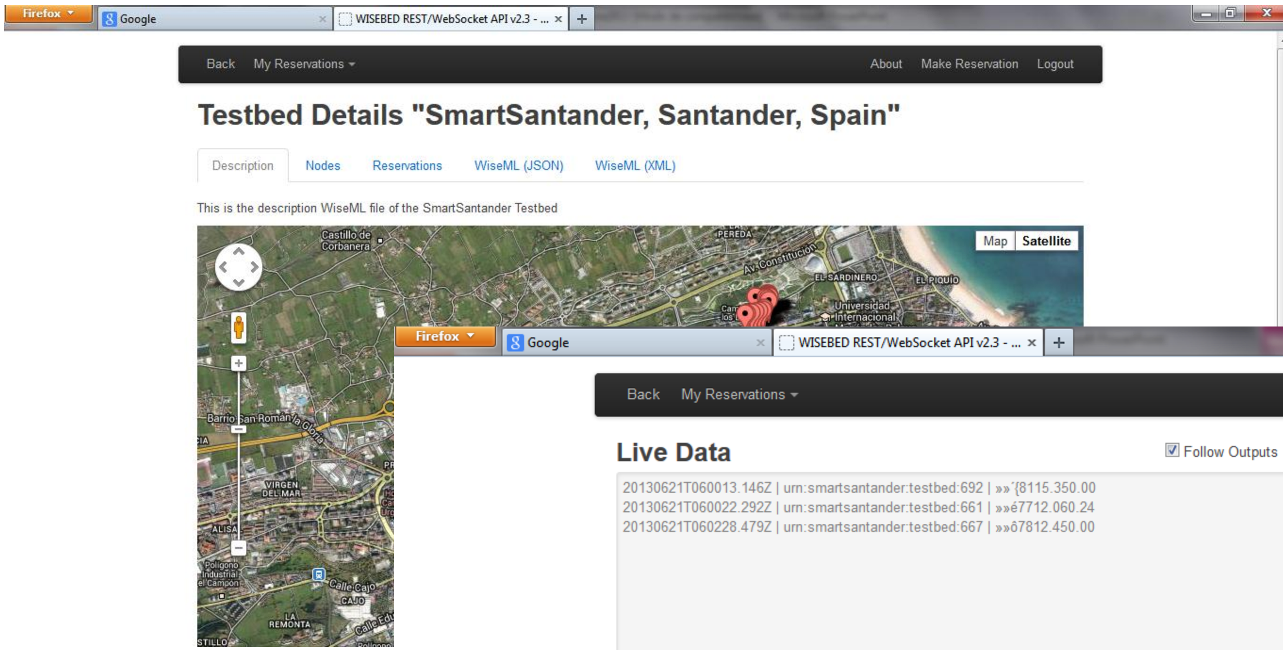
Experimentation support and tools

- WSN network architecture
  - IEEE 802.15.4 overlay network
  - Backhaul wireless network (Digimesh, GPRS, UMTS)

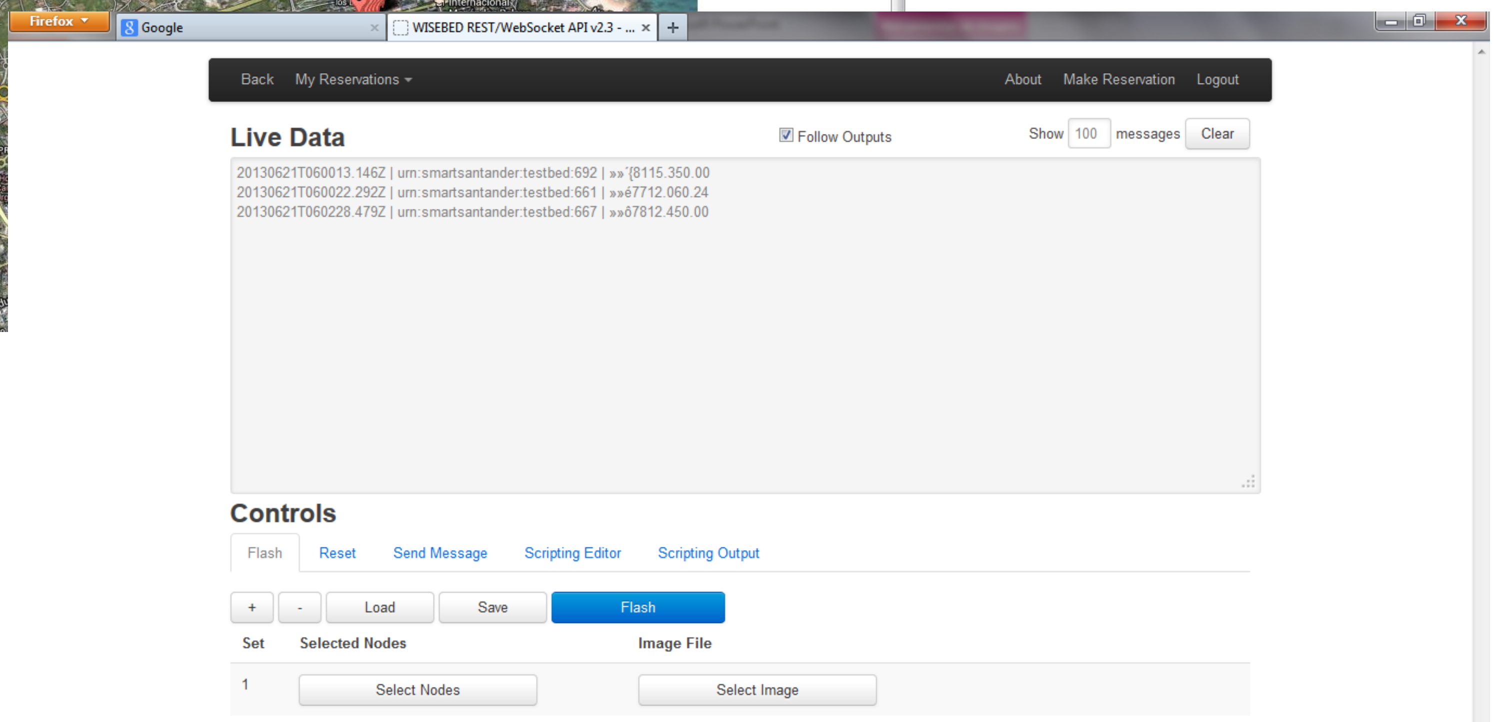


Experimentation support and tools

Specification phase



Execution phase



Experimentation support and tools

- Service level experimentation
  - REST/JSON Interface
    - Access to information records
      - By position
      - By node
      - By type of sensor
    - Access to historic values
  - IDAS framework
    - PUB/SUB access to information

Service Provision

Conclusions

- Dual purpose facility, supporting both experimentation and service provision
- Experimentation over IoT infrastructure with different focus
- The development of citizen-centric services aims at the intensively involvement of users in the smart city
- Application usage is itself meant to be subject of experimentation

The SmartSantander Experimental Facility

GUILDFORD DEPLOYMENT

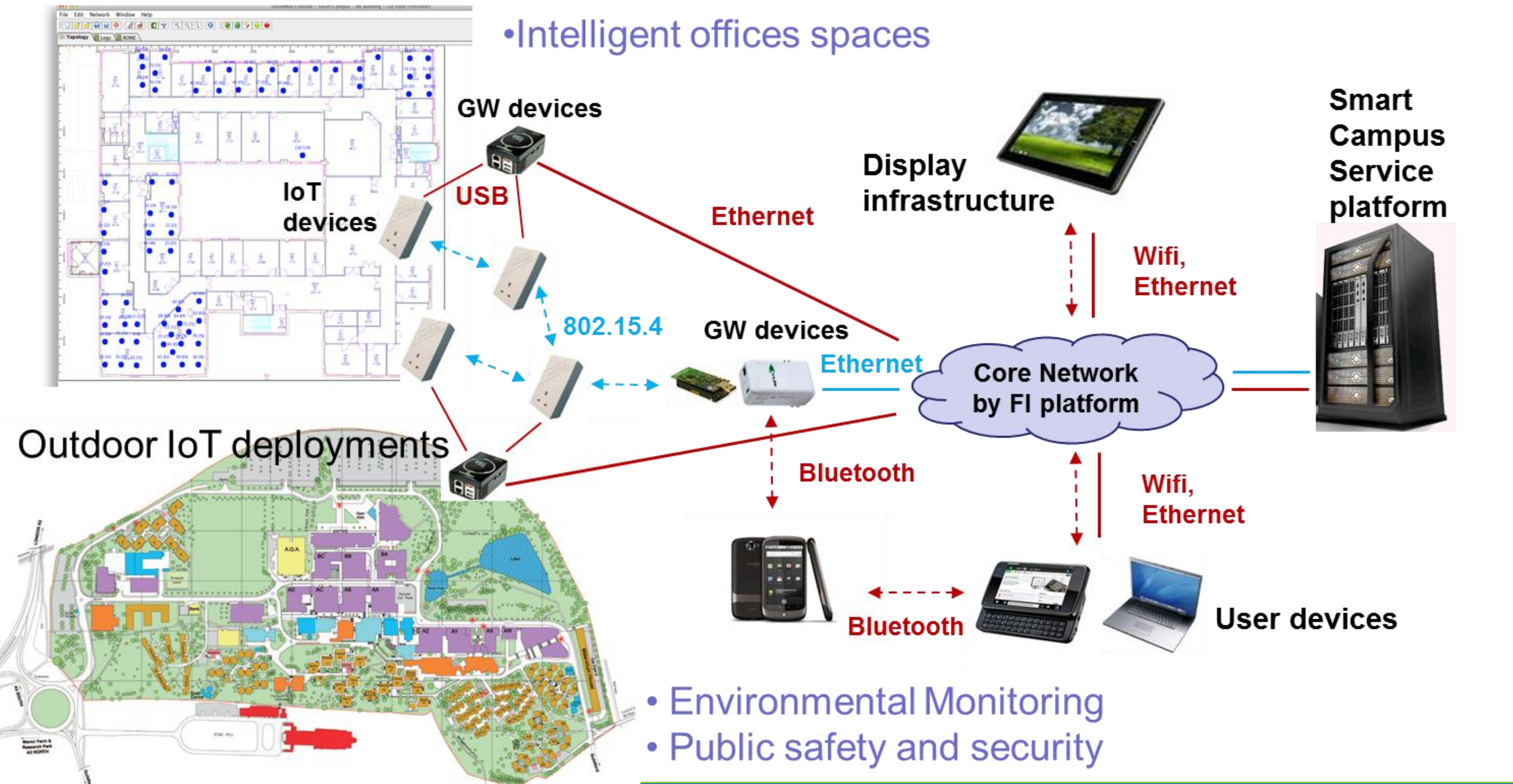
Nati, M., Gluhak, A., Abanger H. and Headley, W. (2013) *SmartCampus: A user-centric testbed for Internet of Things experimentation*. *IEEE Global Wireless Summit 2013 (GWS-2013)*, Atlantic City, New Jersey, USA, June 24-27, 2013

Guildford deployment

Smart Campus deployment

Indoor IoT deployments

•Intelligent offices spaces

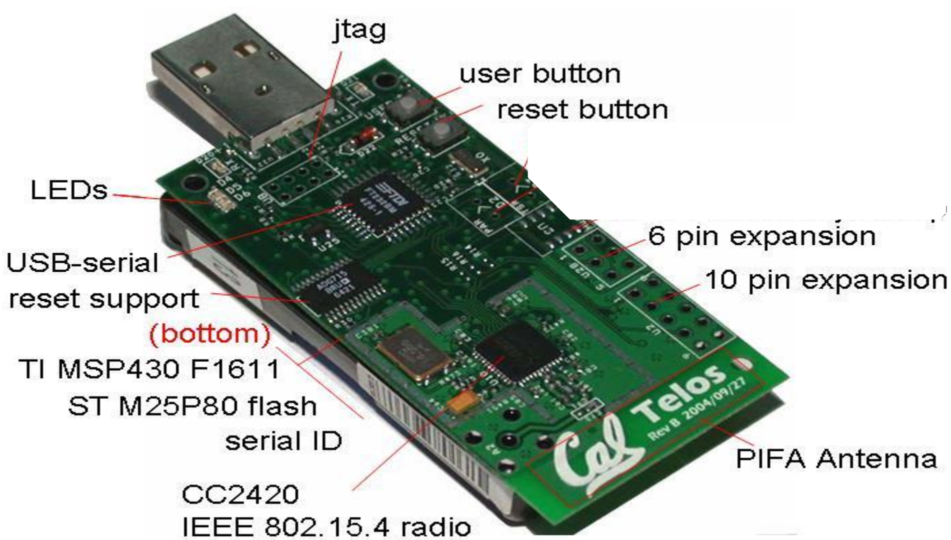


Outdoor IoT deployments

- Environmental Monitoring
- Public safety and security

Indoor IoT nodes

- Based on TelosB mote platform
  - 16 bit MCU, 802.15.4 radio in 2.4GHz ISM band
  - Max transmit power is 1mW

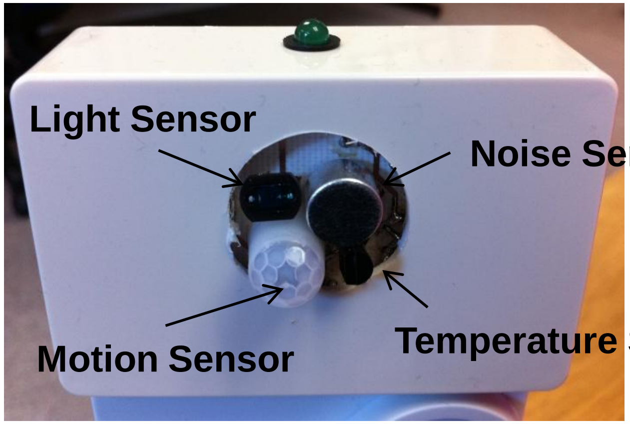


Multimodal sensing unit

- Energy meter measurement unit
  - Off the shelf state of the art meter high precision meter
  - Max sample rate 1 sample every 2 sec

- Features measured are:
  - Power
  - Reactive power
  - Current
  - Phase
  - Voltage/RMS
  - Time operational since connection

LED (activated when data is being taken)



Sensor Suite

(Vibration sensor not shown)

- Sensor suite
  - Light
  - Noise
  - PIR
  - Temperature
  - Vibration

Actuation support

- Turn on/off power socket of energy meter
  - Control attached loads
- SACCOM LED panel
  - Connected via optical coupling to LED of IoT node
  - Provide visual feedback to users via 9 LEDs
  - Driver support on TelosB

- User feedback via external keypad
  - USB connected to GW device

GW tier nodes

- Guru plug server
  - ARM5 CPU, @1.2GHz
  - Linux 2.6.32
  - 512MB RAM and 512MB Flash
  - IO: Ethernet, Wifi, Bluetooth, 2USB

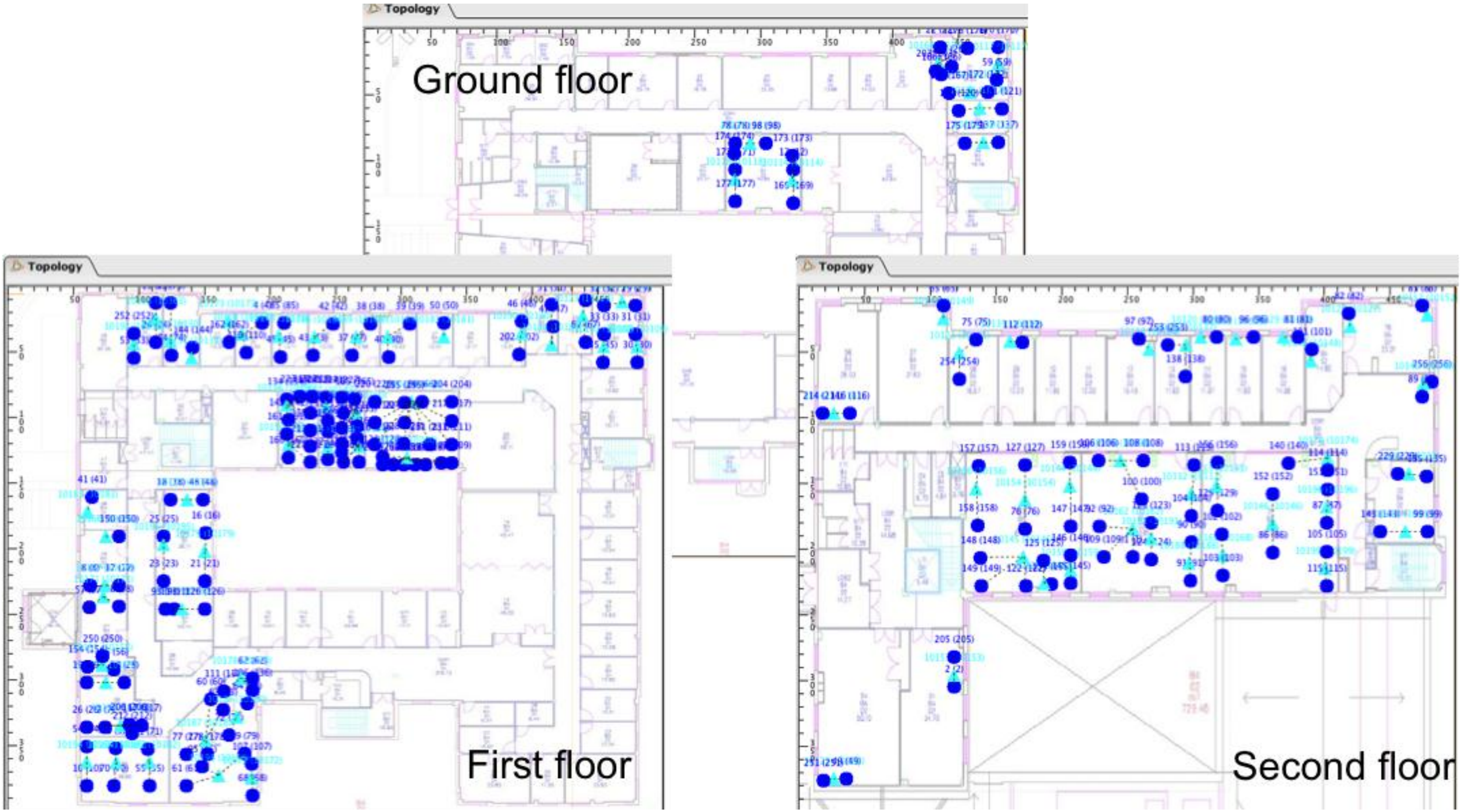
- Use for experimentation
  - Current implementation requires ssh/sftp access to install experimentation codes
  - Other possibility is the use of VTDs [new feature – not tested yet]

Server tier nodes

- Testbed management server
  - Linux based high end server hosting web server front end and experimentation DB

- Application support server
  - Private cloud infrastructure
    - 10 High End Servers (12 Xeon Cores, 24GB RAM each)
    - VMware Cloud Computing Platform
    - 8TB of Storage

Indoor deployment overview



Other IoT nodes

- QR/Code tags across entire campus
  - Unique URL encoded
  - Location data base

- Deployment points
  - Building doors
  - Important intersections
  - Bus stops
- Example of URL encoded
  - http://surrey.ac.uk/apps/?tagid=52050b4da6259

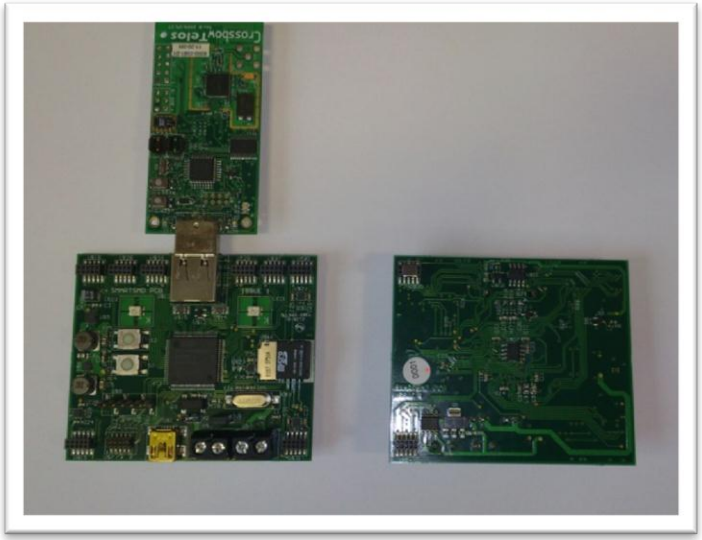
Campus deployment

Other experimentation nodes

- Mobile or “adhoc“ experimentation nodes
  - Android based Smartphones (HTC One S, Sony Xperia S)
  - Advanticsys XM1000 motes with sensor boards (TelosB clone with larger memory)
  - Active RFID tags (OpenBeacon) and reader infrastructure
- Fixed infrastructure
  - Android based tablet devices on walls in corridors (Samsung Galaxy Tab10.1)
  - IP dome cameras

SmartEye (to be deployed)

- Embedded GW/observer platform for outdoor environment
  - Communication with Sensors over USB
    - Debug Information exchange
    - Reprogramming, Turn off/on
  - Communication with Servers
    - Debug Information Streaming
    - Upload of Images (ext. Flash and MicroSD)
  - Observational features
    - Energy profiling (ADC)
    - Virtual Sensor Events injection (DAC, GPIO)
  - Other features
    - Switches for turn off peripherals
    - Li-ion battery operated
  - New SMD version to incorporate
    - Bluetooth Low Energy module
    - GPS module



Summary of HW (1/2)

Type of device	Details	Nature	IoT nodes / unit	Units	Total
Mote platform	TelosB with 802.15.4 radio	Fixed	1	200	200
Multimodal sensing units	Custom design, integrating Energy meter, light, noise, temperature and PIR sensor	Fixed	5	(200)	(1000)
Embedded gateways	GuruPlug servers	Fixed	1	100	100
Smart displays	Wall mounted Android tablets, Samsung Tab10.1	Fixed	1	10	10
Smartphones	HTC One S with proximity, gyro, accelerometer, magnetometer, mic, 2 cameras	Mobile	1 + 7	25	25 (200)
Smartphones	Sony Xperia S with proximity, gyro, accelerometer, magnetometer, mic, 2 cameras, NFC	Mobile	1 +8	5	5 (45)
Total IoT devices					**540 (1555)**

Summary of HW (2/2)

Type of device	Details	Nature	IoT nodes / unit	Units	Total
Mote platform	Advanticsys XM1000	Mobile	1	100	100
Sensor board	Temperature light and humidity sensor for Advanticsys mote	Mobile	3	100	300
Active RFID tags	OpenBeacon proximity tag	Mobile	1	50	50
Active RFID readers	OpenBeaon Ethernet EasyReader PoE II	Fixed	1	5	5
Campus NFC tags	NFC Type2 tags	Fixed	1	300	300
Total IoT devices					**455 (755)**

Upcoming deployment:

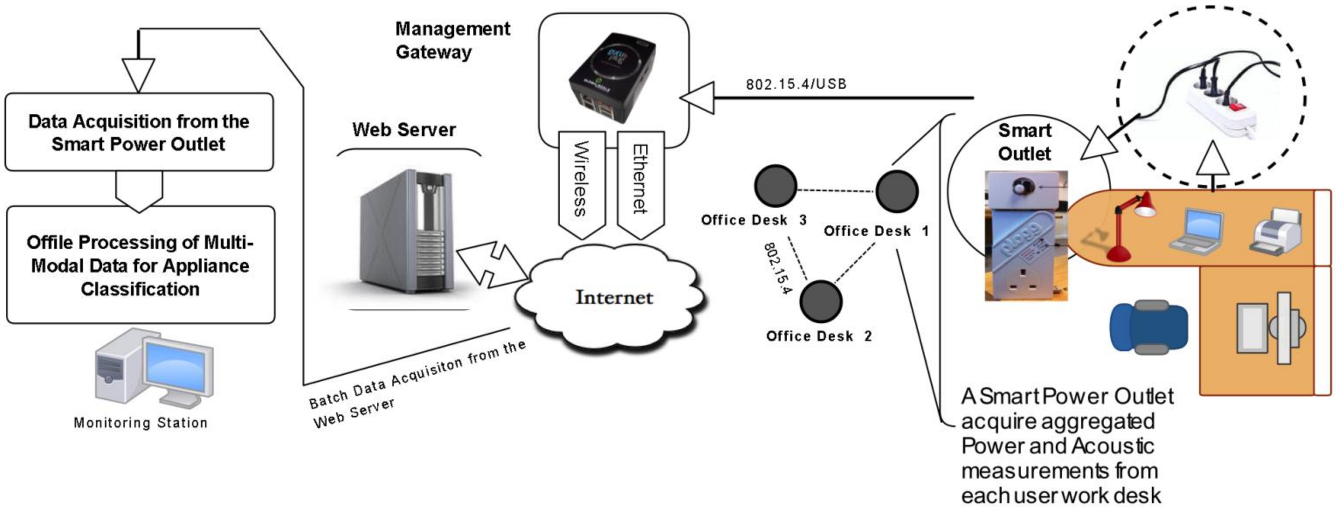
Type of device	Details	Nature	IoT nodes / unit	Units	Total
SmartEye	Outdoor testbed observer and enhanced IoT nodes	Fixed	1	10	10
IP cameras	Dlink DCS 6610	Fixed	1	5	5

Example use cases

- Load disaggregation of low power appliances
- Human dynamics in office spaces
- SACCOM - Soft actuation

Low power appliance monitoring

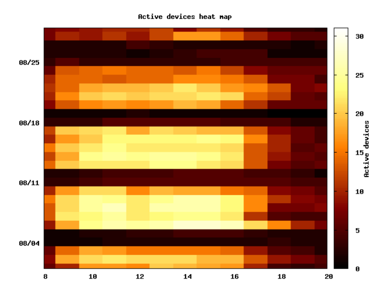
- Evaluate effectiveness of proposed method based on Factorial Hidden Markov Models (FHMM) for low power appliance monitoring
  - Most common office appliances, feature concatenation
  - Assessment of classification accuracy for both binary and multi-state device operation on 10 different test cases
  - Online demonstration

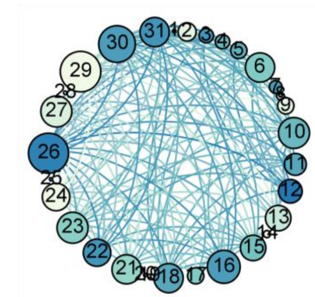


Zoha A., Gluhak A., Nati M., Imran M.A, Low-Power Appliance Monitoring Using Factorial Hidden Markov Model. In *Proceedings of Eight IEEE International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, Melbourne, Australia, 2013

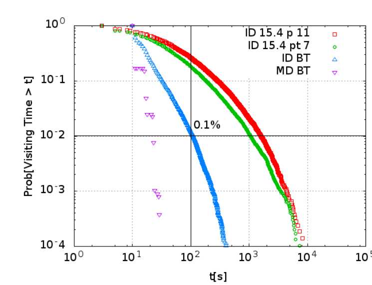
Human dynamics in office spaces

- Movement and co-presence patterns in office
  - Better fine grained traces for mobility and interaction in indoor environment -> short contact durations
  - Compare 802.15.4 with Bluetooth based traces
  - Evaluate assumptions of infrastructure inferred contact patterns





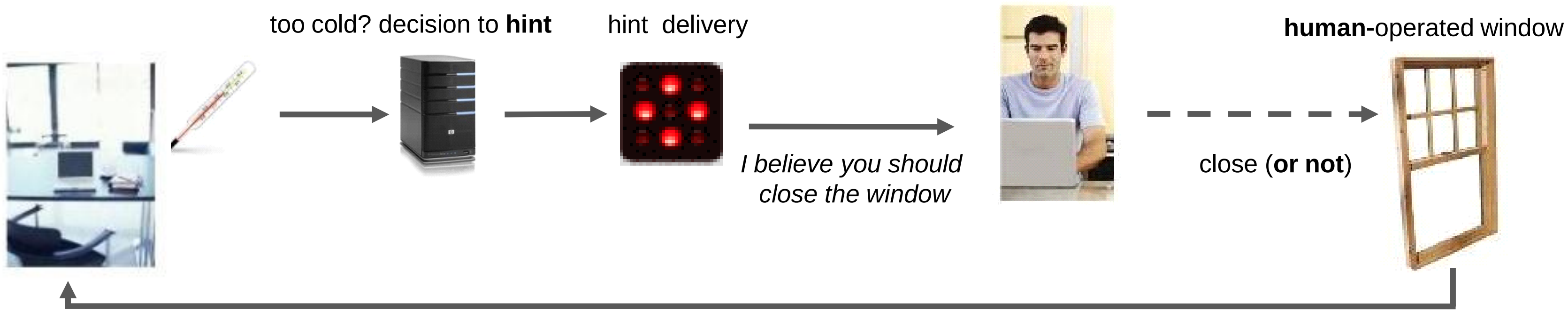
- Experiment
  - 30 participant, 1 month
  - Testbed infrastructure
    - Mobile Smartphone and 15.4 mote carried by participant
    - Fixed : 48 x BT, 20 x 15.4 nodes, covering 3 floors



Nati, M., Gluhak, A., Martelli F., Verdone R. (2013), Measuring and understanding opportunistic co-presence patterns in smart office spaces. *In Proceedings of IEEE International Conference on Internet of Things, iThings 2013,* Beijing, China August 20-23, 2013

SACCOM - Soft actuation (1/2)

- Investigate the concept of soft actuation



- Study details
  - 15 users over 4 weeks period
  - Proposed different soft actuation hints
  - Assessment of user response
    - Experimental observations
    - Post study interview

SACCOM Soft actuation (2/2)

- Provide user feedback

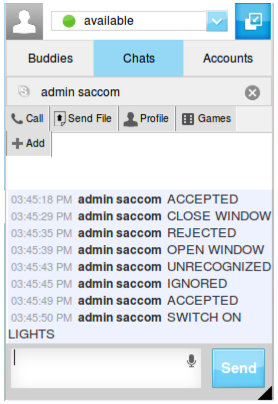
Action	
Open Turn Up Switch On	
Close Turn Down Switch Off	
Device Synch Error	

Target Object	
Window	
Thermostat	
Lights	

- Collect hint reception feedback and user input via key pad

Hint handling outcome	
Ignored	X
Not Recognized	?
Rejected	NO
Accepted	YES

Action / Object	Close Turn Down Switch Off	Open Turn Up Switch On
Window		
Thermostat		
Lights		



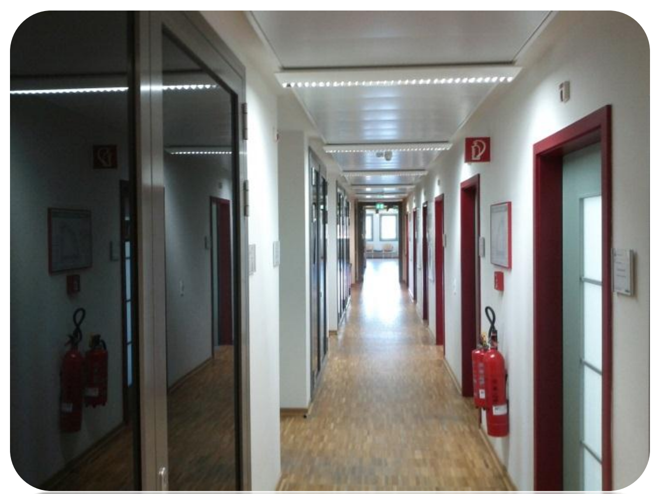
Domaszewicz, J. and Lalis S. (2013), Soft Actuation for Home and Office. *In Proceedings of 9th IEEE International Conference on Intelligent Environments,* Athens, Greece July 18-19, 2013

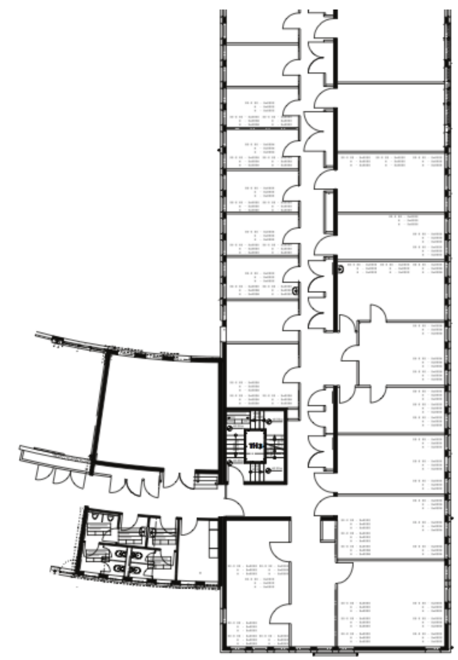
The SmartSantander Experimental Facility

LÜBECK DEPLOYMENT

Lübeck Deployment

- Indoor deployment inherited from WISEBED
- Located inside the office rooms of the Institute of Telematics (ITM) at University of Lübeck





Testbed – Most Common Use Cases

- Evaluation of protocols on all layers
  - Routing protocols
  - 6LoWPAN
  - CoAP
  - …
- Debugging of protocols and applications
  - Using the wired backbone (serial port communication forwarded to testbed user)

Deployment Overview

Type	Sensors	Amount	Extension*
iSense 5139r1	Temp., Light	18	16
iSense 5139r1	PIR, Acc.	18	16
iSense 5148	Temp., Light	9	16
iSense 5148	PIR, Acc.	9	16
iSense Net10	--- (Eth: IPv6<->6LoWPAN)	1	0
TelosB	Temp., Light, PIR, Acc.	54	32
Pacemate	---	54	32
	Sum:	163	128
	Total:		291

The iSense Sensor Node Platform

- Modular, stackable sensor node platform
- Produced by coalesenses GmbH (ITM spin-off)
- Runs iSense OS and Wiselib
- For more information: www.coalesenses.com

iSense Nodes

Core Module 2

Core Module 3

Environmental Module

Security Module

- Core Module 2
  - CPU: Jennic JN5139R1 (Ram 96kB, Flash 128kB, op/sleep 39mA,10myA)
  - Radio: IEEE 802.15.4(2,4 GHz)
- Core Module 3
  - Processor: Jennic JN5148 (128kB RAM, 512kB Flash, 32 Bit RISC Controller, 4-32MHz)
  - Radio: (IEEE 802.15.4 compliant radio, 250kbit/s, hardware AES encryption, ToF ranging engine)
- Environmental Module
  - Temperature & Light Sensors
- Security Module
  - Passive Infrared & Acceleration Sensors

TelosB & Pacemate Nodes

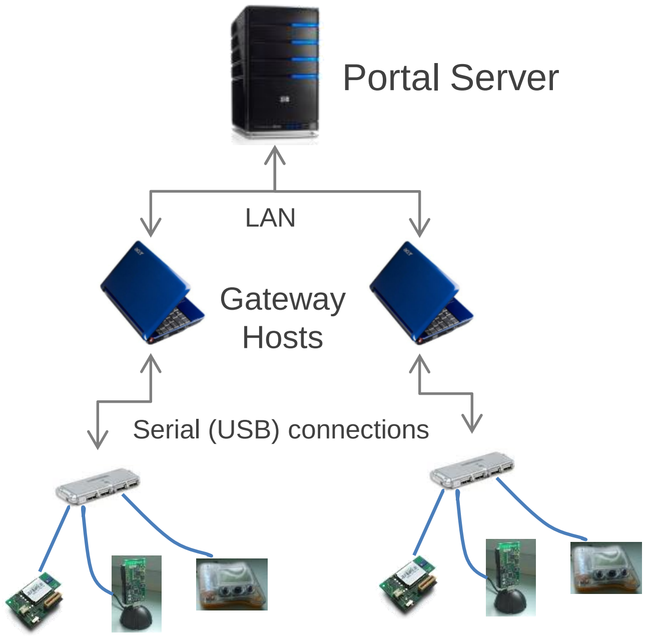
TelosB (Temperature, PIR, Light, Humidity)

- CPU: MSP 430F1611 (Ram 10kB, Flash 48kB, op/sleep 22mA, 5myA)
- Radio: TI CC2420 IEEE 802.15.4(2,4 GHz)
- Sensors: Temp, Light, PIR, Humidity

Pacemate

- CPU: Phillips LPC2136 (Ram 64kB, Flash 256kB, op/sleep 47mA, 60myA)
- Radio: Xemnics RF module (868 MHz)
- Sensors: ---

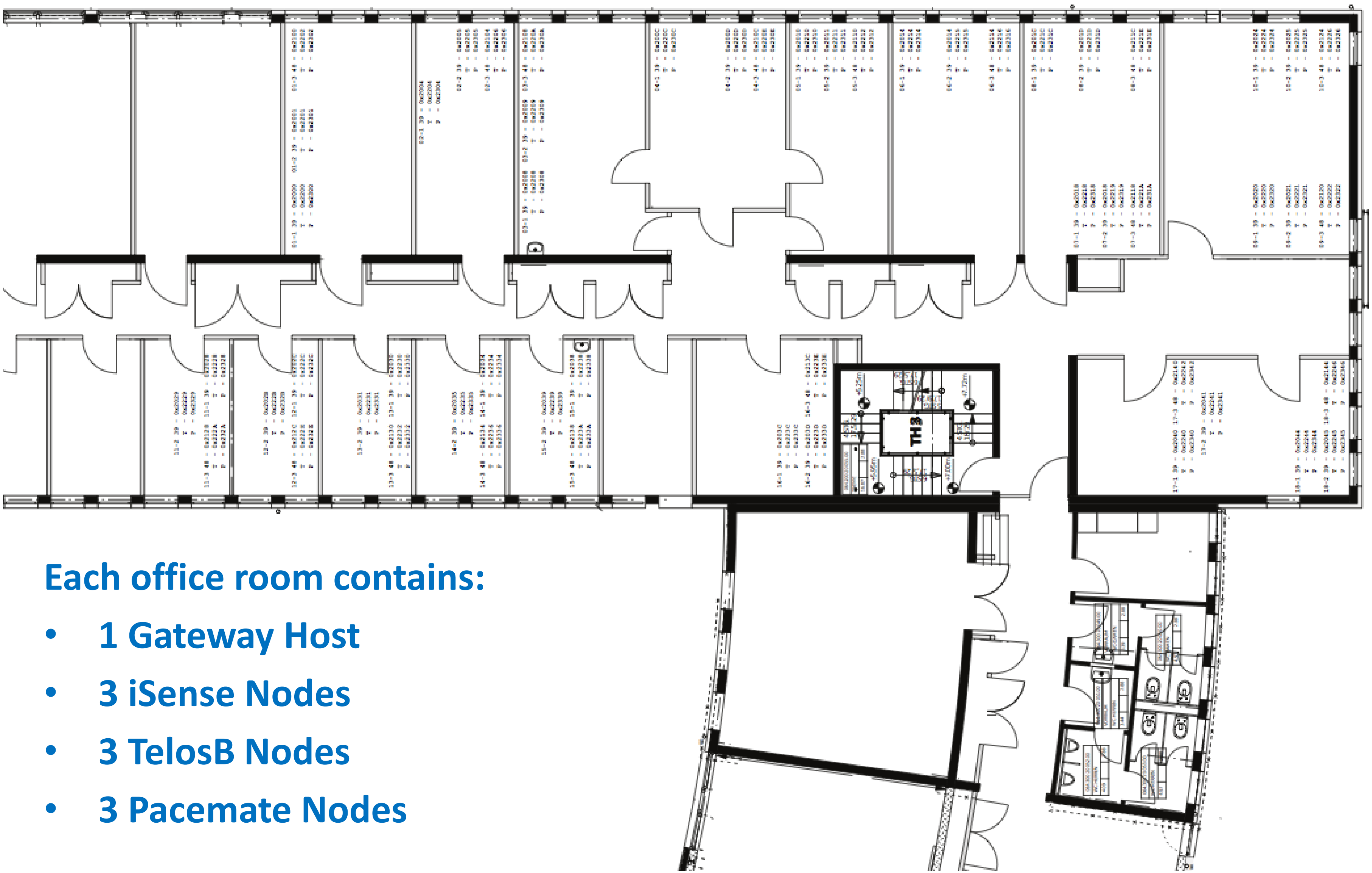
Deployment Hardware Architecture



Hardware Architecture

- **Portal Server**
  - Makes testbed accessible via Web
- **Gateway**
  - Sensor nodes are attached to the gateways via USB
  - Clock-synchronized using the Network Time Protocol (NTP)
  - Three sensor boxes per gateway
- **Sensor Box (left)**
  - 1 iSense node
  - 1 TelosB node
  - 1 Pacemate node

Floor Plan



Each office room contains:

- **1 Gateway Host**
- **3 iSense Nodes**
- **3 TelosB Nodes**
- **3 Pacemate Nodes**

Testbed Access (more later!)

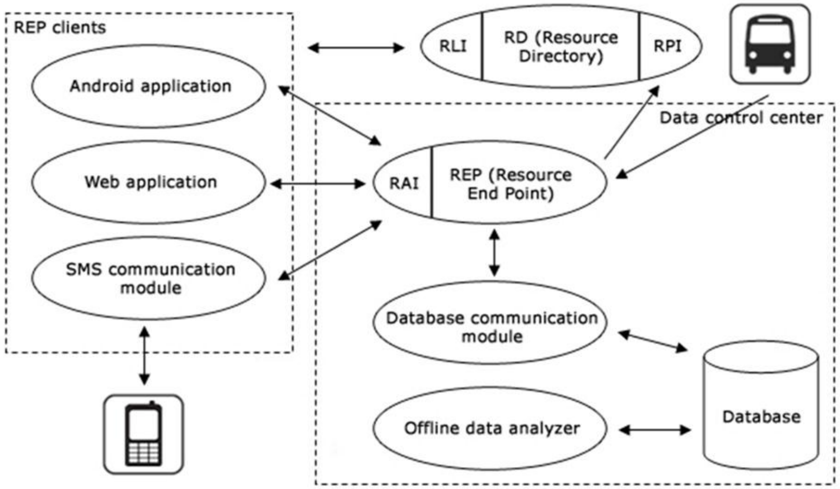
- To get an account follow instructions at:
  http://wisebed.eu/site/conduct-experiments
- Log in to:
  http://wisebed.itm.uni-luebeck.de
- Or use the Experimentation Scripts:
  https://github.com/wisebed/experimentation-scripts

The SmartSantander Experimental Facility

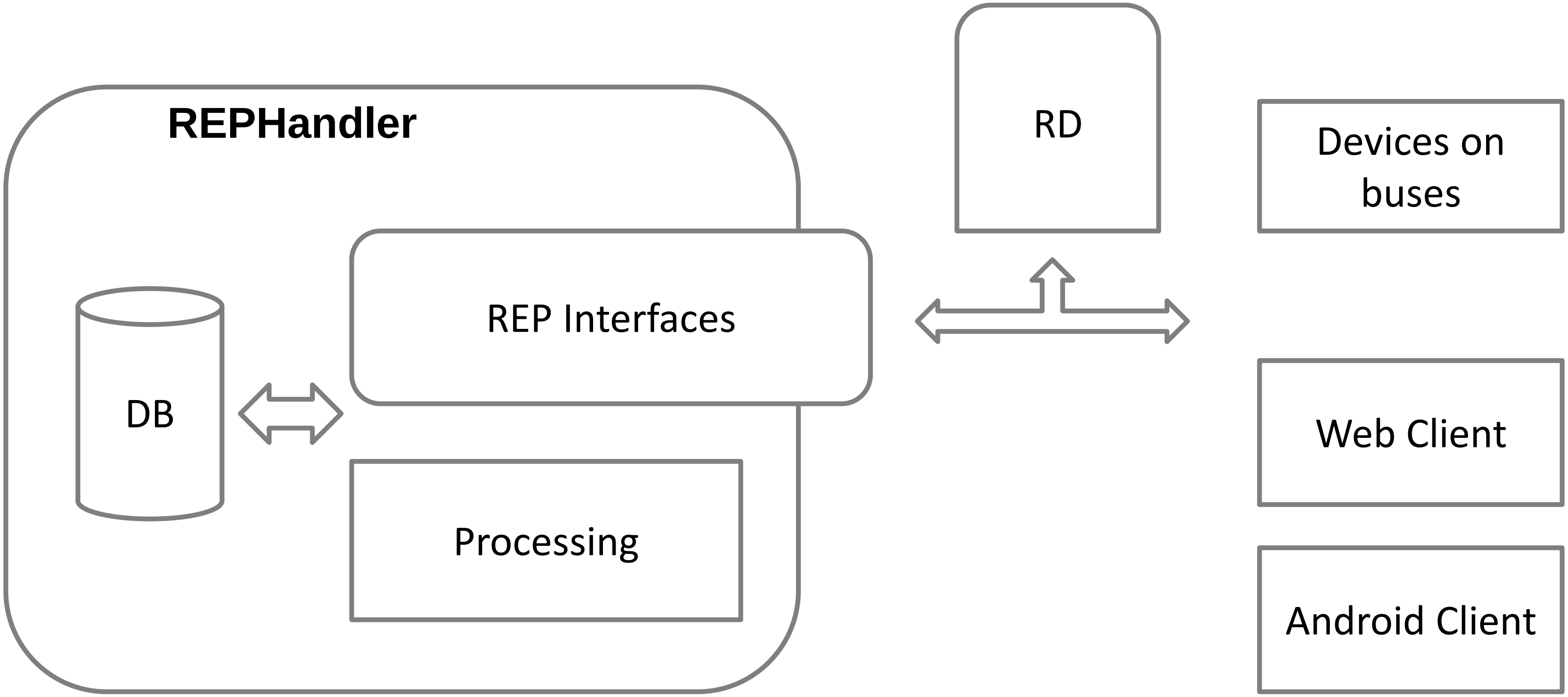
BELGRADE DEPLOYMENT

EkoBus

- Commercial system is deployed in the city of Pančevo (near Belgrade)
- Sensors are located on all public buses
- EkoBus provides bus location monitoring, arrival estimation, environmental monitoring, user interfaces.

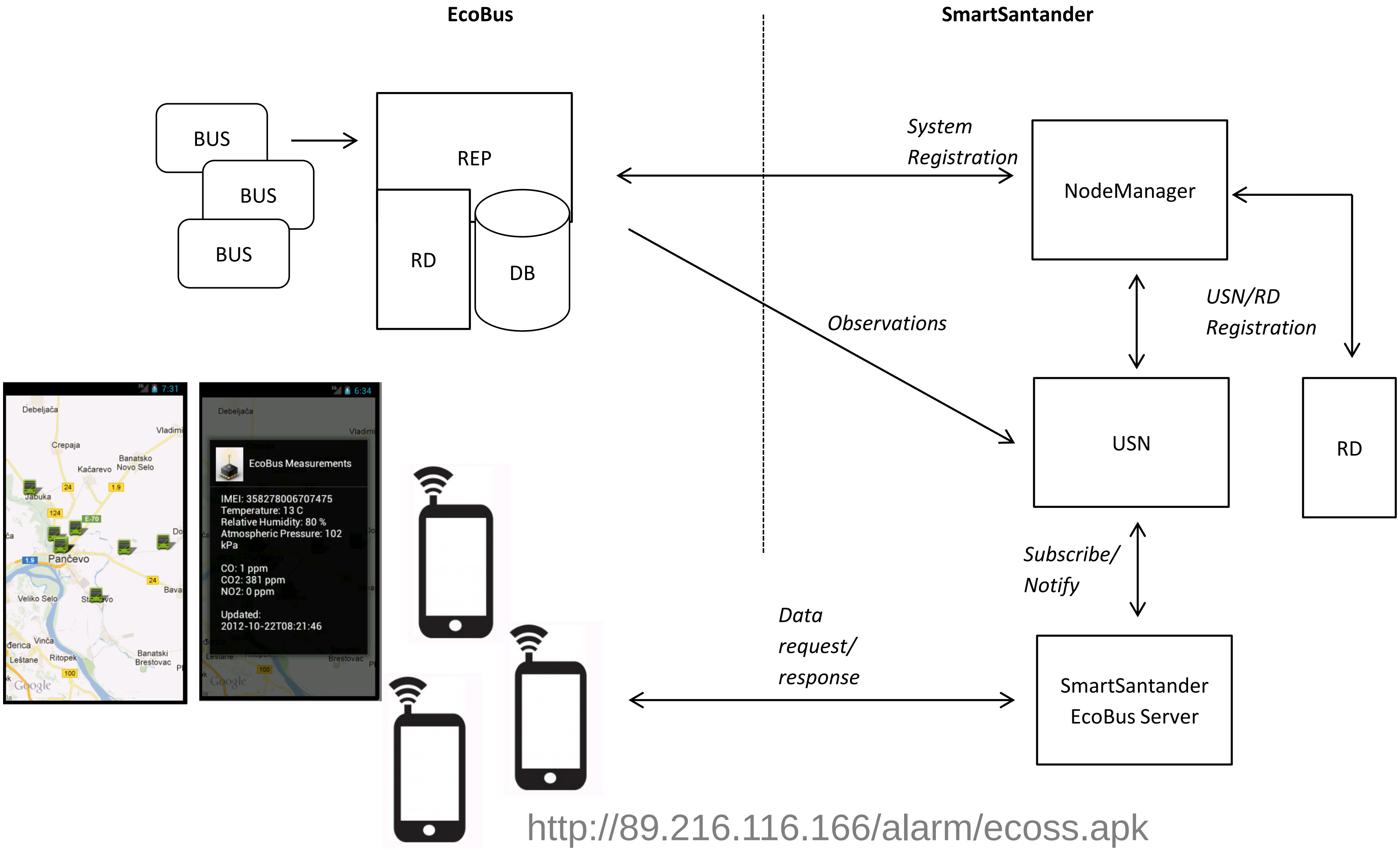


EkoBus system architecture



- REP handler provides system interfaces, data processing, and permanent storage of measurements
- RD keeps descriptions of all available resources in the system
- RESTful implementation

Integration – technical details



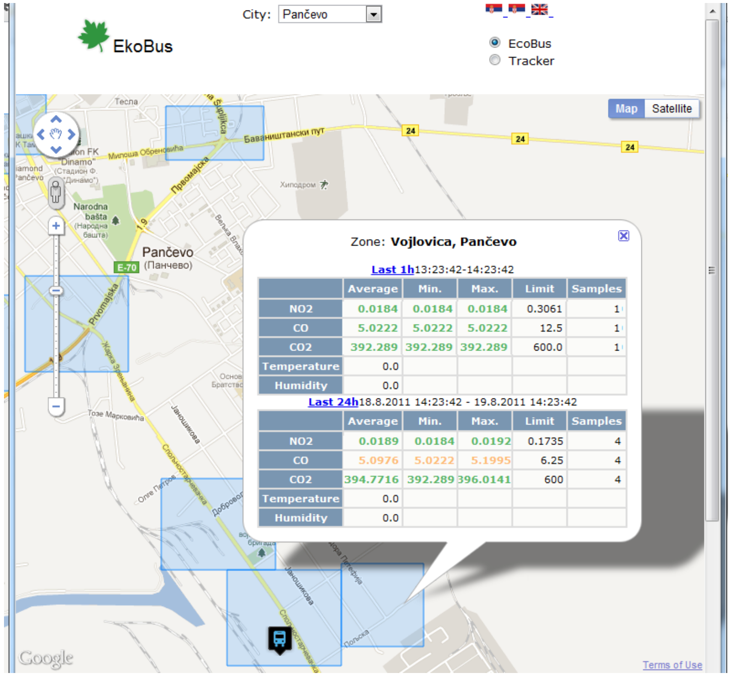
http://89.216.116.166/alarm/ecoss.apk

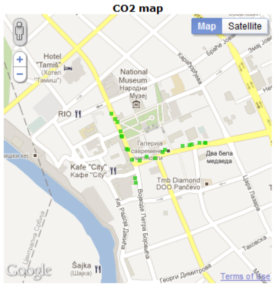
Integration – technical details

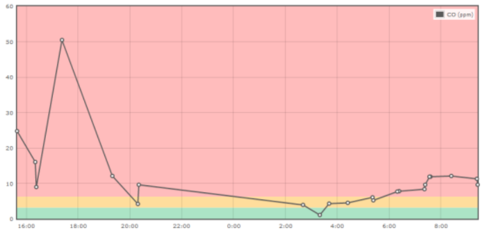
- EkoBus commercial service
  - Not possible to reprogram, no interruption allowed
  - Data collected made available by forwarding to the USN
- To be used by
  - Service developers to develop new applications
  - Citizens/city administration to get indication of environmental conditions (pollution, weather)
- Devices registration & Data forwarding to the USN
- Subscription to observations & Visualization

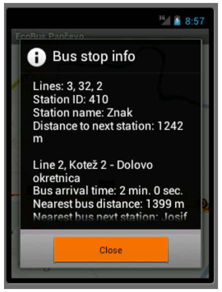
EkoBus

- End user access through web and Android application

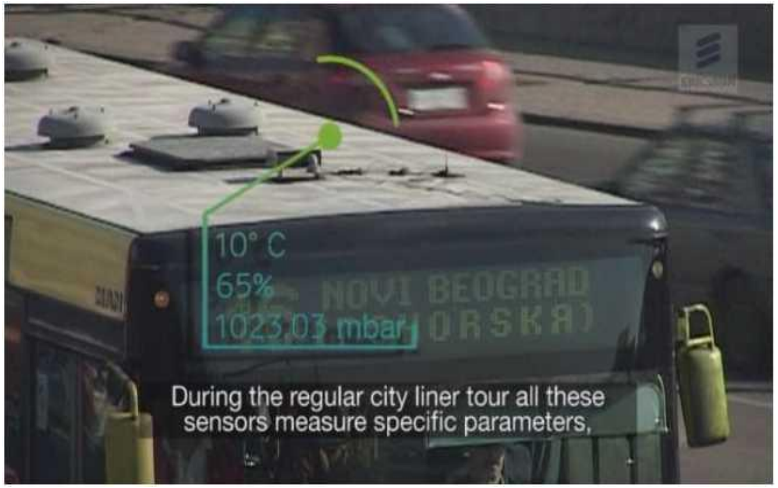








Deployed hardware



- 60 devices are installed
- 50 devices are equipped with GPS GPRS module
- 10 are equipped with sensors for environmental monitoring

Deployed hardware

- Environmental monitoring:
  - CO
  - CO2
  - NO2
  - Temperature
  - Relative humidity
  - Atmospheric pressure
    - NO, SO2, O3, SPL in the latest device version

Deployed hardware

- Several version of devices are designed, especially environmental monitoring devices
- Devices with environmental sensors are one of the most sensitive parts of the system
- Devices use TELIT GM862 for the GPRS communication
- Environmental sensors are installed on the separate board and can be easily replaced

Deployed hardware

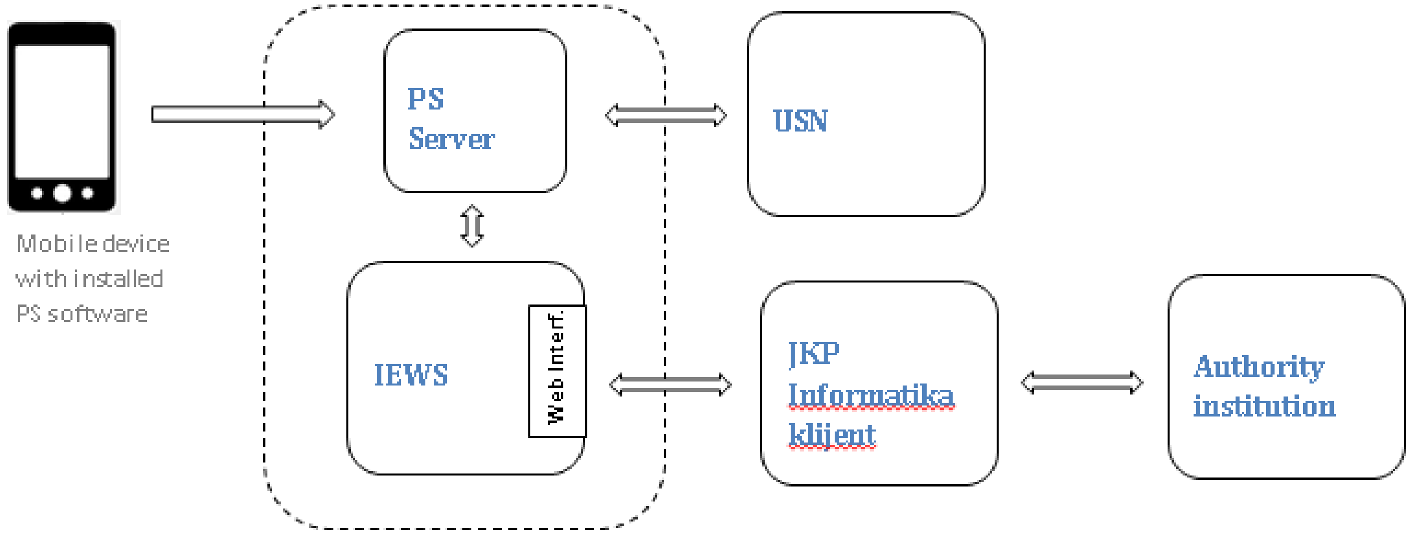
- Replace sensor board easily
- Protect other components

- Extended set of environmental measurements

Participatory sensing

- Participatory sensing integration with JKP Informatika
- JKP Informatika is public utility company
- Maintenance and development of integral information system of the City of Novi Sad,
- Provision of Internet services, Web application design and Internet presentations design and hosting.
- Maintaining and providing telecommunication system services

Block scheme of the integration



- JKP informatika has services for event authority notifying
- Existing PS server component is extended to communicate with IEWS

IEWS

- Informatika Events Web Service provides web service interfaces to reported events
- Event reading
  - REST based interface, JSON object exchanging
- Interface for event/user rating
  - Fetching events by accumulated user/event ratting
  - Remove/keep event after reading
- Event types are agreed with existing set (PS server)
- Rating, user details implemented in mobile app

SMART SANTANDER

Part 2: Experimentation with SmartSantander

Learning goals

- Understand a typical IoT experimentation lifecycle
- Become familiar with the different experimentation tools of SmartSantander
- Understand challenges of user centric experimentation

Outline

1. Experimentation lifecycle
2. Overview of experimentation tools
    1. TMON
    2. WiseGui
    3. Experimentation Scripts
3. User centric experimentation

Part2: Experimentation with SmartSantander

EXPERIMENTATION LIFECYCLE

Experimentation life-cycle

– Resource selection
– Configuration specs
– Definition of KPIs, debug & log info
- Provisioning of images

Specification phase

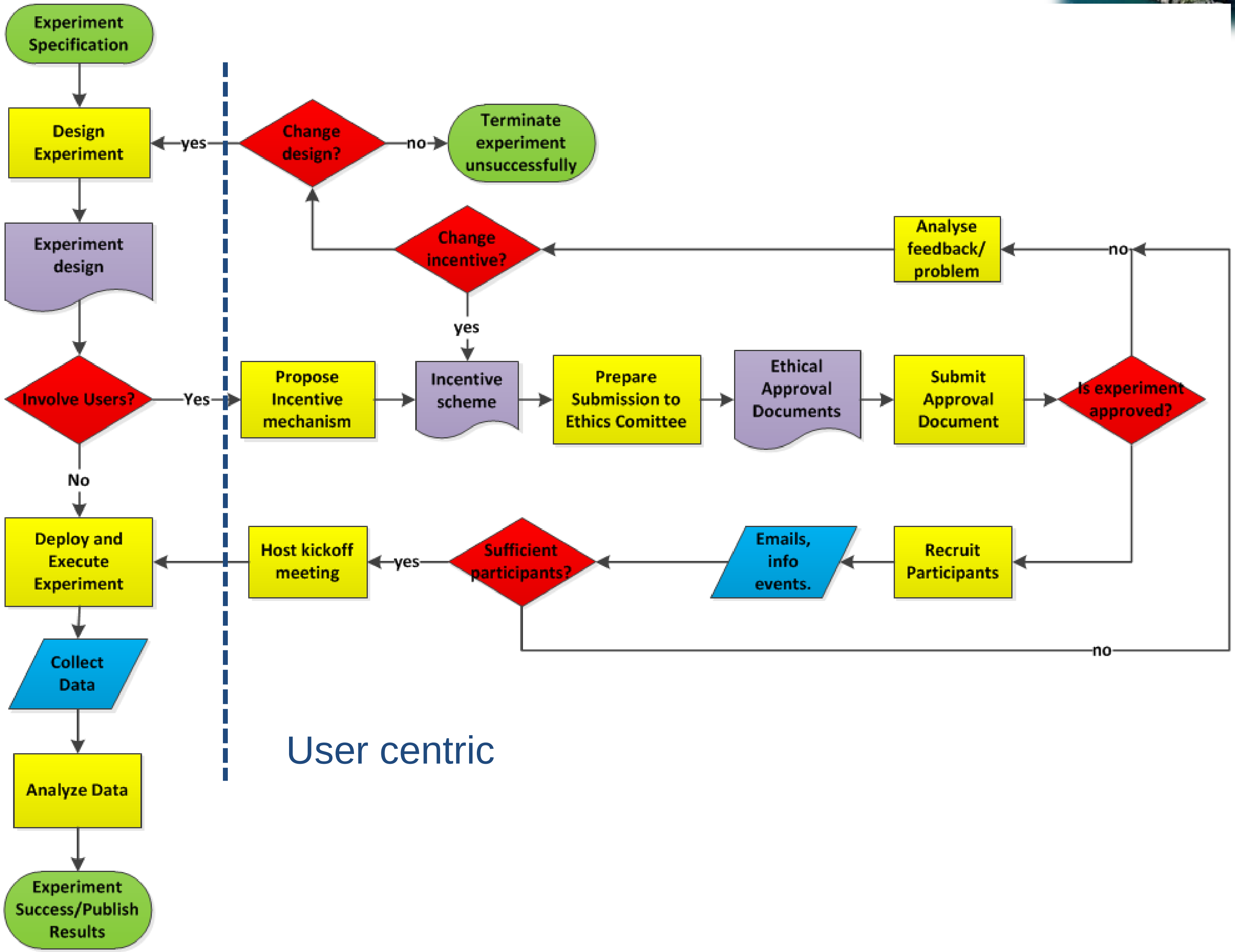
– Reservation
– Scheduling
– Deployment and configuration

Setup phase

– Execution control
– Event injection
– Monitoring
– Data collection
– Logging

Execution phase

Example process



User centric

Specification phase (1/2)

- Identify clear experimentation objectives
  - What is the goal of your experiment?
  - What are the scenarios you want to evaluate and assumptions you take?
  - What are the KPIs you want to collect or debugging parameters?
  - Is it system or service testing?
  - Do real end users have to participate in the experiment?
- Select adequate experimentation resources
  - Derive from above requirements for environment, scale, node capabilities, timing and topology
  - Know the existing testbed topology and characteristics
  - Consider availability of existing resources on testbeds

Specification phase (2/2)

- Select adequate experimentation tools
  - GUI vs. scripting clients etc.

- Configuring your experiment
  - Choose carefully how you want to log statistics and debug output and how to derive them
  - Instrument experimentation code adequately
  - Determine number of runs and duration of experiment interactively based on errors and desired confidence intervals

Setup phase

- Reserve selected resources for experiment
  - 3 methods of reservation supported, hidden behind the reservation API: GoogleCalendar, database or persistent memory
- Provide finalised images for upload to selected resources
  - An images will be mapped to each selected experimentation resource
- Initiate upload of experiment
  - Manual or automatic scheduling

Execution phase (1/3)

- Experiments can be initiated and controlled during reserved time periods
  - Testbed environment provides functions to flash nodes, reset nodes, check if nodes are alive
  - A controller instance is required per experiment to interact with nodes (relay message back and forth)
  - Per default a controller sends data to you
  - Specific experimentation control logic should be implemented in your experimentation code
  - Testbed provides means to send commands to a set of nodes through the controller

Execution phase (2/3)

- Handling traces generated by your nodes
  - You can decide freely the message payload of the trace messages generated by your experimentation nodes
  - Per default messages are sent live to your experimentation client via the controller and stored into an SQL experimentation data base
  - You can see traces live or replay later in the TMON UI from data base
  - Beware of synchronisation of time stamps between nodes

Execution phase (3/3)

- Interacting with your experiment
  - Your logic in the experimentation code defines how your experiment can be controlled
  - In some testbeds auxiliary boards can provide energy measurement information or generation of external sensor stimuli
- Analysing traces and extracting statistics from them
  - TMON offers a plug-in based architecture to provide different views to trace data from the database
  - Traces can be exported to a file and processed outside the environment by analytical tools

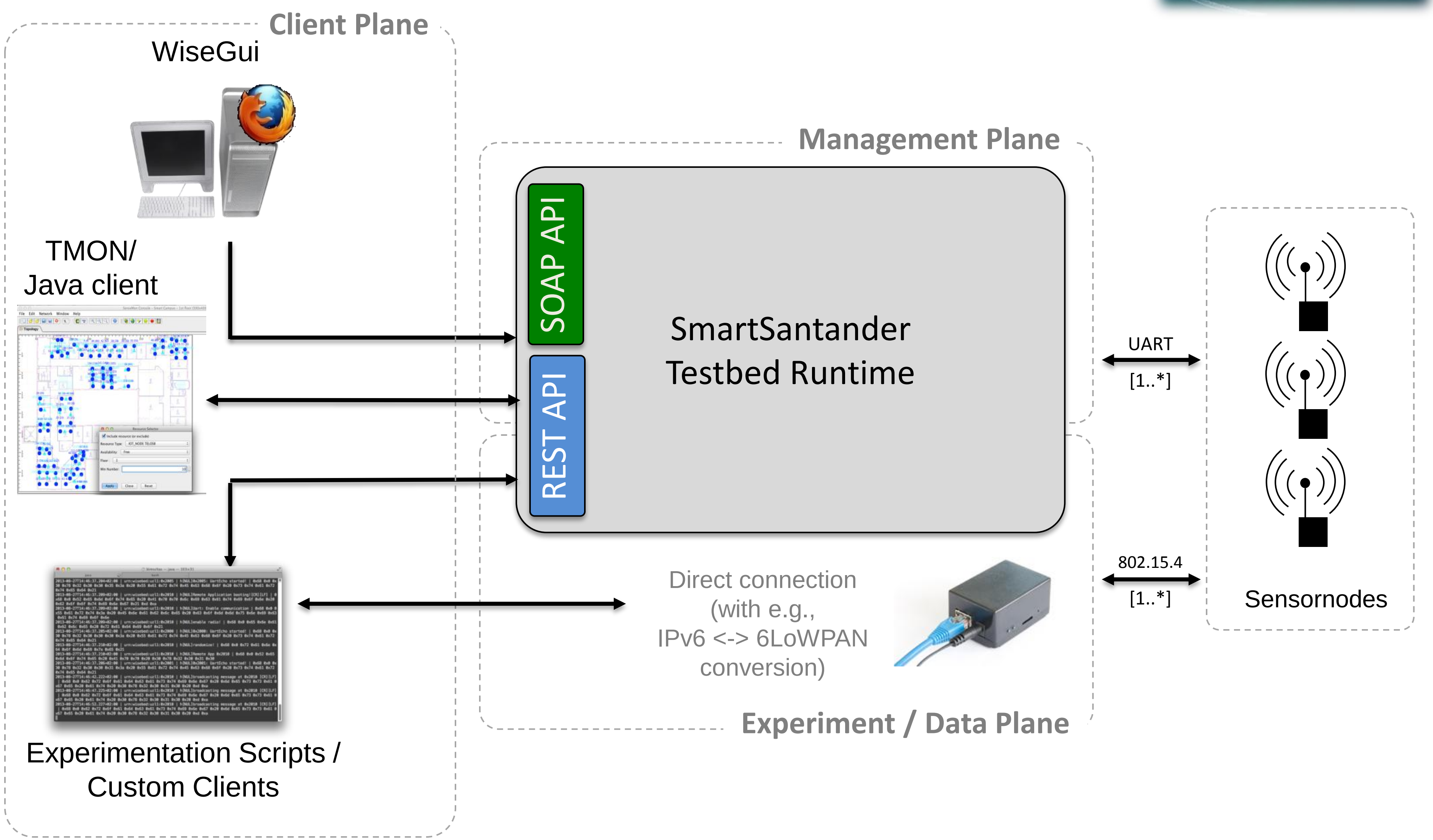
Part 2: Experimentation with SmartSantander

SYSTEMS EXPERIMENTATION

Outline

1. Experimentation lifecycle
2. Overview of experimentation tools
   1. TMON
   2. WiseGui
   3. Experimentation Scripts
3. Challenges of user centric experimentation

2. High Level Overview



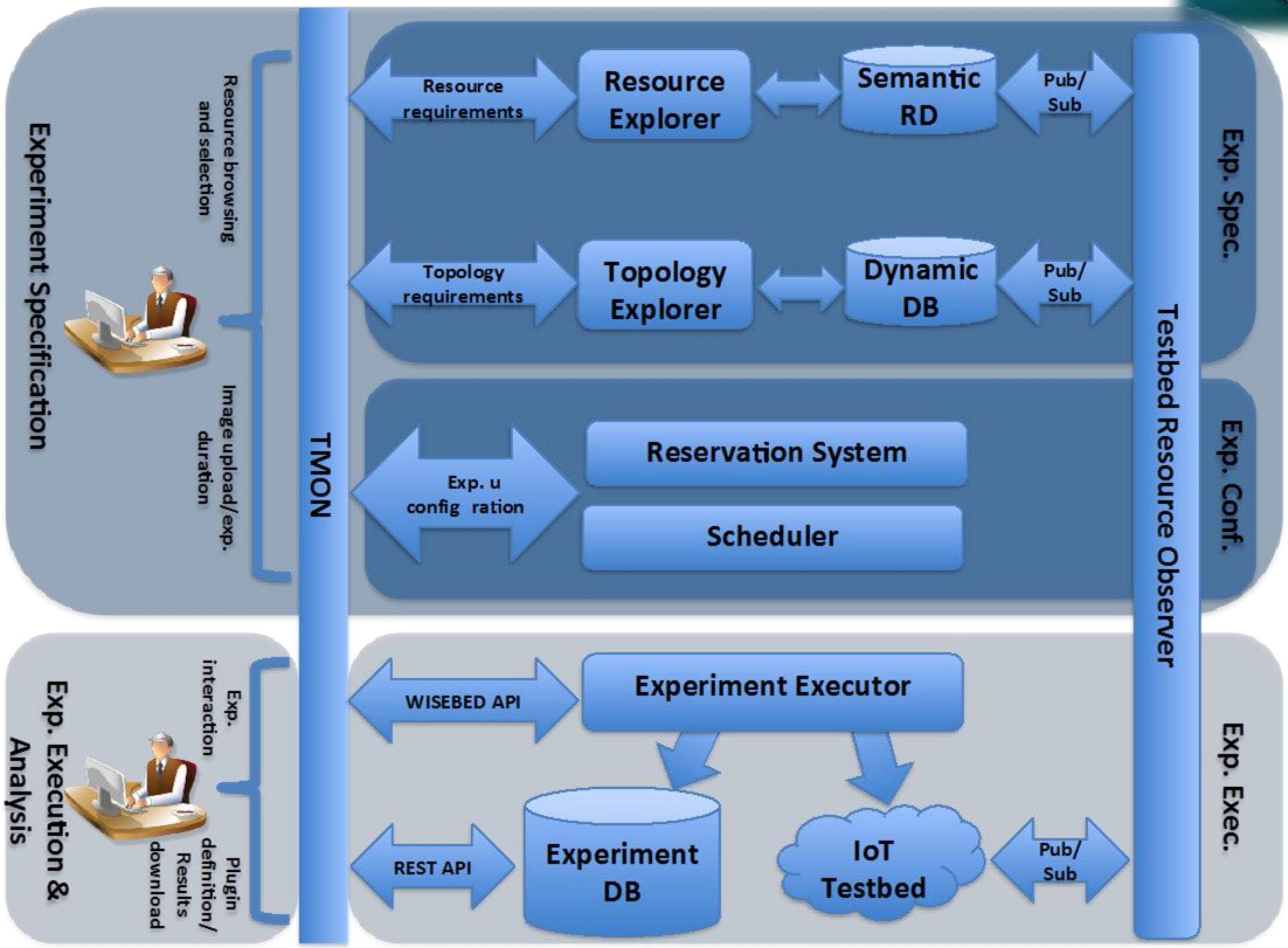
Experimentation with SmartSantander

TMON

TMON overview

- Java based experimentation environment offering the following key features
  - Resource explorer
  - Topology explorer
  - Reservation, reprogramming
  - Experimentation control
  - Visualisation of traces and live results
  - Replay of experiment traces
  - Plugin based architecture for customisation of views and data processing
- Currently supported testbed sites
  - Guildford and Luebeck

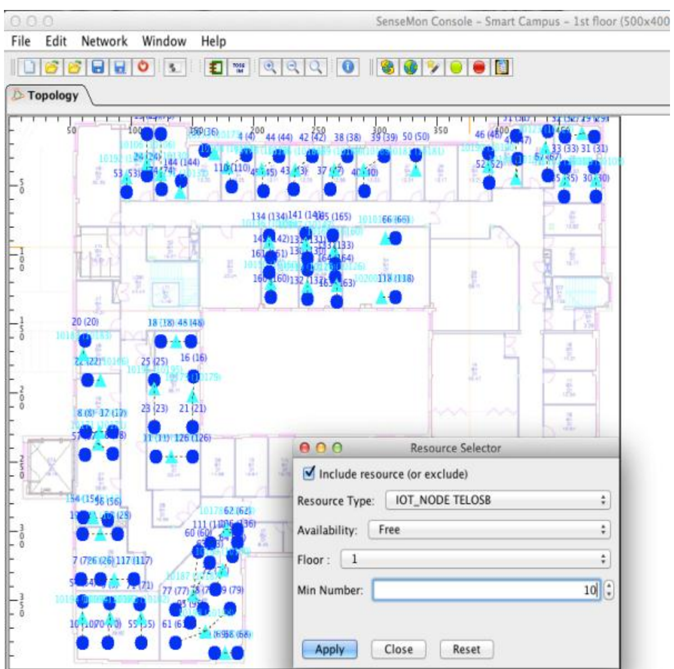
TMON architecture

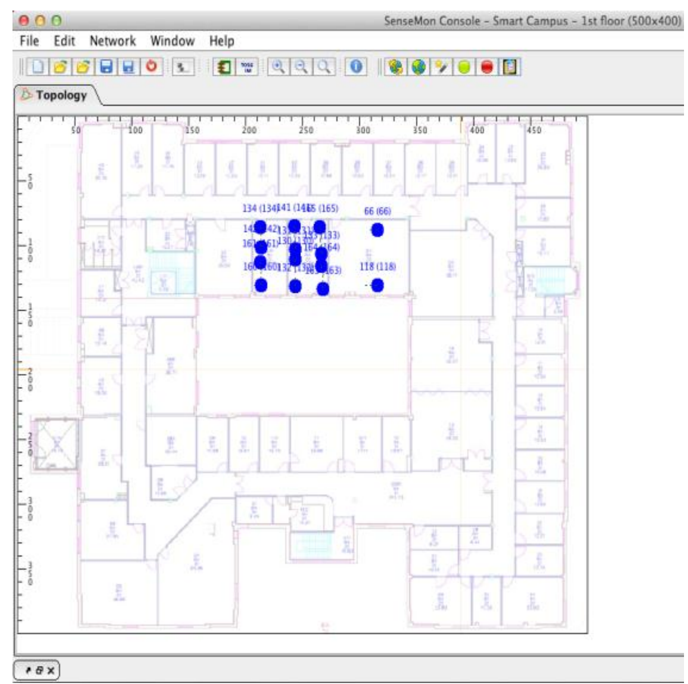


Nati, M., Gluhak, A., Abangar, H., & Meissner, S. (2012). A Framework for Resource Selection in Internet of Things Testbeds. In 8th International ICST Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, TRIDENTCOM 2012 (pp. 224-239). Thessaloniki, Greece

Resource explorer

- Simplifies identification of suitable experimentation resources
- Allows visual selection and queries
  - – Support for semantic queries for resource capabilities

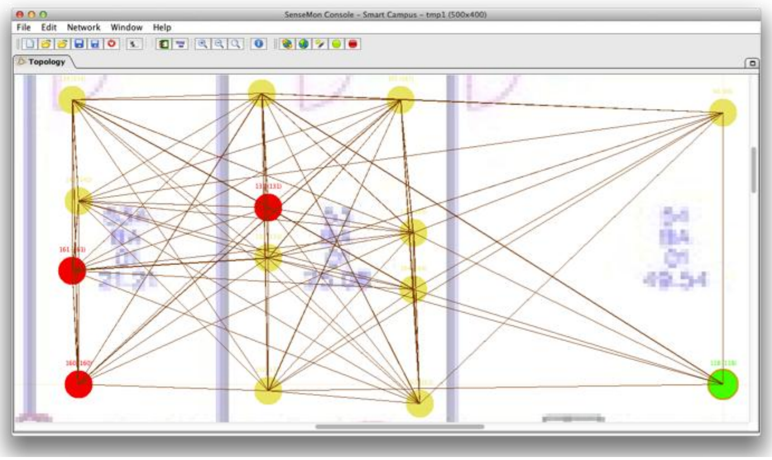




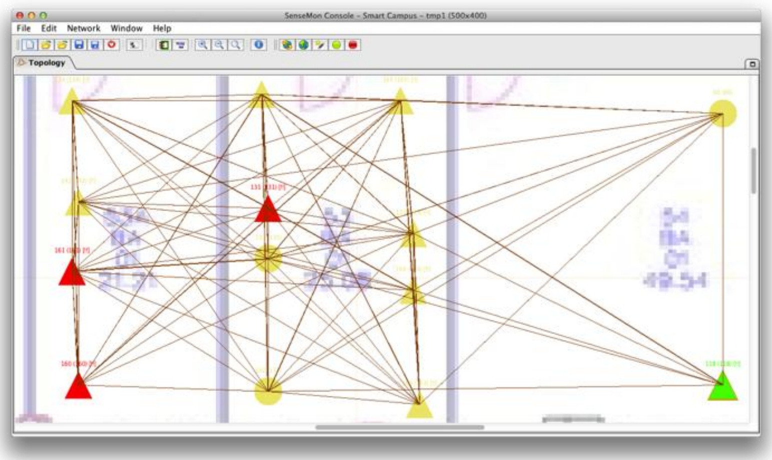
Topology explorer

- Supports selection of nodes based on topology requirements
  - Supports visualization of links for a set of resources (Channel ID, TX Power, Packet Error Rate, LQI, RSSI)
  - Pre-defined topology exploration rules
  - Visualize source of interference (signal strength)

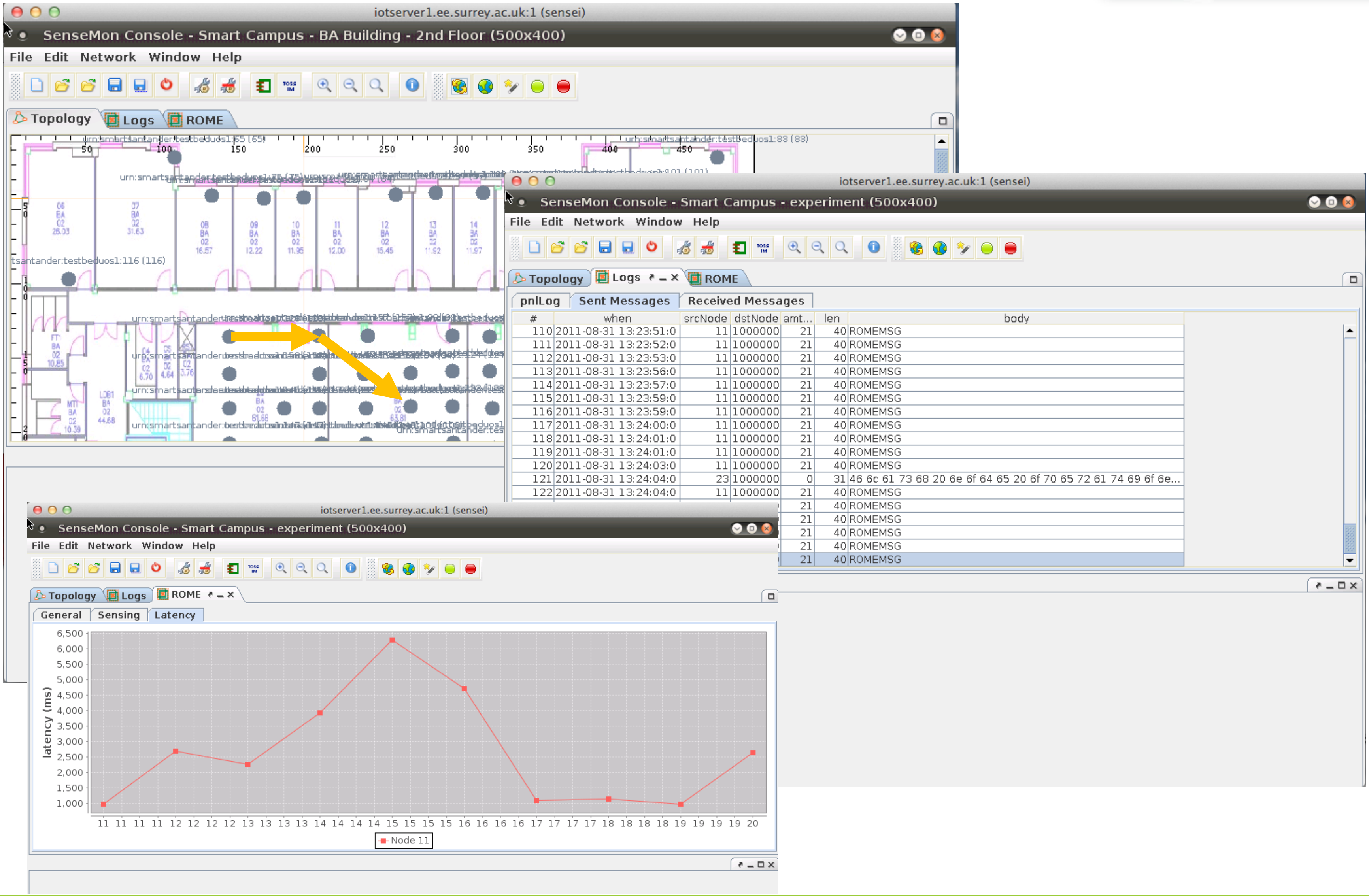
Ch ID 20, TX Power 7, PER 0.5



Ch ID 15, TX Power 7, PER 0.5



Visualisation support



Experimentation with SmartSantander

WISEGUI
LIVE-DEMO (SCREENSHOT WALK-THROUGH)

2.2 WiseGui

- Web-based front end for WSN testbeds based on the SmartSantander/WISEBED REST API
- Used for the following SmS deployments:
  - Santander
  - Lübeck
  - Patras
  - Guildford
- Lübeck Deployment: http://wisebed.itm.uni-luebeck.de

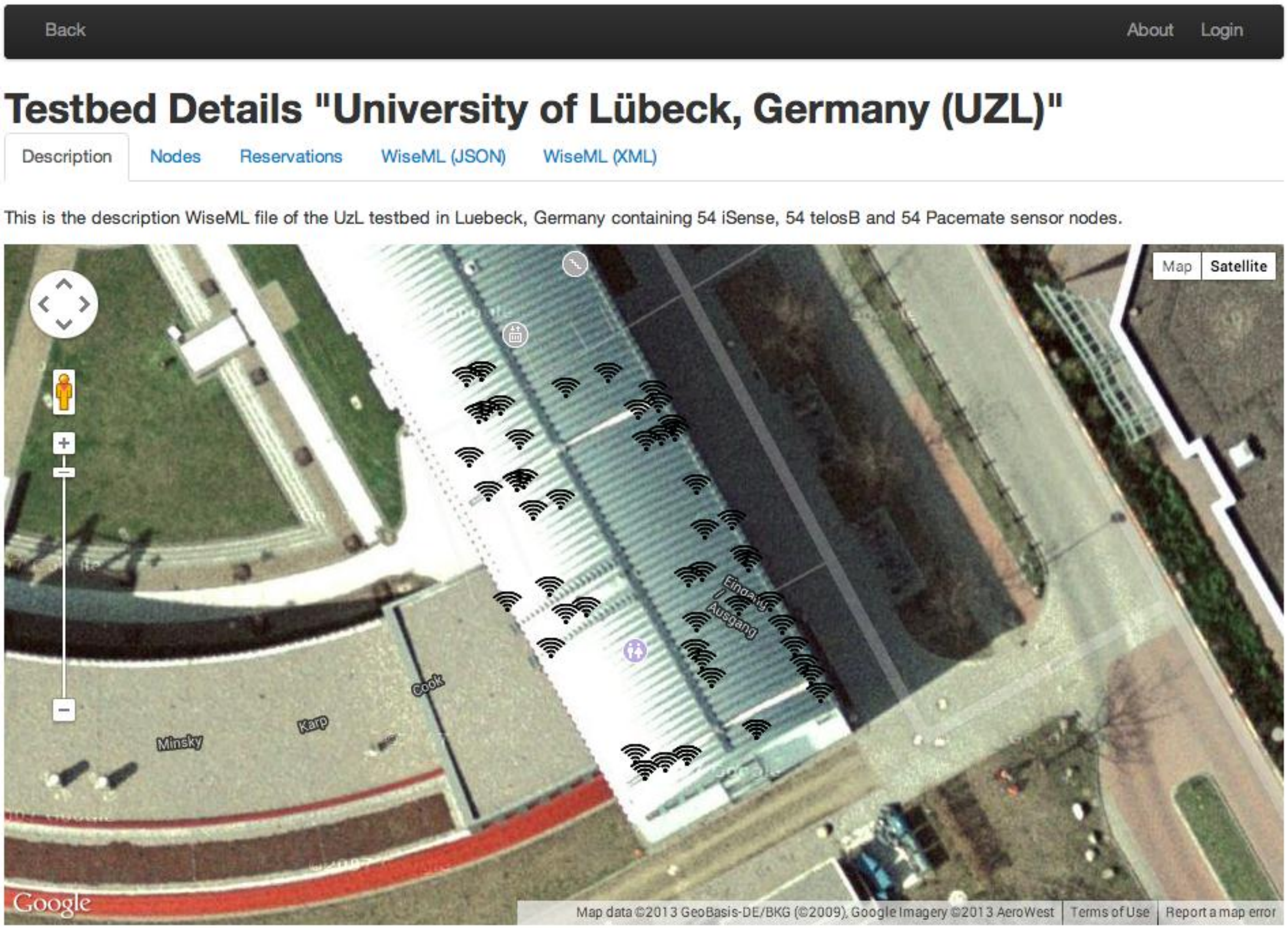
2.2 WiseGui – Testbeds Overview

Testbed Overview | About

Name	URN prefixes	Session Management Endpoint URL
University of Lübeck, Germany (UZL)	urn:wisebed:uzl1:	http://wisebed.itm.uni-luebeck.de:8888/sessions
University of Braunschweig, Germany (TUBS)	urn:wisebed:tubs:	http://wbportal.ibr.cs.tu-bs.de:8080/sessions
Computer Technology Institute & Press (CTI)	urn:wisebed:ctitestbed:	http://hercules.cti.gr:8888/sessions
University of Geneva	urn:wisebed:unigetestbed:	http://testbed.tcs.unige.ch:8888/sessions
Lübeck University of Applied Sciences, Germany (FHL)	urn:wisebed-cosa-testbed-fhl1:	http://cosa-testbed-fhl1.fh-luebeck.de:49021/sessions

Screenshot taken from: http://wisebed.itm.uni-luebeck.de

2.2 WiseGui – Testbed Map View



2.2 WiseGui – Testbed Node Table

Back | About | Login

Testbed Details "University of Lübeck, Germany (UZL)"

Description | Nodes | Reservations | WiseML (JSON) | WiseML (XML)

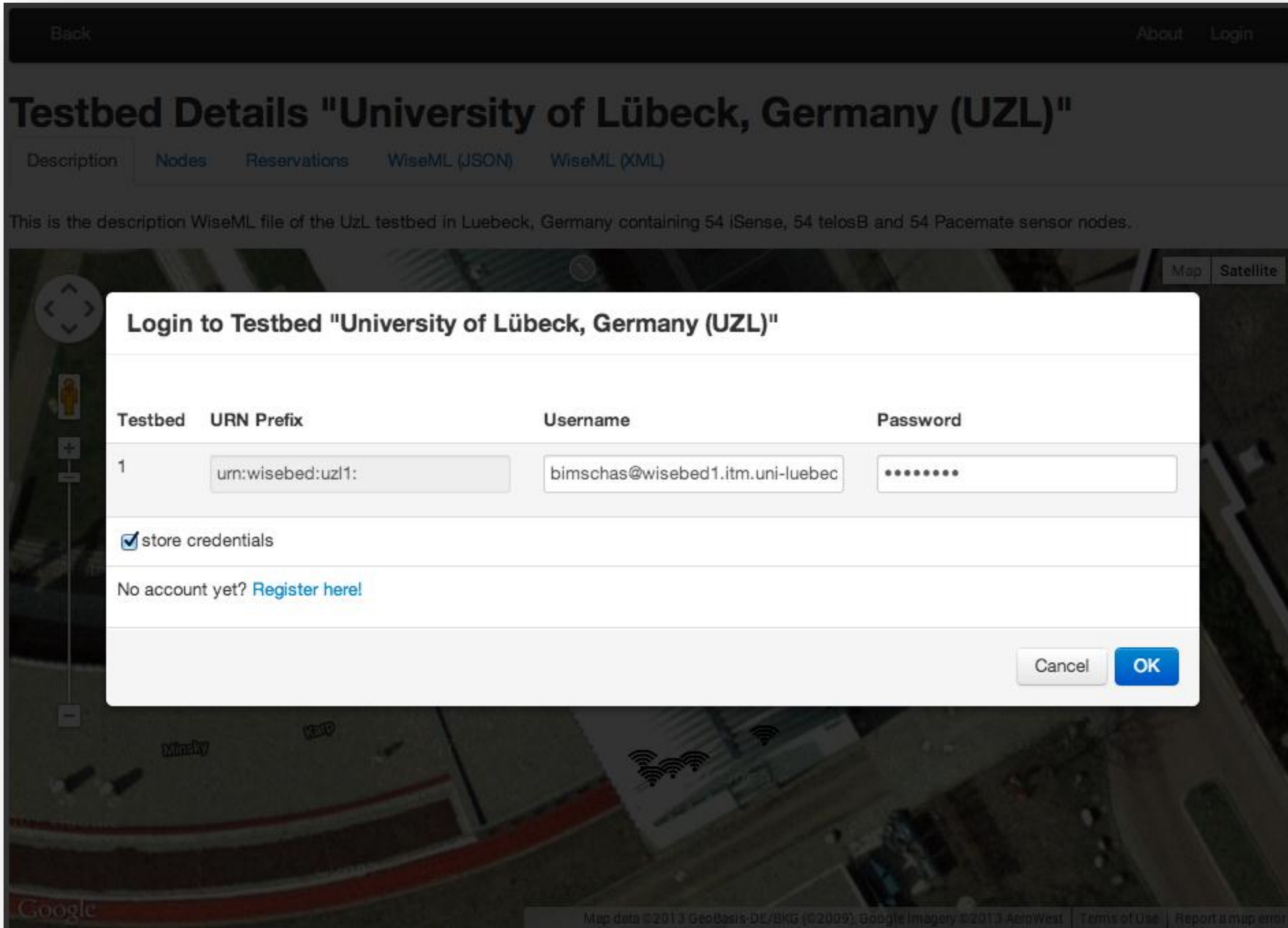
Filter displayed nodes...

✓ Nodes of every type
Only nodes of type telosb
Only nodes of type isense48
Only nodes of type isense39

Advanced

Node URN	Type	Position	
urn:wisebed:uzl1:0x2000	isense39	(31,6,1)	pir,acc
urn:wisebed:uzl1:0x2001	isense39	(31,3,1)	pir,acc
urn:wisebed:uzl1:0x2004	isense39	(27.5,2.5,1)	temperature,light
urn:wisebed:uzl1:0x2005	isense39	(27.5,1,1)	pir,acc
urn:wisebed:uzl1:0x2008	isense39	(25,3,1)	temperature,light
urn:wisebed:uzl1:0x2009	isense39	(25,2,1)	pir,acc
urn:wisebed:uzl1:0x200c	isense39	(17,0.5,0)	temperature,light
urn:wisebed:uzl1:0x200d	isense39	(20.5,1.5,1)	temperature,light
urn:wisebed:uzl1:0x2010	isense39	(14,1,1)	temperature,light
urn:wisebed:uzl1:0x2011	isense39	(14,4,1)	pir,acc
urn:wisebed:uzl1:0x2014	isense39	(13.5,1,1)	pir,acc
urn:wisebed:uzl1:0x2015	isense39	(10.5,3,1)	temperature,light

2.2 WiseGui – Login



2.2 WiseGui – Making a Reservation

Back My Reservations About **Make Reservation** Logout

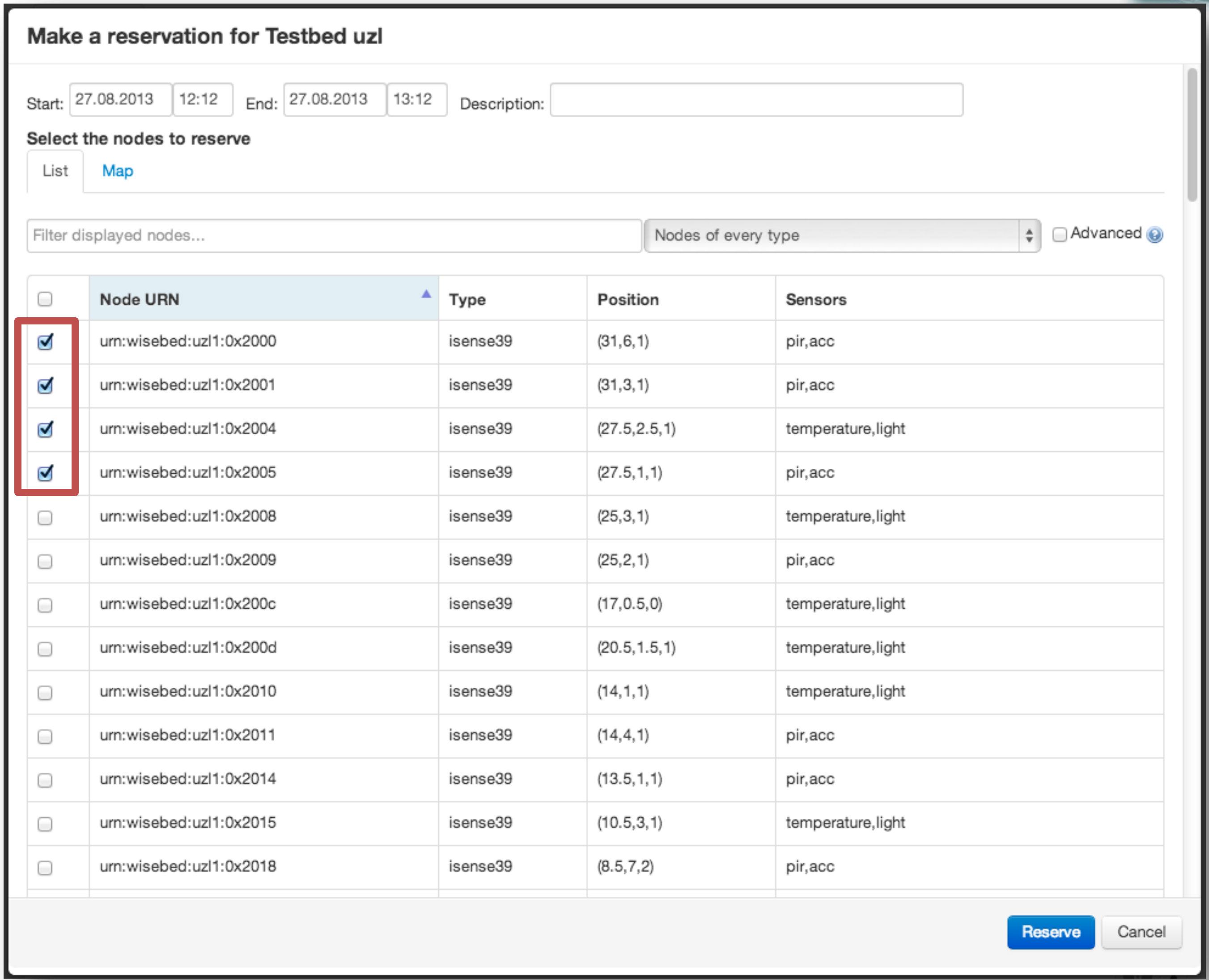
Testbed Details "University of Lübeck, Germany (UZL)"

Description | Nodes | Reservations | WiseML (JSON) | WiseML (XML)

Filter displayed nodes... | Nodes of every type | Advanced

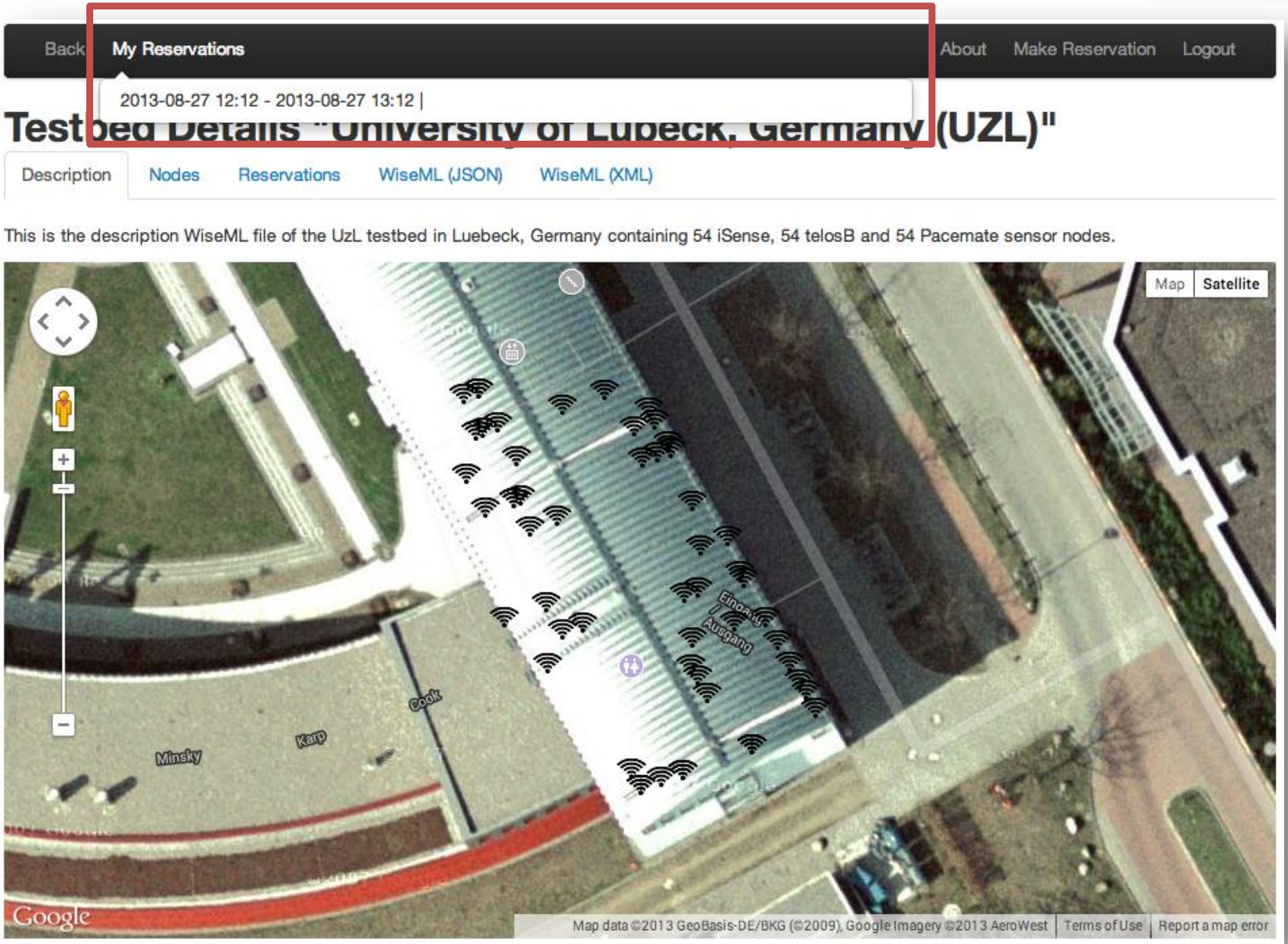
Node URN	Type	Position	Sensors
urn:wisebed:uzl1:0x2000	isense39	(31,6,1)	pir,acc
urn:wisebed:uzl1:0x2001	isense39	(31,3,1)	pir,acc
urn:wisebed:uzl1:0x2004	isense39	(27.5,2.5,1)	temperature,light
urn:wisebed:uzl1:0x2005	isense39	(27.5,1,1)	pir,acc
urn:wisebed:uzl1:0x2008	isense39	(25,3,1)	temperature,light
urn:wisebed:uzl1:0x2009	isense39	(25,2,1)	pir,acc
urn:wisebed:uzl1:0x200c	isense39	(17,0.5,0)	temperature,light
urn:wisebed:uzl1:0x200d	isense39	(20.5,1.5,1)	temperature,light
urn:wisebed:uzl1:0x2010	isense39	(14,1,1)	temperature,light
urn:wisebed:uzl1:0x2011	isense39	(14,4,1)	pir,acc
urn:wisebed:uzl1:0x2014	isense39	(13.5,1,1)	pir,acc
urn:wisebed:uzl1:0x2015	isense39	(10.5,3,1)	temperature,light

2.2 WiseGui – Making a Reservation

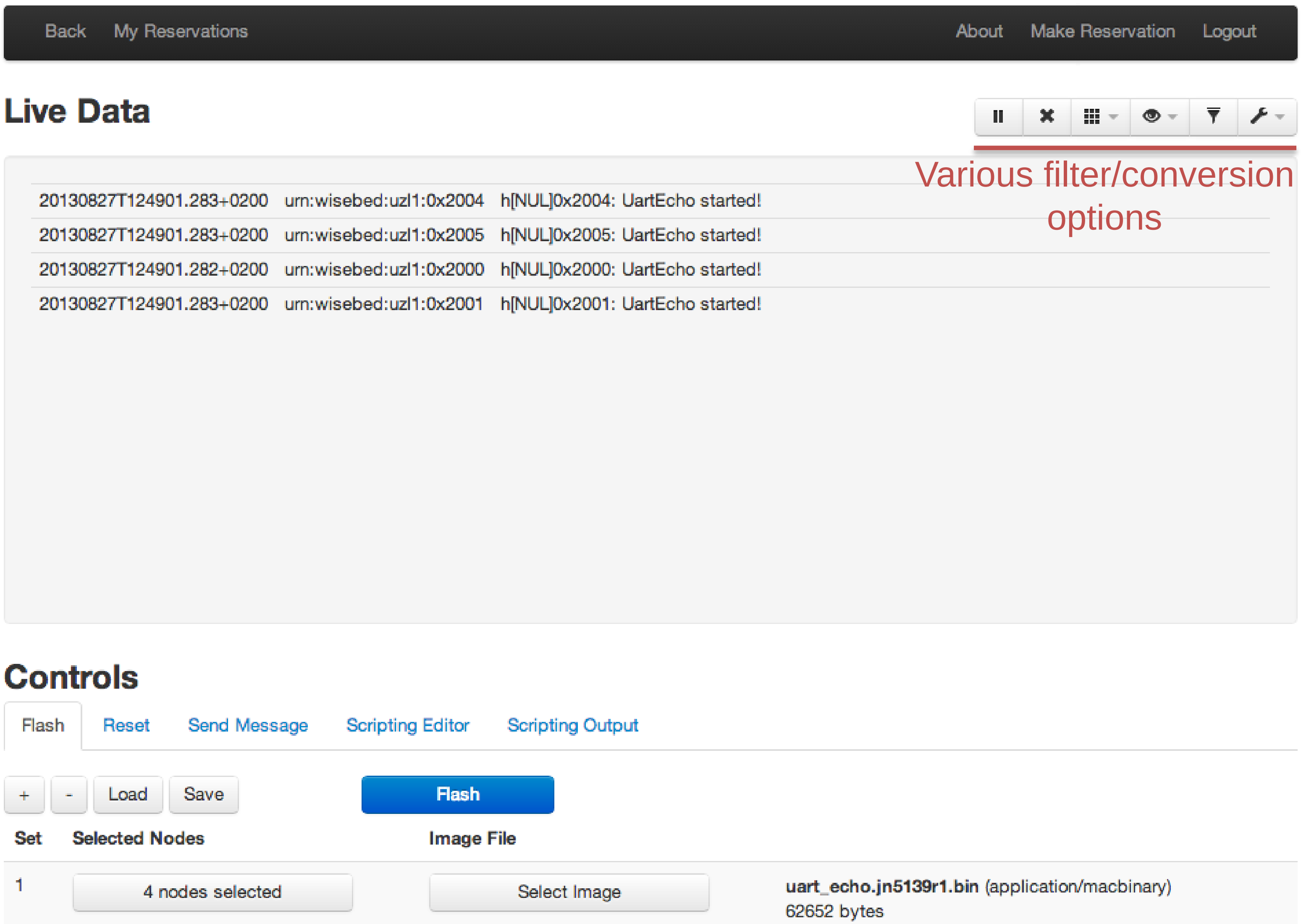


	Node URN	Type	Position	Sensors
☑	urn:wisebed:uzl1:0x2000	isense39	(31,6,1)	pir,acc
☑	urn:wisebed:uzl1:0x2001	isense39	(31,3,1)	pir,acc
☑	urn:wisebed:uzl1:0x2004	isense39	(27.5,2.5,1)	temperature,light
☑	urn:wisebed:uzl1:0x2005	isense39	(27.5,1,1)	pir,acc
☐	urn:wisebed:uzl1:0x2008	isense39	(25,3,1)	temperature,light
☐	urn:wisebed:uzl1:0x2009	isense39	(25,2,1)	pir,acc
☐	urn:wisebed:uzl1:0x200c	isense39	(17,0.5,0)	temperature,light
☐	urn:wisebed:uzl1:0x200d	isense39	(20.5,1.5,1)	temperature,light
☐	urn:wisebed:uzl1:0x2010	isense39	(14,1,1)	temperature,light
☐	urn:wisebed:uzl1:0x2011	isense39	(14,4,1)	pir,acc
☐	urn:wisebed:uzl1:0x2014	isense39	(13.5,1,1)	pir,acc
☐	urn:wisebed:uzl1:0x2015	isense39	(10.5,3,1)	temperature,light
☐	urn:wisebed:uzl1:0x2018	isense39	(8.5,7,2)	pir,acc

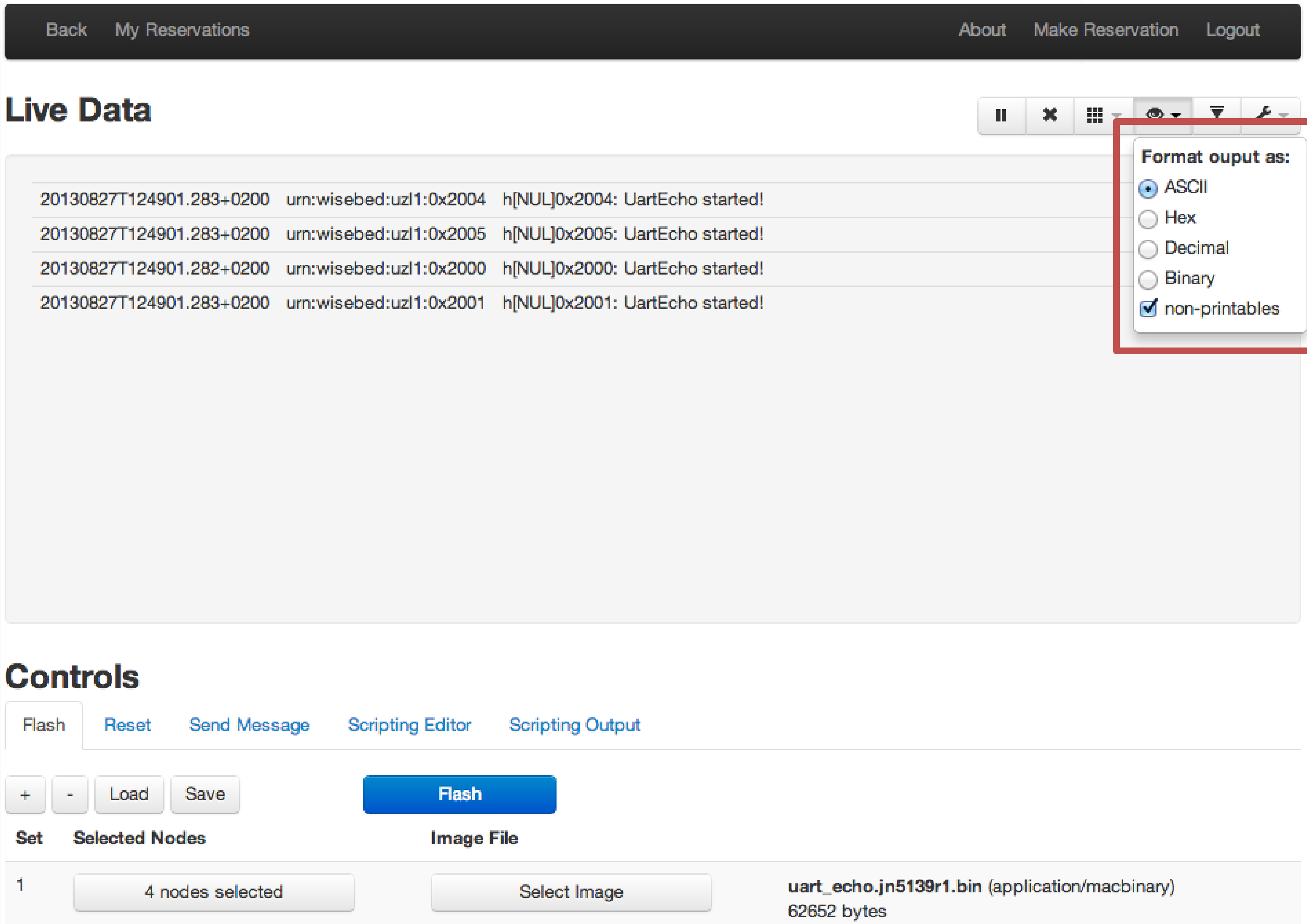
2.2 WiseGui – Select Reservation



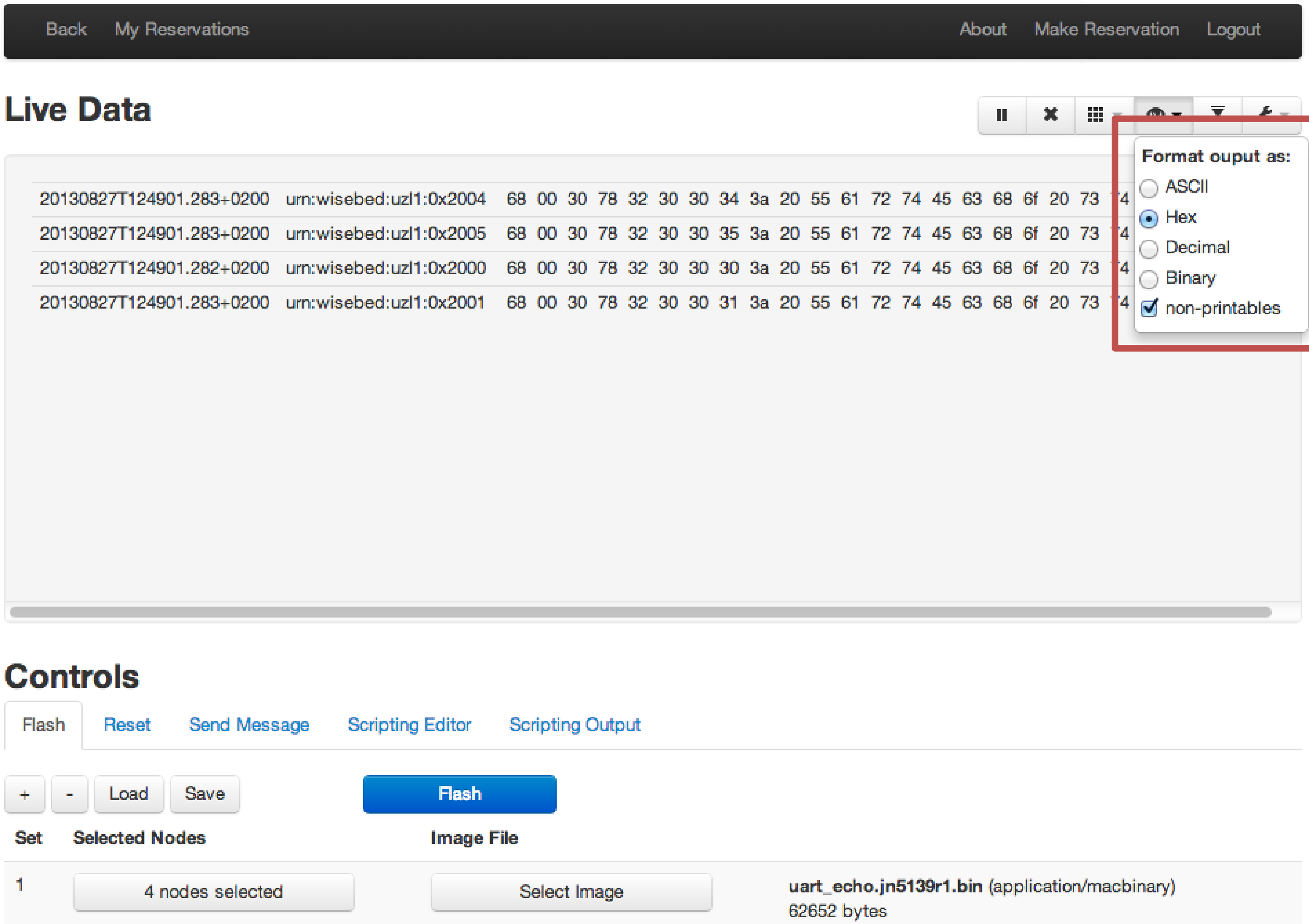
2.2 WiseGui – Live Data View



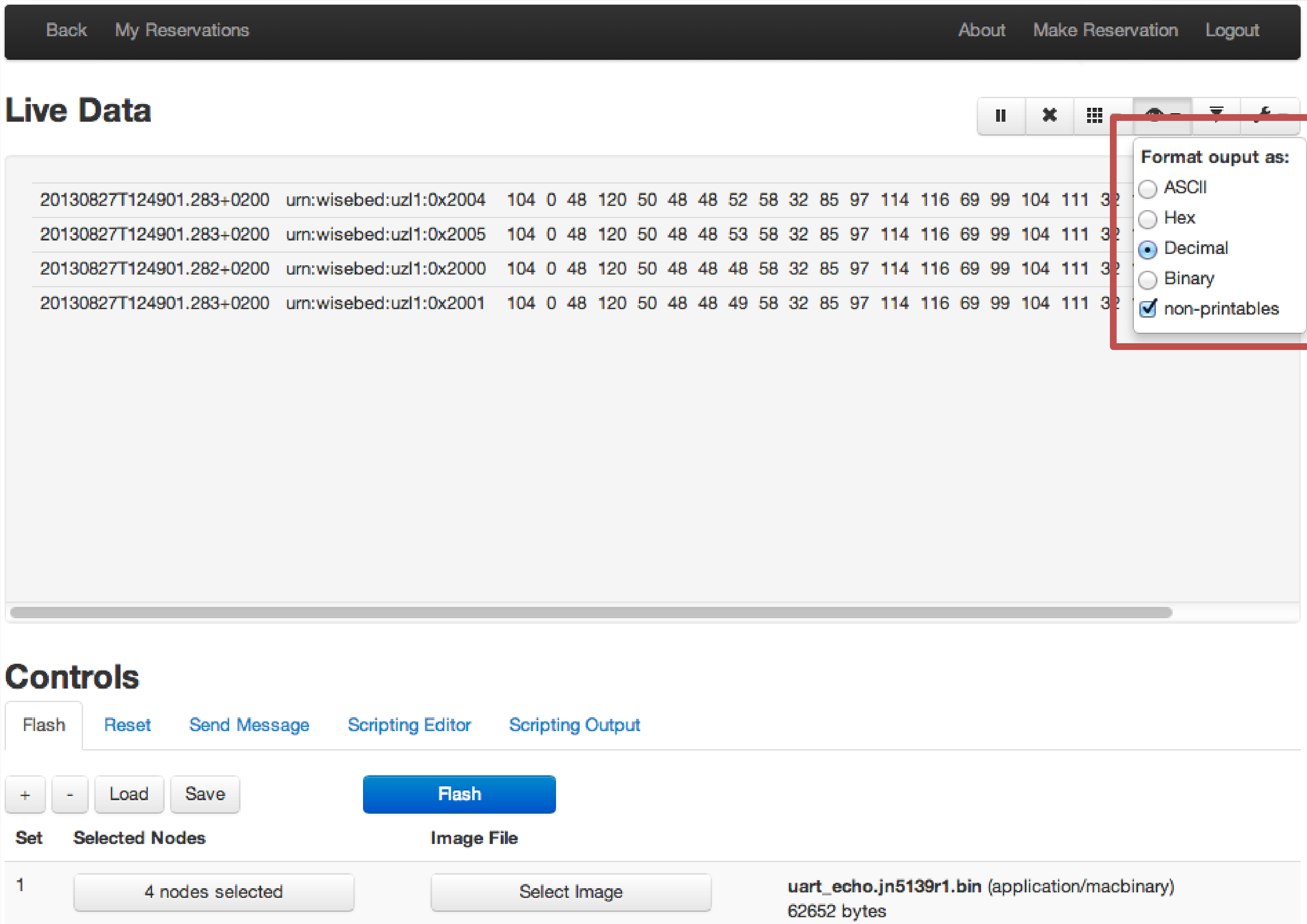
2.2 WiseGui – Live Data View



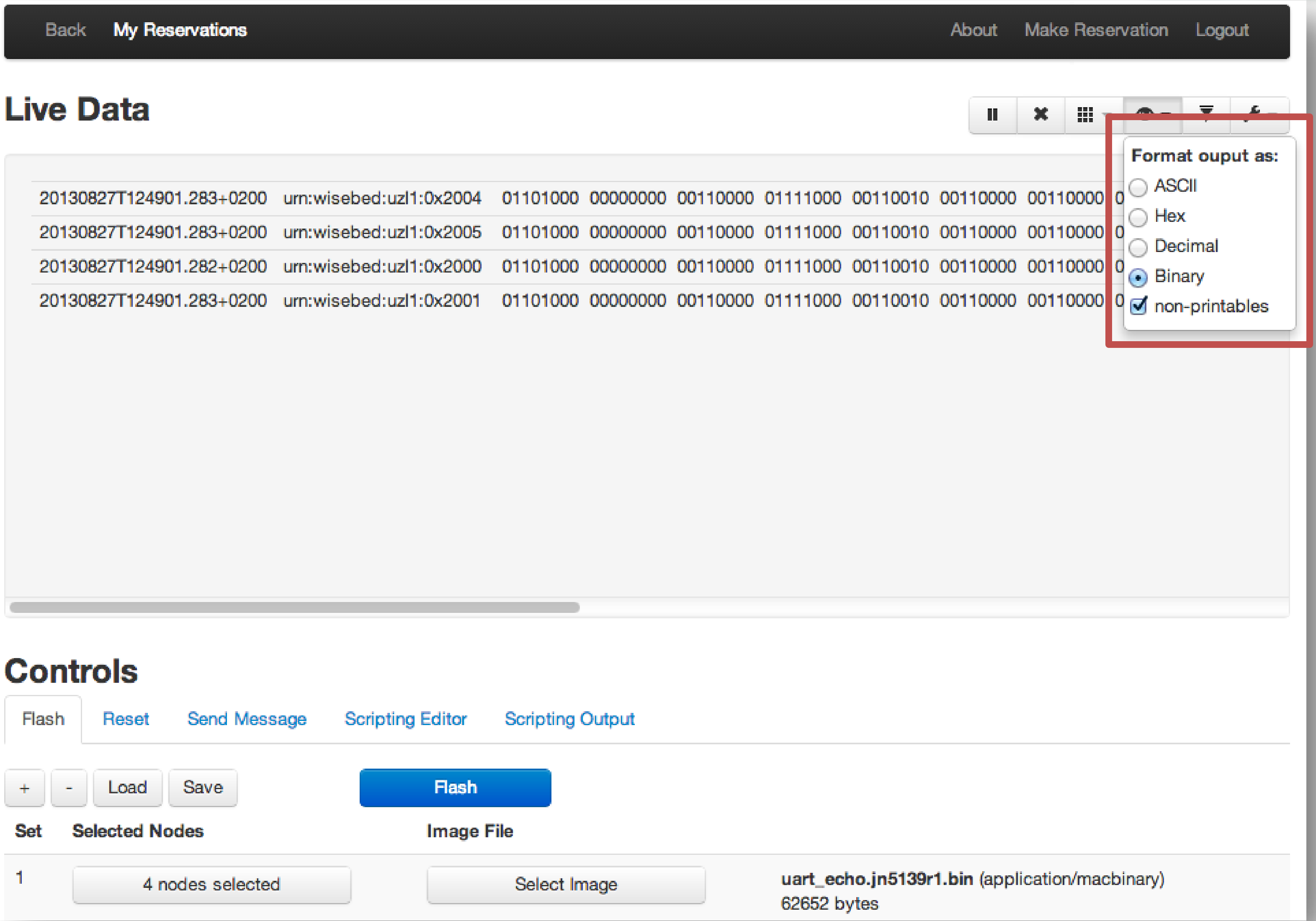
2.2 WiseGui – Live Data View



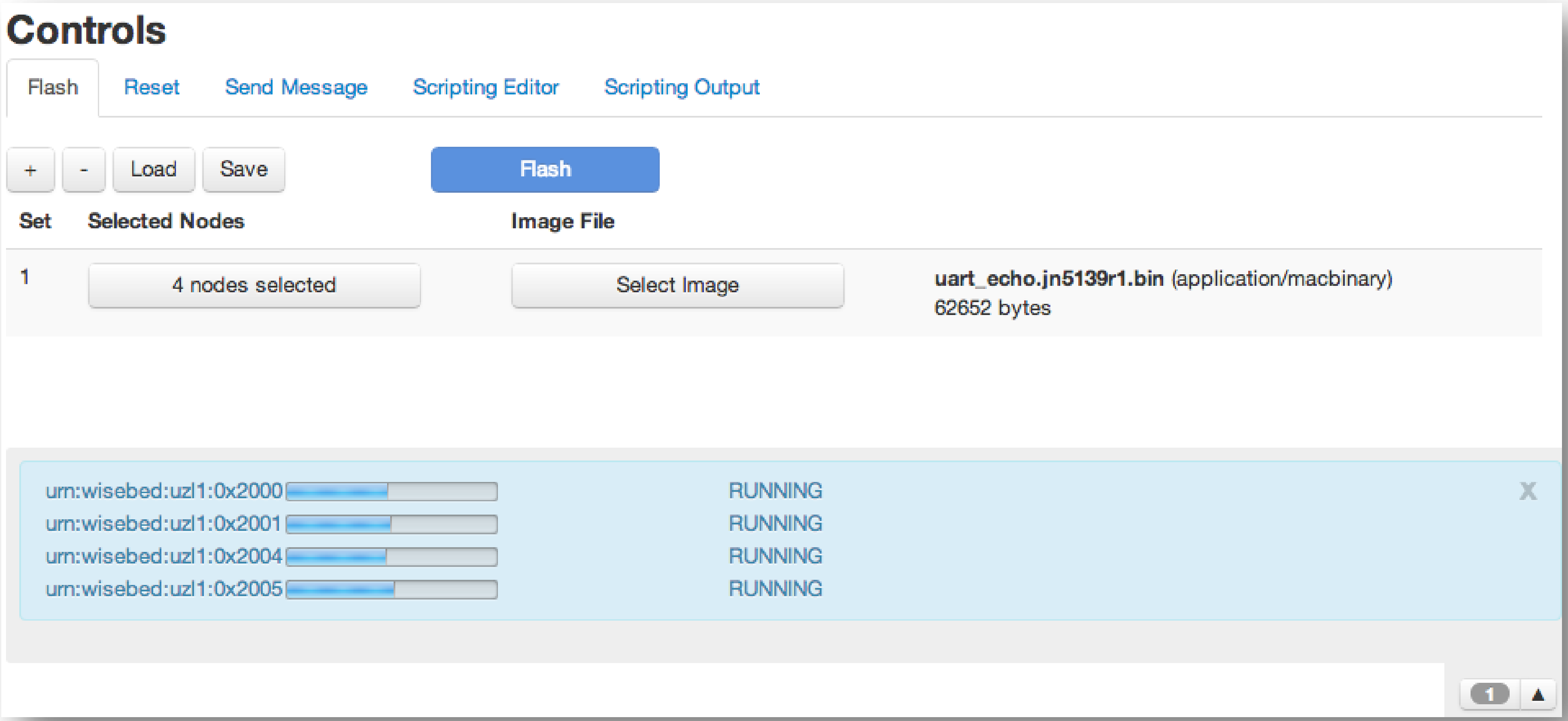
2.2 WiseGui – Live Data View



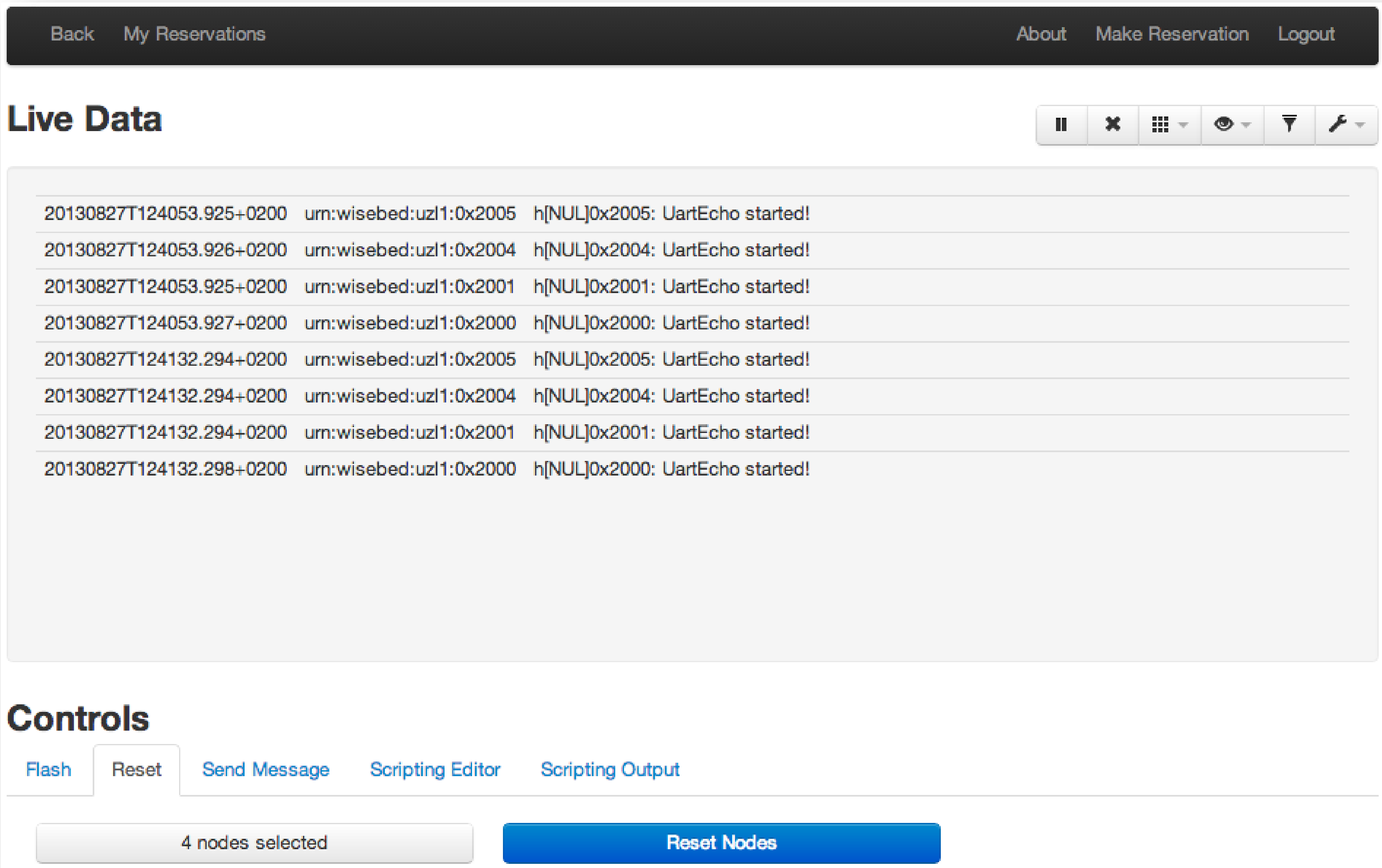
2.2 WiseGui – Live Data View



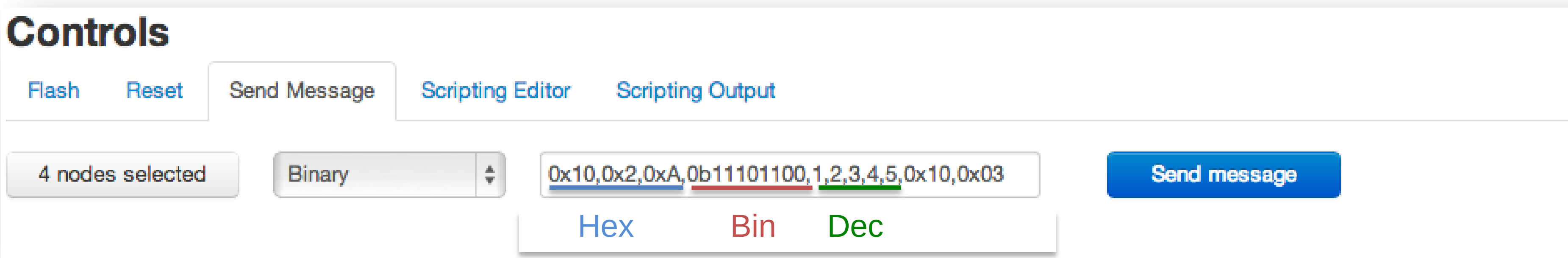
2.2 WiseGui – Flashing Nodes



2.2 WiseGui – Resetting Nodes



2.2 WiseGui – Sending Messages



2.2 WiseGui – Scripting Environment

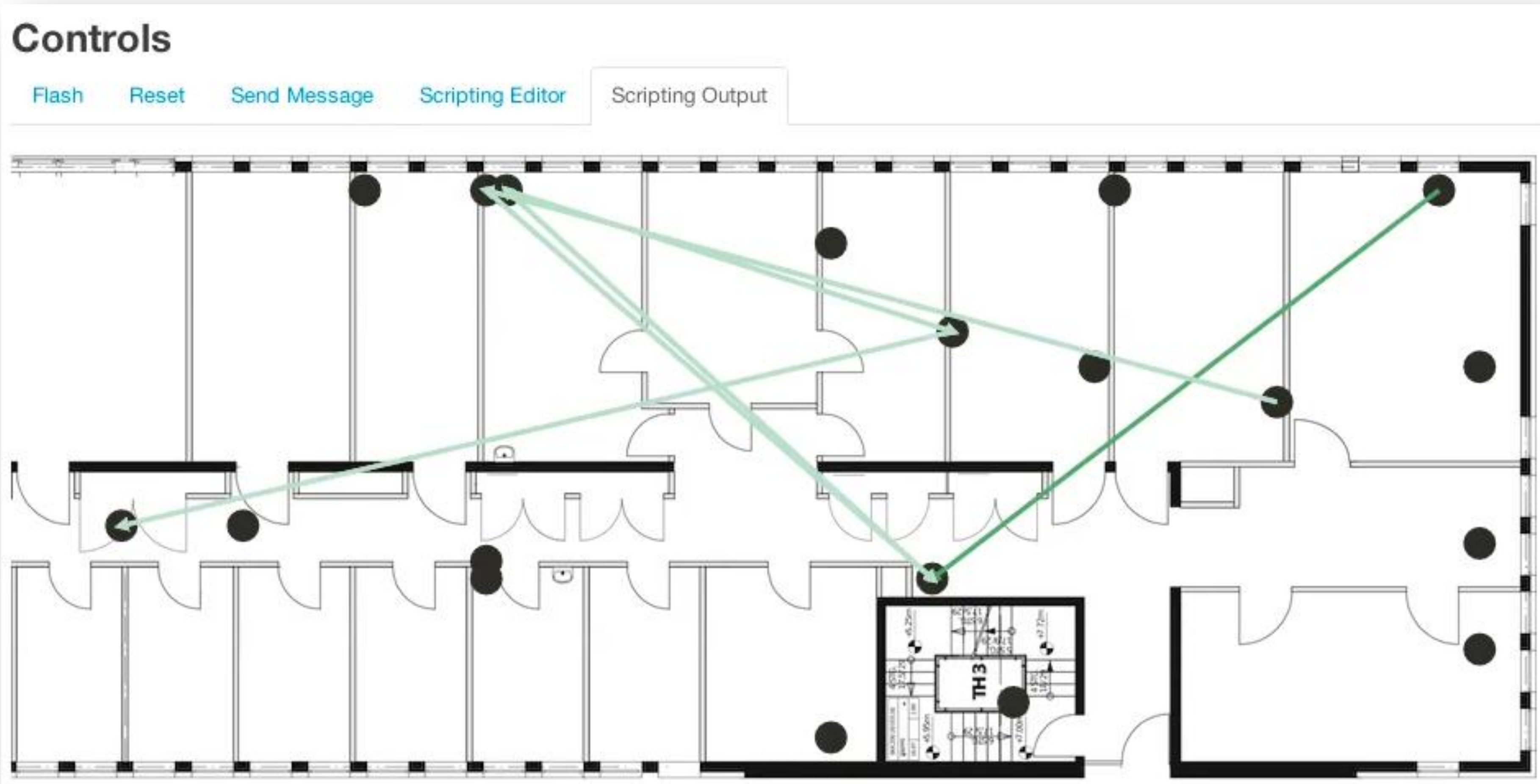
- JavaScript scripting environment for execution in browser
- Allows users to automate / script experiments in JavaScript without any software installation
- Program against REST API
  (using wisebed.js client library)
- Scripting environment pre-initialized with
  - WebSocket to sensor nodes serial ports, and
  - reservation data (reserved nodes, timespan, …)

2.2 WiseGui – Scripting Editor

2.2 WiseGui – Scripting Editor Demo

- Demo shows a scripted application
  - Periodically, sensor nodes ping other nodes
  - 6LoWPAN packets are dumped to serial port
  - Dump is forwarded to WiseGui scripting environment
  - Concurrent transmissions distinguished with label
  - Output: live visualization of packet trajectory in wireless network
- Libraries used: wisebed.js, jquery, d3.js

2.2 WiseGui – Scripting Editor Demo



2.2 WiseGui – Scripting Editor Demo

Demo Video

2.2 WiseGui - Summary

- Completely based on HTML5/JavaScript
- Runs on client side (Browser)
- WebSocket-based bi-directional communication with nodes
- Integrated scripting environment

- Uses REST API and wisebed.js
- Open Source https://github.com/wisebed/wisegui

Experimentation with SmartSantander

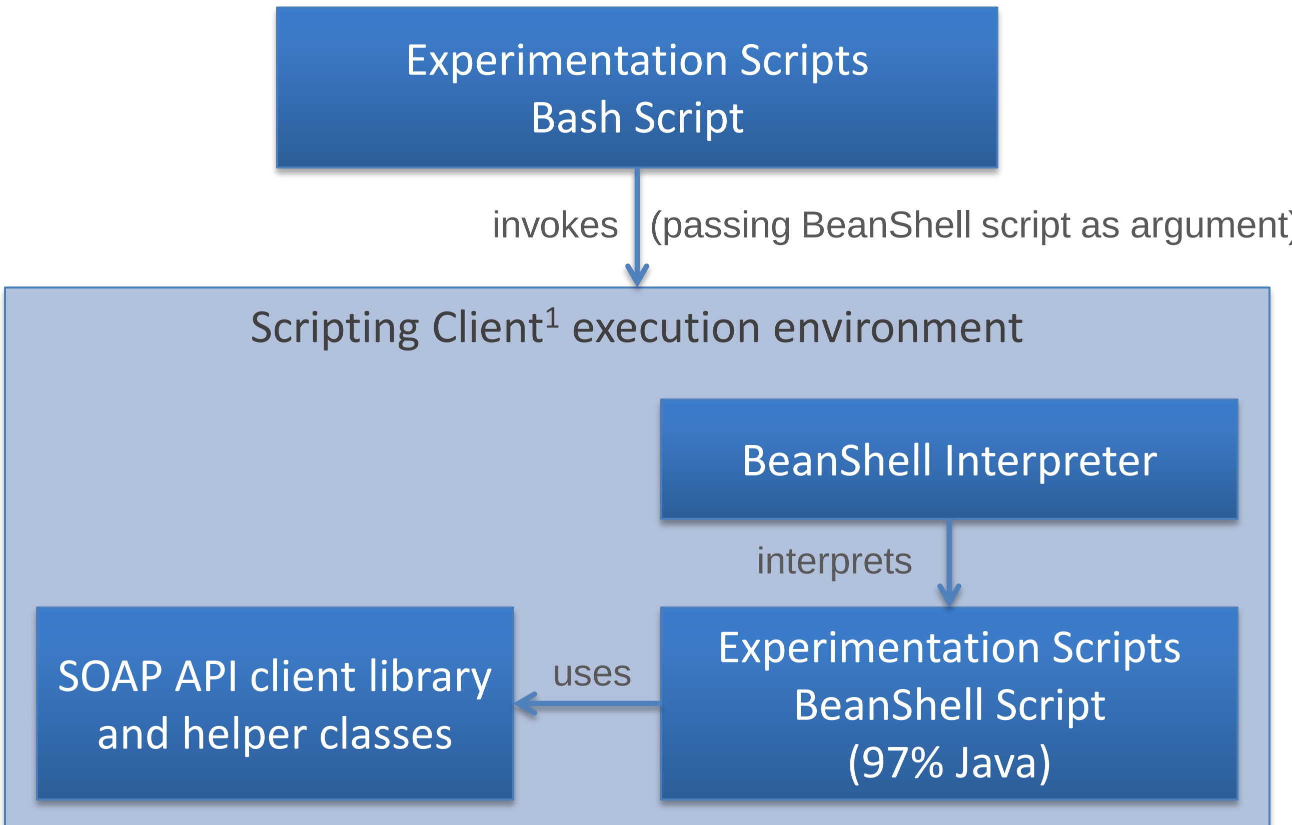
EXPERIMENTATION-SCRIPTS LIVE-DEMO (SCREENSHOT WALK-THROUGH)

2.3 Experimentation Scripts

- Set of command line scripts to execute, control and interact with your experiment
- Allows you to
  - Execute basic operations (flash, reset, …)
  - Automate your experiments (e.g., to explore parameter space)
  - Automatically repeat experiments
  - Programmatically analyze, convert and process output of nodes

- Download: https://github.com/wisebed/experimentation-scripts/
- Documentation: https://github.com/wisebed/experimentation-scripts/wiki

2.3 Experimentation Scripts



[1] https://github.com/wisebed/scripting-client [2] https://github.com/wisebed/api-wsdl [3] https://github.com/wisebed/api-java

2.3 Live Presentation

1. List node URNs of type „isense“

   wb-list-node-urns luebeck.properties csv isense39
   export NODES=…

2. Reserve nodes (20 minutes, starting from now)

   wb-reserve luebeck.properties 20 0 $NODES
   export RESERVATION=…

3. Check liveliness

   wb-are-nodes-alive luebeck.properties $RESERVATION

4. Flash nodes

   wb-flash luebeck.properties $RESERVATION binary-images/example_app.bin

5. Listen to node output

   wb-listen luebeck.properties $RESERVATION

6. Reset nodes

   wb-reset luebeck.properties $RESERVATION

2.3 Testbed Properties File

bimschas — bash — 103×31

```
bimschas@opium [~] $ cat luebeck.properties
testbed.snaa.endpointurl  = http://wisebed.itm.uni-luebeck.de:8890/snaa
testbed.rs.endpointurl    = http://wisebed.itm.uni-luebeck.de:8889/rs
testbed.sm.endpointurl    = http://wisebed.itm.uni-luebeck.de:8888/sessions
testbed.urnprefixes       = urn:wisebed:uzl1:
testbed.usernames         = bimschas@wisebed1.itm.uni-luebeck.de
testbed.passwords         =
testbed.protobuf.hostname = wisebed.itm.uni-luebeck.de
testbed.protobuf.port     = 8885
bimschas@opium [~] $
```

2.3 Printing Available Nodes

```
bimschas@opium [~] $ wb-list-node-urns $TB_LIVE csv isense39
urn:wisebed:uzl1:0x201c,urn:wisebed:uzl1:0x201d,urn:wisebed:uzl1:0x2015,urn:wisebed:uzl1:0x2014,urn:wis
ebed:uzl1:0x200d,urn:wisebed:uzl1:0x200c,urn:wisebed:uzl1:0x2004,urn:wisebed:uzl1:0x2005,urn:wisebed:uz
l1:0x2025,urn:wisebed:uzl1:0x2024,urn:wisebed:uzl1:0x2030,urn:wisebed:uzl1:0x2031,urn:wisebed:uzl1:0x20
18,urn:wisebed:uzl1:0x2035,urn:wisebed:uzl1:0x2019,urn:wisebed:uzl1:0x2034,urn:wisebed:uzl1:0x202d,urn:
wisebed:uzl1:0x202c,urn:wisebed:uzl1:0x2029,urn:wisebed:uzl1:0x2028,urn:wisebed:uzl1:0x2021,urn:wisebed
:uzl1:0x2020,urn:wisebed:uzl1:0x2038,urn:wisebed:uzl1:0x2039,urn:wisebed:uzl1:0x203c,urn:wisebed:uzl1:0
x203d,urn:wisebed:uzl1:0x2044,urn:wisebed:uzl1:0x2045,urn:wisebed:uzl1:0x2041,urn:wisebed:uzl1:0x2040,u
rn:wisebed:uzl1:0x2011,urn:wisebed:uzl1:0x2010,urn:wisebed:uzl1:0x2009,urn:wisebed:uzl1:0x2008,urn:wise
bed:uzl1:0x2000,urn:wisebed:uzl1:0x2001
bimschas@opium [~] $
```

filter by node type

output mode

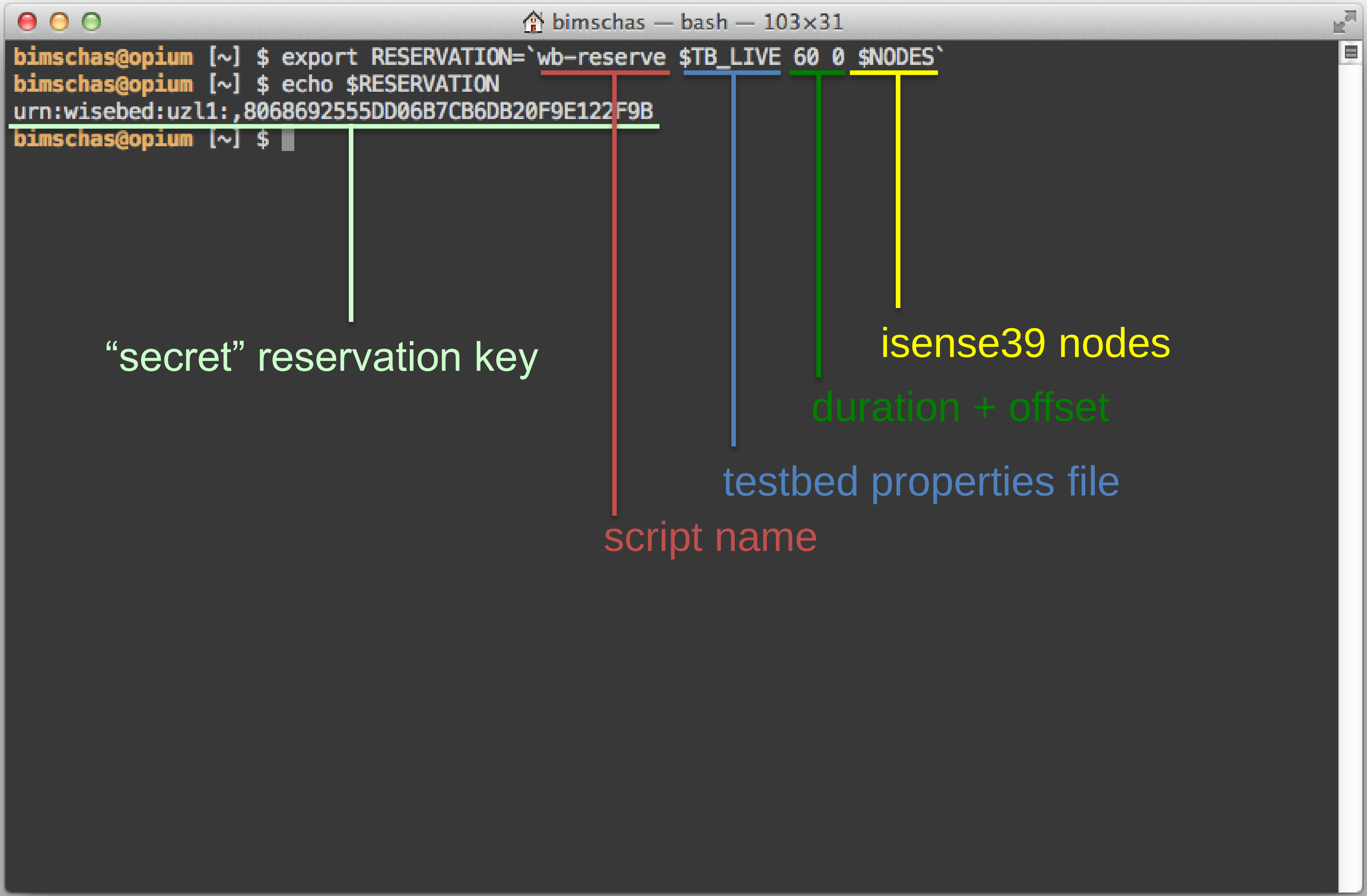
testbed properties file

script name

2.3 "Pro" Tip: Use Environment Variables

```
bimschas — bash — 103×31
bimschas@opium [~] $ export TB_LIVE=luebeck.properties
bimschas@opium [~] $ export NODES=`wb-list-node-urns $TB_LIVE csv isense39`
bimschas@opium [~] $ echo $NODES
urn:wisebed:uzl1:0x201c,urn:wisebed:uzl1:0x201d,urn:wisebed:uzl1:0x2015,urn:wisebed:uzl1:0x2014,urn:wis
ebed:uzl1:0x200d,urn:wisebed:uzl1:0x200c,urn:wisebed:uzl1:0x2004,urn:wisebed:uzl1:0x2005,urn:wisebed:uz
l1:0x2025,urn:wisebed:uzl1:0x2024,urn:wisebed:uzl1:0x2030,urn:wisebed:uzl1:0x2031,urn:wisebed:uzl1:0x20
18,urn:wisebed:uzl1:0x2035,urn:wisebed:uzl1:0x2019,urn:wisebed:uzl1:0x2034,urn:wisebed:uzl1:0x202d,urn:
wisebed:uzl1:0x202c,urn:wisebed:uzl1:0x2029,urn:wisebed:uzl1:0x2028,urn:wisebed:uzl1:0x2021,urn:wisebed
:uzl1:0x2020,urn:wisebed:uzl1:0x2038,urn:wisebed:uzl1:0x2039,urn:wisebed:uzl1:0x203c,urn:wisebed:uzl1:0
x203d,urn:wisebed:uzl1:0x2044,urn:wisebed:uzl1:0x2045,urn:wisebed:uzl1:0x2041,urn:wisebed:uzl1:0x2040,u
rn:wisebed:uzl1:0x2011,urn:wisebed:uzl1:0x2010,urn:wisebed:uzl1:0x2009,urn:wisebed:uzl1:0x2008,urn:wise
bed:uzl1:0x2000,urn:wisebed:uzl1:0x2001
bimschas@opium [~] $
```

2.3 Reserving Nodes



2.3 Checking Liveliness

```
bimschas — bash — 103×31
bimschas@opium [~] $ wb-are-nodes-alive $TB_LIVE $RESERVATION
urn:wisebed:uzl1:0x2010 | true |
urn:wisebed:uzl1:0x2044 | true |
urn:wisebed:uzl1:0x2045 | true |
urn:wisebed:uzl1:0x2019 | true |
urn:wisebed:uzl1:0x2041 | true |
urn:wisebed:uzl1:0x2040 | true |
urn:wisebed:uzl1:0x2014 | true |
urn:wisebed:uzl1:0x202d | true |
urn:wisebed:uzl1:0x202c | true |
urn:wisebed:uzl1:0x2011 | true |
urn:wisebed:uzl1:0x2018 | true |
urn:wisebed:uzl1:0x2015 | false | Failed checking if node is alive. Reason: Device is not connected.
urn:wisebed:uzl1:0x2020 | true |
urn:wisebed:uzl1:0x2021 | true |
urn:wisebed:uzl1:0x2034 | true |
urn:wisebed:uzl1:0x2035 | false | Failed checking if node is alive. Reason: Device is not connected.
urn:wisebed:uzl1:0x200c | true |
urn:wisebed:uzl1:0x200d | true |
urn:wisebed:uzl1:0x2038 | true |
urn:wisebed:uzl1:0x2039 | true |
urn:wisebed:uzl1:0x2030 | true |
urn:wisebed:uzl1:0x2009 | true |
urn:wisebed:uzl1:0x2008 | true |
urn:wisebed:uzl1:0x2031 | true |
urn:wisebed:uzl1:0x2005 | true |
urn:wisebed:uzl1:0x2004 | true |
urn:wisebed:uzl1:0x2025 | true |
urn:wisebed:uzl1:0x201d | false | Failed checking if node is alive. Reason: Device is not connected.
urn:wisebed:uzl1:0x201c | false | Failed checking if node is alive. Reason: Device is not connected.
urn:wisebed:uzl1:0x2024 | true |
```

"secret" reservation key

testbed properties file

script name

2.3 Listening to Node Outputs

```
bimschas — java — 103×31
bimschas@opium [~] $ wb-listen $TB_LIVE $RESERVATION
2013-08-27T15:47:14.200+02:00 | urn:wisebed:uzl1:0x2010 | h[NUL]broadcasting message at 0x2010 [CR][LF]
 | 0x68 0x0 0x62 0x72 0x6f 0x61 0x64 0x63 0x61 0x73 0x74 0x69 0x6e 0x67 0x20 0x6d 0x65 0x73 0x73 0x61 0
x67 0x65 0x20 0x61 0x74 0x20 0x30 0x78 0x32 0x30 0x31 0x30 0x20 0xd 0xa
2013-08-27T15:47:19.201+02:00 | urn:wisebed:uzl1:0x2010 | h[NUL]broadcasting message at 0x2010 [CR][LF]
 | 0x68 0x0 0x62 0x72 0x6f 0x61 0x64 0x63 0x61 0x73 0x74 0x69 0x6e 0x67 0x20 0x6d 0x65 0x73 0x73 0x61 0
x67 0x65 0x20 0x61 0x74 0x20 0x30 0x78 0x32 0x30 0x31 0x30 0x20 0xd 0xa
2013-08-27T15:47:22.521+02:00 | urn:wisebed:uzl1:0x200c | h[NUL]TestUart::enable()[CR][LF] | 0x68 0x0 0
x54 0x65 0x73 0x74 0x55 0x61 0x72 0x74 0x3a 0x3a 0x65 0x6e 0x61 0x62 0x6c 0x65 0x28 0x29 0xd 0xa
2013-08-27T15:47:22.520+02:00 | urn:wisebed:uzl1:0x2034 | h[NUL]Hello World from Example Application![C
R][LF] | 0x68 0x0 0x48 0x65 0x6c 0x6c 0x6f 0x20 0x57 0x6f 0x72 0x6c 0x64 0x20 0x66 0x72 0x6f 0x6d 0x20
0x45 0x78 0x61 0x6d 0x70 0x6c 0x65 0x20 0x41 0x70 0x70 0x6c 0x69 0x63 0x61 0x74 0x69 0x6f 0x6e 0x21 0xd
 0xa
2013-08-27T15:47:22.522+02:00 | urn:wisebed:uzl1:0x200c | i123456789:[LF] | 0x69 0x31 0x32 0x33 0x34 0x
35 0x36 0x37 0x38 0x39 0x3a 0xa
2013-08-27T15:47:22.523+02:00 | urn:wisebed:uzl1:0x2005 | h[NUL]0x2005: UartEcho started! | 0x68 0x0 0x
30 0x78 0x32 0x30 0x30 0x35 0x3a 0x20 0x55 0x61 0x72 0x74 0x45 0x63 0x68 0x6f 0x20 0x73 0x74 0x61 0x72
0x74 0x65 0x64 0x21
2013-08-27T15:47:22.526+02:00 | urn:wisebed:uzl1:0x2010 | h[NUL]Remote Application booting![CR][LF] | 0
x68 0x0 0x52 0x65 0x6d 0x6f 0x74 0x65 0x20 0x41 0x70 0x70 0x6c 0x69 0x63 0x61 0x74 0x69 0x6f 0x6e 0x20
0x62 0x6f 0x6f 0x74 0x69 0x6e 0x67 0x21 0xd 0xa
2013-08-27T15:47:22.526+02:00 | urn:wisebed:uzl1:0x2010 | h[NUL]Uart: Enable communication | 0x68 0x0 0
x55 0x61 0x72 0x74 0x3a 0x20 0x45 0x6e 0x61 0x62 0x6c 0x65 0x20 0x63 0x6f 0x6d 0x6d 0x75 0x6e 0x69 0x63
 0x61 0x74 0x69 0x6f 0x6e
2013-08-27T15:47:22.527+02:00 | urn:wisebed:uzl1:0x2010 | h[NUL]enable radio! | 0x68 0x0 0x65 0x6e 0x61
 0x62 0x6c 0x65 0x20 0x72 0x61 0x64 0x69 0x6f 0x21
2013-08-27T15:47:22.523+02:00 | urn:wisebed:uzl1:0x2001 | h[NUL]0x2001: UartEcho started! | 0x68 0x0 0x
30 0x78 0x32 0x30 0x30 0x31 0x3a 0x20 0x55 0x61 0x72 0x74 0x45 0x63 0x68 0x6f 0x20 0x73 0x74 0x61 0x72
0x74 0x65 0x64 0x21
2013-08-27T15:47:22.527+02:00 | urn:wisebed:uzl1:0x2010 | h[NUL]randomize! | 0x68 0x0 0x72 0x61 0x6e 0x
64 0x6f 0x6d 0x69 0x7a 0x65 0x21
```

2.3 Resetting Nodes

```
bimschas — bash — 103×31
bimschas@opium [~] $ wb-reset $TB_LIVE $RESERVATION
urn:wisebed:uzl1:0x2010 | true |
urn:wisebed:uzl1:0x2044 | true |
urn:wisebed:uzl1:0x2045 | true |
urn:wisebed:uzl1:0x2019 | true |
urn:wisebed:uzl1:0x2041 | true |
urn:wisebed:uzl1:0x2040 | true |
urn:wisebed:uzl1:0x2014 | true |
urn:wisebed:uzl1:0x202d | true |
urn:wisebed:uzl1:0x202c | true |
urn:wisebed:uzl1:0x2011 | true |
urn:wisebed:uzl1:0x2018 | true |
urn:wisebed:uzl1:0x2015 | false | Failed resetting node. Reason: Device is not connected.
urn:wisebed:uzl1:0x2020 | true |
urn:wisebed:uzl1:0x2021 | true |
urn:wisebed:uzl1:0x2034 | true |
urn:wisebed:uzl1:0x2035 | false | Failed resetting node. Reason: Device is not connected.
urn:wisebed:uzl1:0x200c | true |
urn:wisebed:uzl1:0x200d | true |
urn:wisebed:uzl1:0x2038 | true |
urn:wisebed:uzl1:0x2039 | true |
urn:wisebed:uzl1:0x2030 | true |
urn:wisebed:uzl1:0x2009 | true |
urn:wisebed:uzl1:0x2008 | true |
urn:wisebed:uzl1:0x2031 | true |
urn:wisebed:uzl1:0x2005 | true |
urn:wisebed:uzl1:0x2004 | true |
urn:wisebed:uzl1:0x2025 | true |
urn:wisebed:uzl1:0x201d | false | Failed resetting node. Reason: Device is not connected.
urn:wisebed:uzl1:0x201c | false | Failed resetting node. Reason: Device is not connected.
urn:wisebed:uzl1:0x2024 | true |
```

2.3 Flashing Nodes

bimschas — bash — 103×31

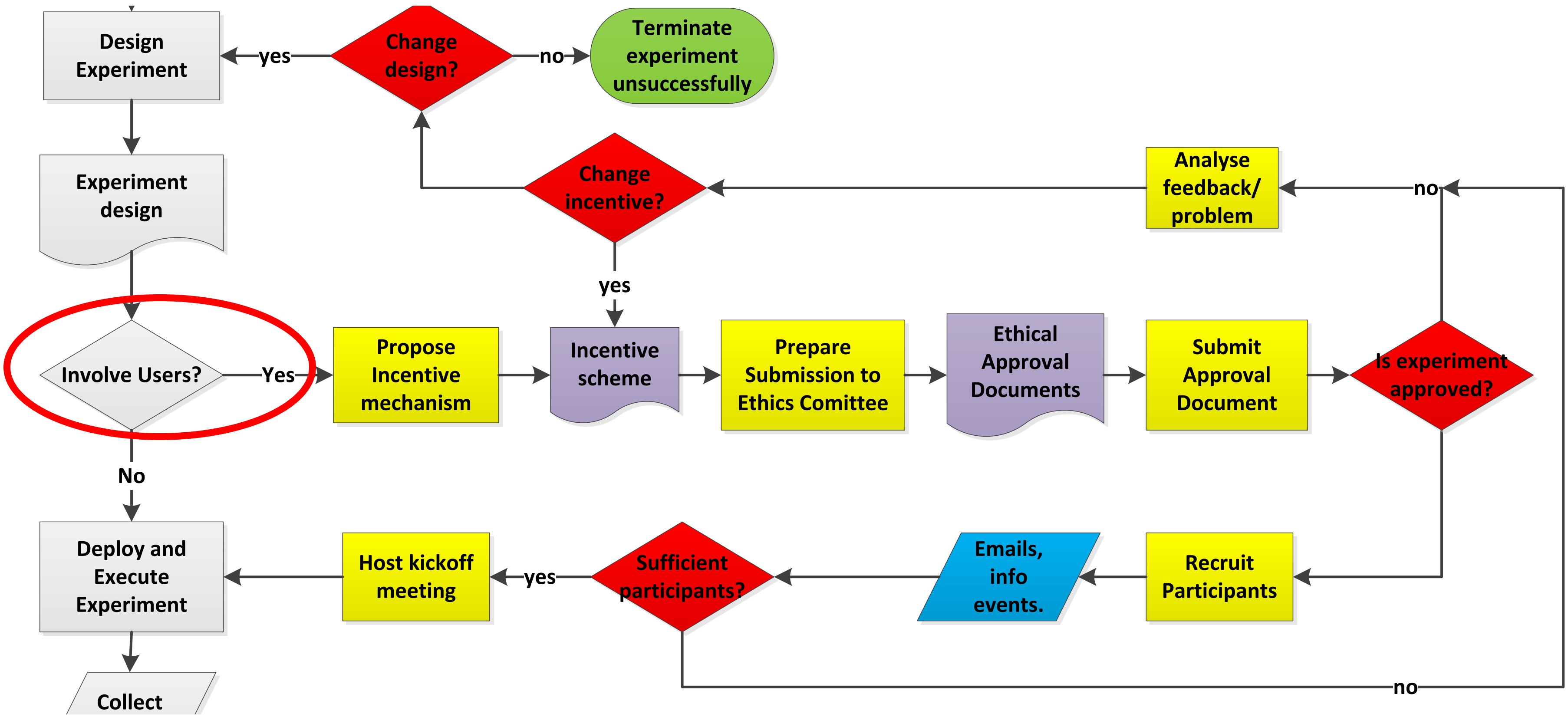
```
bimschas@opium [~] $ wb-flash $TB_LIVE $RESERVATION ~/tb/bin/uart_echo.jn5139r1.bin
urn:wisebed:uzl1:0x2010 | true |
urn:wisebed:uzl1:0x2044 | true |
urn:wisebed:uzl1:0x2045 | true |
urn:wisebed:uzl1:0x2019 | true |
urn:wisebed:uzl1:0x2041 | true |
urn:wisebed:uzl1:0x2040 | true |
urn:wisebed:uzl1:0x2014 | true |
urn:wisebed:uzl1:0x202d | true |
urn:wisebed:uzl1:0x202c | true |
urn:wisebed:uzl1:0x2011 | true |
urn:wisebed:uzl1:0x2018 | true |
urn:wisebed:uzl1:0x2015 | false | Failed flashing node. Reason: Node is not connected.
urn:wisebed:uzl1:0x2020 | true |
urn:wisebed:uzl1:0x2021 | true |
urn:wisebed:uzl1:0x2034 | true |
urn:wisebed:uzl1:0x2035 | false | Failed flashing node. Reason: Node is not connected.
urn:wisebed:uzl1:0x200c | true |
urn:wisebed:uzl1:0x200d | true |
urn:wisebed:uzl1:0x2038 | true |
urn:wisebed:uzl1:0x2039 | true |
urn:wisebed:uzl1:0x2009 | true |
urn:wisebed:uzl1:0x2030 | true |
urn:wisebed:uzl1:0x2008 | true |
urn:wisebed:uzl1:0x2031 | true |
urn:wisebed:uzl1:0x2005 | true |
urn:wisebed:uzl1:0x2004 | true |
urn:wisebed:uzl1:0x2025 | true |
urn:wisebed:uzl1:0x201d | false | Failed flashing node. Reason: Node is not connected.
urn:wisebed:uzl1:0x201c | false | Failed flashing node. Reason: Node is not connected.
urn:wisebed:uzl1:0x2024 | true |
```

image file

Experimentation with SmartSantander

USER CENTRIC EXPERIMENTATION

User centric experimentation



Users in experimentation

- Understand the role of users in an experiment
  - Profile of users
  - Nature and level of participation
  - Level of inconvenience
  - Level of risk, e.g. exposure

- Understand the impact of experiment to its environment
  - Resources, users **part and not part** of an experiment, surroundings

- Dimensioning of the experiment
  - Number of users
  - Duration of experiment
  - Nature of data captured

Ensuring user participation

- Make sure incentive motivates honest participation in entire experiment not just registration
  - Self-drive and curiosity better than external reasons
- Finding supporting incentives is difficult
  - What will motivate your users to remain part of an experiment?
  - The greater the inconvenience or risk the more likely incentives are needed
  - Beware of “buying” users -> ethics
- Incentives may be manifold and not just material or financial

Criteria used by EC research

Informed Consent
Does the proposal involve children?
Does the proposal involve patients?
Does the proposal involve persons not able to give consent?
Does the proposal involve adult healthy volunteers?
Biological research
…..
Privacy
Does the proposal involve processing of genetic information or personal data (e.g. health, sexual lifestyle, ethnicity, political opinion, religious or philosophical conviction)
Does the proposal involve tracking the location or observation of people without their knowledge?
Research on Animals
….
Research Involving Third Countries
…
Dual Use
…
ICT Implants
…

Ethics in experiments (1/2)

- Concerns related to health and safety (physiological and mental)
  - Procedures involving any risk to a participant's health
  - Research where the safety of the researcher may be in question
  - Surveys, questionnaires and any research, the nature of which might be offensive, distressing or deeply personal for the particular target group

- Concerns related to specific groups of participants or experimenters for research
  - using **undergraduate students** as participants
  - using **children under the age of 16** or those over 16 who are unable to give informed consent as participants
  - **using prisoners or young offenders**
  - carried out by **persons unconnected with the University**, but wishing to use staff and/or students as participants

Ethics in experiments (2/2)

- Other concerns for proposed research
  - involving **access to records of personal or sensitive confidential information**, concerning identifiable individuals
  - requiring participants to take part in the study **without their knowledge and consent** at the time
  - **involving financial payments or payments in kind** to participants
  - investigating existing working or professional practices at the researcher's own place of work (including staff surveys)
  - involving the donation of bodily material, organs and the recently deceased

Ethics committee

- Aims to protect the ethical integrity of research
- Composed of experienced and independent experts/advisors from different disciplines
- Identifies ethical issues that could arise from proposed research
- Provides feedback in how research approach should be shaped to comply to ethical values

Ethics committee review

- Start well in time with the planning
  - A review cycle can take easily up to 4 weeks
  - Allow for at least one refinement cycle

- Be prepared for a lot of paper work
  - Cover letter
  - Coversheet
  - Protocol
  - Annex with all docs used for intended communications, interviews, informed consent (5-10 documents)

- Use a language that is easy to understand by non-expert in your field

Recruitment

- Try to capture the attention and interest
- Use methods and exact communication as specified in the ethics committee application
  - You may be subject to audits
- Diversity of methods available
  - Email announcements, social media
  - Information pack
  - Information event
  - Kick-off event
- Registration and informed consent

Site preparation

- Verify validity of environment around users
  - External factors that can have influences on experiment
  - Experiment could have influence on non-experimenters
- Manage instrumentation carefully to minimise disruption
  - User environment with sensors or interaction devices beyond testbed
  - User devices with experimentation code
- Verify experimentation setup beforehand
  - Errors are more difficult to fix during experiments

Kick-off

- Expect not all to attend the kick-off
  - Select accessible time slot /location for such event to maximise attendance
  - Provide alternatives (e.g. one-to-ones)

- Communicate clearly your expectations

- Provide all adequate training necessary for expected level of participation

- Respect the hosting organisation
  - Keep time loss to minimum

Humans during experiments

- Expect the unexpected
  - Lack of interest and engagement
  - Forgetfulness
  - Maliciousness and deliberate tempering
  - Unavailability (Holiday, business trips, sickness …)
- Design for detecting unexpected situations to preserve quality of your data
  - Post processing errors is more difficult without true baseline or reference
- Manage risks adequately
  - Contingency plans, insurance cover

Other considerations

- Provide adequate interaction channels during experiments
- Be reactive and helpful – remember they are doing you a favour
  - Minimise loss of data
  - Avoid loss of moral/motivation
- Anonymity requirements complicate matters
- Interviews
  - Pre-experiment, during experiment, post-experiment

SEVENTH FRAMEWORK PROGRAMME

SMART SANTANDER

Part 3- Santander Waspmote Programming

Learning Goals

- After this tutorial you will…
  - Know how to write a basic application for SmartSantander Waspmote devices
  - Know which functionalities are available at the nodes
  - … be ready for experimenting ☺

Learning Goals

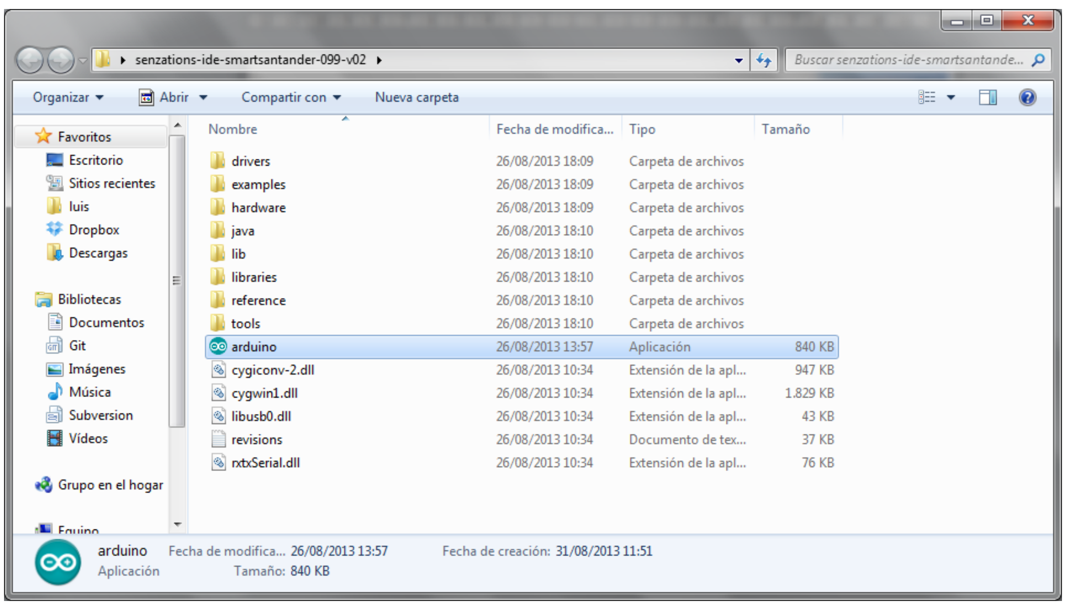
- After this tutorial you will not…
  – Immediately be able to try your applications

Outline

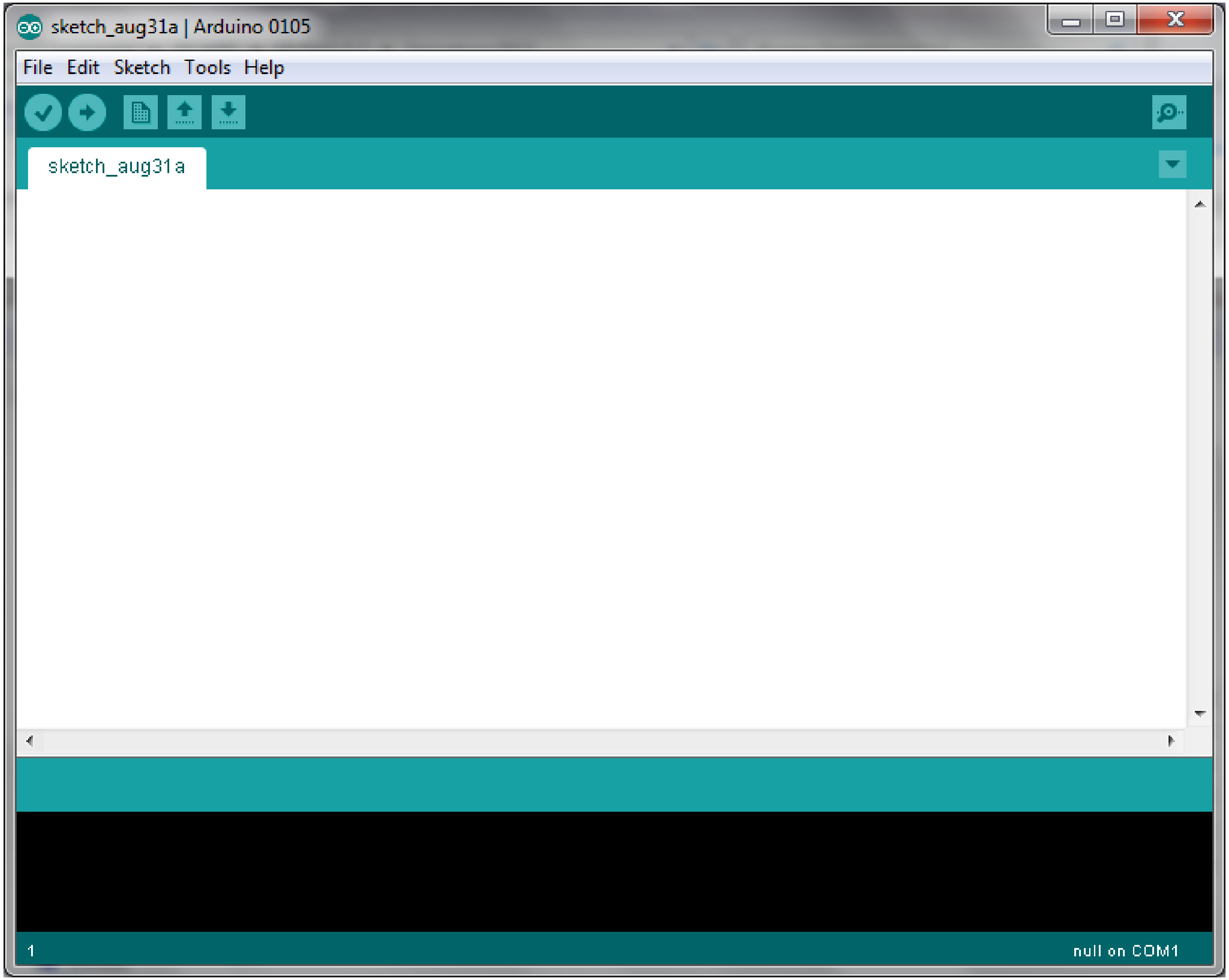
- Setting up your environment
- SmartSantander Waspmote devices
- Programming API
- Sample program
- Conclusions

Setting up your environment

- Download and install the IDE
  - Software can be downloaded from the link (https://www.dropbox.com/s/1w2ifiig97ihlvb/senzations-ide-smartsantander-099-v02.rar)
- Once the software has been downloaded and unzipped, the IDE can be executed by clicking on arduino application

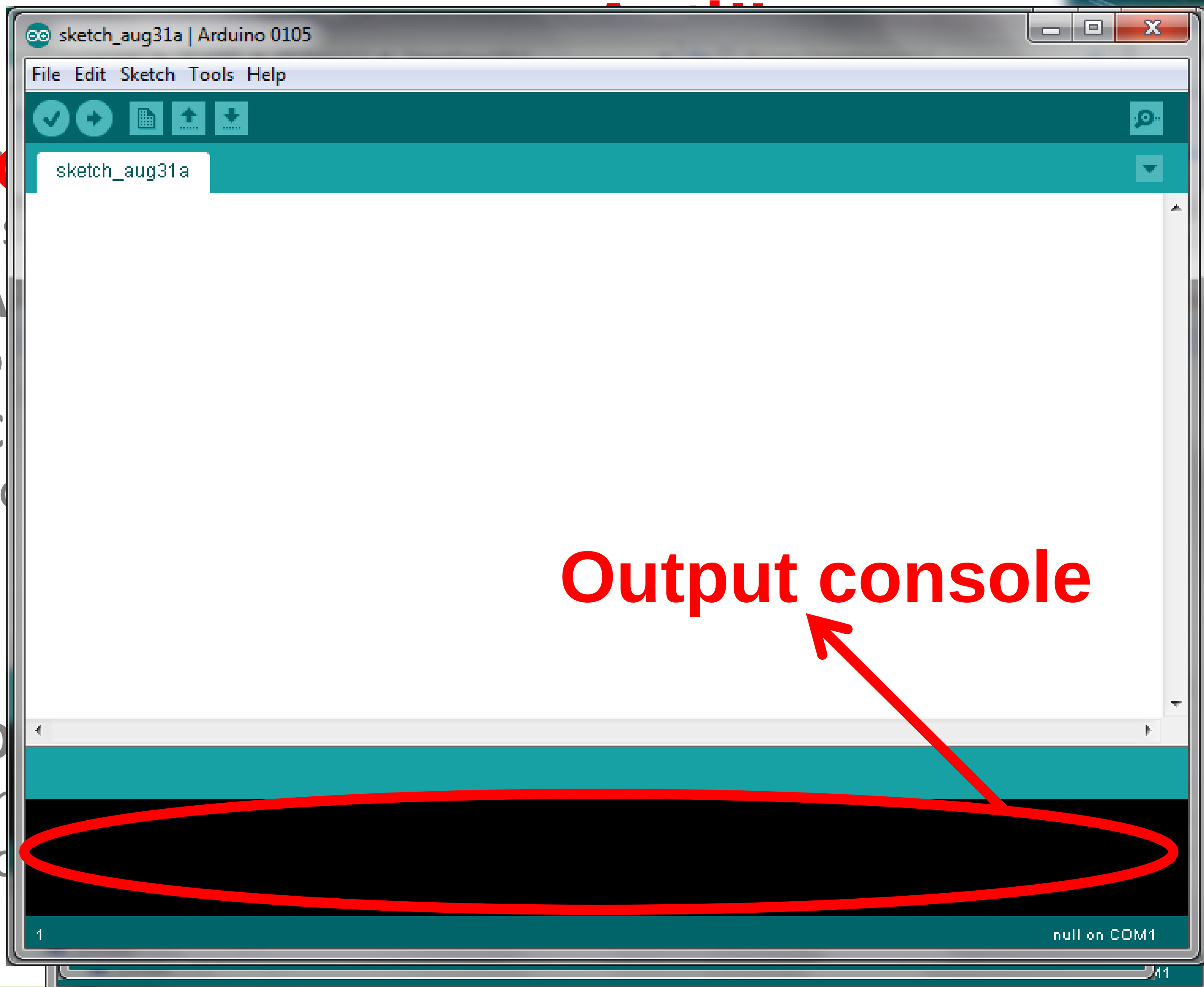


Setting up your environment



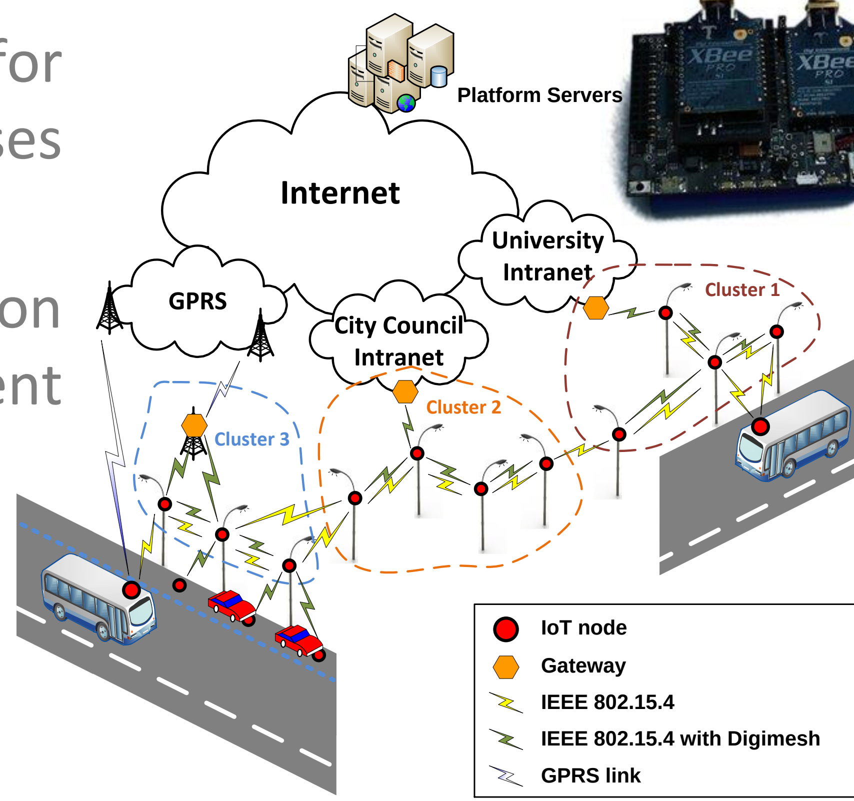
Setting up your environment

- The
  - T ... ell
    as
  - A
    o
  - C
    c
    d
  - O
    m
    (c



SmartSantander Waspmote devices

- SmartSantander devices supports simultaneous service provision and experimentation capabilities.
- Equipped with two transceivers
  - IEEE 802.15.4 for experimentation purposes (freely accessible)
  - Digimesh for service provision and testbed management (protected usage)



Programming API

- To guarantee the service provision and manage the different nodes deployed (fixed and mobile nodes), major modifications of the Libelium Waspmote libraries have been carried out.
  - Integrated in the IDE you have downloaded
  - Accessible through `smartExpTools` class
- Experimenters will have access exclusively to these functions, using other functions to manage or control the node IS NOT ALLOWED.

Programming API

- Functions available
  - Manage the IEEE 802.15.4 interface
  - Handle the incoming packets
  - Retrieve sensor data
  - Miscellanea:
    - Manage the time
    - Use EEPROM memory
    - Send and receive messages to/from the experimenter

Programming API

- Managing the IEEE 802.15.4 interface
  - void **init802**();
    - It initiates the 802.15.4 interface
  - void **stop802**();
    - It stops the 802.15.4 interface
  - uint8_t **sendPacket**(uint8_t* address, uint8_t* data, uint8_t length);
    - It sends a packet to other nodes' XBee modules
    - `uint8_t* address`: MAC address where to send the packet to
    - `uint8_t* data`: pointer to buffer containing the payload to send
    - `uint8_t length`: data buffer length
    - return 0 on success, 1 if there was an error sending the packet, 2 otherwise.
  - uint8_t **sendBcastPacket**(uint8_t* data, uint8_t length);
    - It sends a packet in broadcast mode
    - `uint8_t* data`: pointer to buffer containing the payload to send
    - `uint8_t length`: data buffer length
    - return 0 on success, 1 if there was an error sending the packet, 2 otherwise.

Programming API

- Managing the IEEE 802.15.4 interface
  - void **receivePacket**(unsigned int timeout);
    - It receives a packet from other nodes's XBee modules
    - `unsigned int timeout`: max time in milliseconds waiting for packets
  - uint8_t **isPacketAvailable**();
    - It checks whether is a packet available in the 802.15.4 interface
    - return 1 on success, 0 otherwise
  - void **setChannel802**(uint8_t channel);
    - It sets the channel in 802.15.4 (Range from 0x0B-0x1A)
    - `uint8_t channel`: 802.15.4 channel
  - void **setID802**(uint8_t* panID);
    - It sets the PAN ID in 802.15.4
    - `uint8_t* panID`: PAN ID of 802.15.4 network

Programming API

- Managing the IEEE 802.15.4 interface
  - void **setEE802**(uint8_t encryption);
    - It selects whether to use encryption or not
    - `uint8_t encryption`: 0:Disabled; 1:Enabled
  - void **setKey802**(const char* key);
    - It sets the encryption key in case encryption is selected
    - `char* key`: string to set as encryption key (16-bytes long)

Programming API

- Managing the incoming packets
  - uint8_t* **getPacketData**();
    - Reads data from the packet available
    - return uint8_t* with data available (This is a STATIC pointer). NULL otherwise.
  - uint8_t **getPacketLength**();
    - Provides the length of the data available
    - return uint8_t with the length. 0 otherwise.
  - uint8_t* **getPacketMac**();
    - Reads the source MAC address for the packet available.
    - return uint8_t* with the MAC address (This is a STATIC pointer). NULL otherwise.
  - uint8_t **getPacketRSSI**();
    - It returns the RSSI from the latest packet received
    - return uint8_t with the latest packet RSSI (0x58 == -88 dBm)
  - void **discardPacket**();
    - Discards a packet from the FIFO incoming packets buffer

Programming API

- Managing sensor readings
  - float **sensorReadTemp**();
    - Returns temperature sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format
  - float **sensorReadLight**();
    - Returns light intensity sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format
  - float **sensorReadNoise**();
    - Returns noise sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format

Programming API

- Managing sensor readings
  - float **sensorReadHumidity**();
    - Returns relative humidity sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return float value in float format
  - float **sensorReadWatermark**()
    - Returns soil moisture sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format
  - float **sensorReadSoilTemp**();
    - Returns soil temperature sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format

Programming API

- Managing sensor readings
  - float **sensorReadRadiation**();
    - Returns solar radiation sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format
  - float **sensorReadPressure**();
    - Returns atmospheric pressure sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format
  - float **sensorReadAnemometer**();
    - Returns wind speed sensor value. If sensor is not present in that node or no measure has been done yet, it will return -1
    - return value in float format

Programming API

- Managing the time
  - unsigned long **getMillis**();
    - Wrapper for millis() function. Return current millis counter from the microprocessor
    - return unsigned log with current millis counter (from microprocessor)
  - void **setDelay**(unsigned long time);
    - Wrapper for delay() function. It waits for the time passed to the function.
    - `unsigned long time`: time in millis to wait.
  - const char* **getTimestamp**();
    - Returns the timestamp from the RTC
    - return char* with time in format YYYYMMDDHHmmSS

Programming API

- Miscellanea
  - int **readEEPROMexp**(int pos);
    - used to read values from EEPROM.
    - `int pos`: memory position to read from
    - return int with the value read, -1 if the position is not available. (Allowed position between 1024 and 2048).
  - int **writeEEPROMexp**(int pos, uint8_t value);
    - used to write values into the EEPROM.
    - `int pos`: memory position to write to
    - `uint8_t value`: byte to write in the selected position
    - return 0 if the operation worked successfully, -1 if the position is not available. (Allowed position between 1024 and 2048).

Programming API

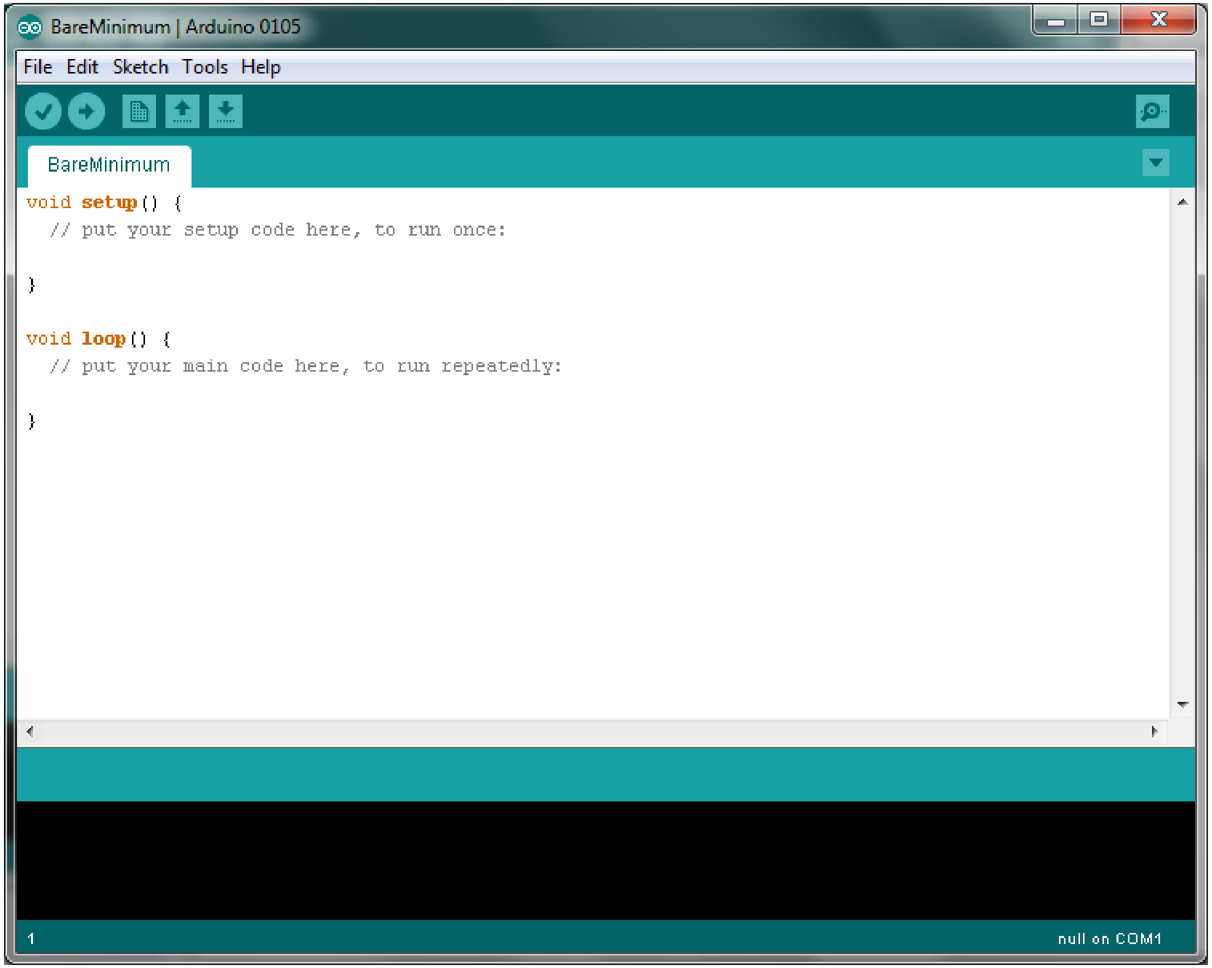
- Miscellanea
  - uint8_t **sendLog**(uint8_t* data, uint8_t len);
    - It sends a log created by the experimenter towards the SmartSantander backbone
    - `uint8_t* data`: data to be sent
    - `size_t len`: length of data (maximum data length is 68)
  - uint8_t **readLog**(uint8_t* dataRecv);
    - Read packet sent to the node from the SmartSantander backbone (i.e. experiment controller)
    - `uint8_t* dataRec`: array to store the log received. Maximum length of 62
    - return length of the log received.
  - uint8_t **islogAvailable**();
    - Checks whether there is available a packet coming from the backbone
    - return 1 on success, 0 otherwise

Sample program

- Target experiment:
  - Assess the RSSI between nodes in a WSN
- Strategy:
  - Send a beacon message every 10 seconds from every node
  - Upon reception of a packet on one of the nodes, we extract the sender MAC address and the received RSSI
  - Send a packet towards the experiment controller with this information for offline processing

Sample program

- Bare minimum



Sample program

- Definition of variables and constants

```
*   Example code for an experimentation program. It sends
*   a packet every ten seconds with a specific id. If a
*   packet is received with that id, a log message, with
*   the MAC address and RSSI, is sent towards the
*   SmartSantander backbone .
*
*   Remarks:
*
*   @author = TLMAT_UC
*   Version: 0.00
*****************************************************/
#define DELAY_RECEIVE 10000

uint8_t id = 0xA0;
int i = 0;
int len;
unsigned long previous;
uint8_t data[9];
```

Sample program

- Bootstrap actions

```
void setup()
{
  smartExpTools.init802();
}
```

Sample program

- Continuous operation

```
void loop()
{
    smartExpTools.sendBcastPacket(&id,1);
    previous = smartExpTools.getMillis();
    while (smartExpTools.getMillis() - previous < DELAY_RECEIVE)
    {
      smartExpTools.receivePacket(1);
      if (smartExpTools.isPacketAvailable())
      {
        processPkts();
        smartExpTools.discardPacket();
      }
      if(previous > smartExpTools.getMillis()) {previous = smartExpTools.getMillis();}
    }
}
```

Sample program

- Structured programming

```
void processPkts()
{
  switch (*(smartExpTools.getPacketData()))
  {
    case 0xA0:
            memcpy(data, smartExpTools.getPacketMac(),8);
            data[8] = smartExpTools.getPacketRSSI();
            smartExpTools.sendLog(data,9);
            break;
    default:
            smartExpTools.sendLog("UNKNOWN_PACKET", 14);
            break;
   }
}
```

Conclusions

- Easy to use library
- Extensible upon request from experimenter
- Soon to come
  - Online compilation tool
  - Try your applications in Santander

SMART SANTANDER

SmartSantander Service Experimentation

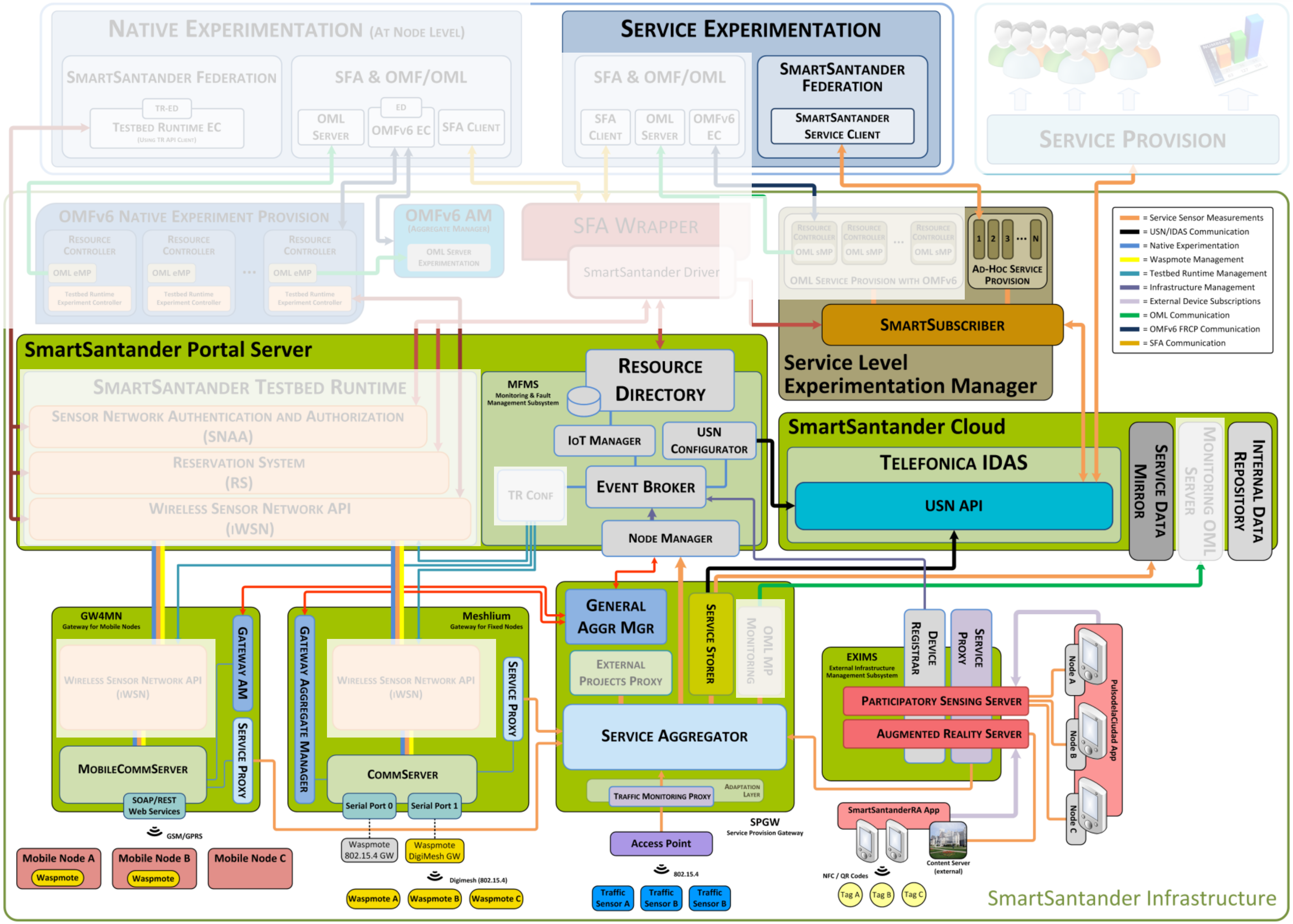
Outline

- Introduction and principles
- Service Level Experimentation Manager
- Applicability examples

Introduction and principles

- Experimenting with the applicability of the IoT infrastructure
  – Infrastructure deployed in Santander around 4 real-world scenarios:
    - Parking management
    - Traffic monitoring
      – At city entrances
      – Through public vehicles
    - Environmental monitoring
      – Air quality
      – Weather conditions
    - Precision irrigation

Introduction and principles



Introduction and principles

- No need to make node reservation
  - Nodes are shared among all experimenters
  - Even if they are reserved for node-level experimentation
- Access to information generated by the infrastructure
  - Last values
  - Historic information

Service Level Experimentation Manager

- General terms
  - REST interface
  - HTTP GET method

```
GET /SENZATIONS/GetLastNoise HTTP/1.1
Host: slem.smartsantander.eu
```

  - Return JSON

```
HTTP/1.1 200 OK
Date: Wed, 21 Aug 2013 17:25:36 GMT
Content-Type: application/json
```

[{"date":"2013-08-21 19:25:23","nodeId":"742","latitude":"43.46078","longitude":"-3.81848","noise":"60.00"},
{"date":"2013-08-21 19:24:40","nodeId":"737","latitude":"43.45487","longitude":"-3.81289","noise":"65.00"},
{"date":"2013-08-21 19:24:39","nodeId":"275","latitude":"43.46258","longitude":"-3.79933","noise":"57.00"},
{"date":"2013-08-21 19:24:12","nodeId":"258","latitude":"43.46262","longitude":"-3.80161","noise":"60.00"},
{"date":"2013-08-21 19:23:51","nodeId":"180","latitude":"43.46301","longitude":"-3.79621","noise":"85.00"},
{"date":"2013-08-21 19:23:41","nodeId":"407","latitude":"43.46294","longitude":"-3.79703","noise":"83.00"},
{"date":"2013-08-21 19:23:34","nodeId":"398","latitude":"43.46262","longitude":"-3.79819","noise":"64.00"}]

Service Level Experimentation Manager

- Host
  - slem.smartsantander.eu
- Per experimenter domain
  - SENZATIONS
- Methods and parameters:
  - **GetFixedNodes**
    - http://slem.smartsantander.eu/SENZATIONS/GetFixedNodes
    - Description: Return the id, type and location of all fixed nodes deployed
    - Return array of JSON {"nodeId":“XXX","type":“xxx","latitude":“xx.xxxx","longitude":“xx.xxxx"}
      - nodeId: String representation of node identifier
      - type: Descriptor of the type of node according to the sensors with which it is equipped
      - latitude: String representation of float value. Associated to the node location
      - longitude: String representation of float value. Associated to the node location
  - **GetMobileNodes**
    - http://slem.smartsantander.eu/SENZATIONS/GetMobileNodes
    - Description: Return the id and type of all mobile nodes deployed
    - Return array of JSON {"nodeId":“xxx","type":“xxx"}

Service Level Experimentation Manager

- Methods and parameters:
  - **GetSensorTypes**
    - http://slem.smartsantander.eu/SENZATIONS/GetSensorTypes
    - Description: Return a list of available sensor type descriptors.
    - Return array of JSON {"type":"xxx"}
  - **GetLastValuesBySensorType**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastValuesBySensorType/SensorType
      - **SensorType:** Descriptor of the type of sensors from which you want last values
    - Description: Return the last observations from all the nodes of a particular type. If a node has more than one sensor, it will provide a list of "*phenomenon":"value*" pairs. One for each sensor.
    - Return array of JSON {"nodeId":"xx","type":"xxx","longitude":"xx.xxxx","latitude":"xx.xxxx","battery":"xx.xx", "date":"YYYY-MM-DD hh:mm:ss", "*phenomenon":"value", "phenomenon":"value",*…}
      - battery: (%) String representation of float value
      - date: timestamp of the observation
      - phenomenon: physical parameter observed
      - value: String representation of phenomenon value
      - latitue and longitude: String representation of float value. Associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetHistoricByNodeID**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricByNodeID/Date1/Date2/NodeId
      - **Date1:** From boundary of the historic record. In format: *YYYY-MM-DD hh:mm:ss*
      - **Date2:** To boundary of the historic record. In format: *YYYY-MM-DD hh:mm:ss*
      - **NodeId:** Identifier of the node from which observations wants to be retrieved
    - Description: Return the observations from the selected node within the boundaries of the selected dates
    - Return observations as an array of JSON {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","battery":"xx.xx", *"phenomenon":"value",phenomenon":"value",…*}
      - battery value is not present on all nodes
      - latitude and longitude is only present if the requested node is a mobile node. Associated to the observation
  - **GetHistoricByLocation**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricByLocation/Date1/Date2/Lat/Lon/Dist
      - **Lat:** Latitude coordinate for the position around which nodes are to be queried
      - **Lon:** Longitude coordinate for the position around which nodes are to be queried
      - **Dist:** Maximum distance (in Km) from the defined position
    - Description: Return the observations got from any node less than *Dist* kilometres away from the *<LatLon>* position given within the boundaries of the selected dates
    - Return observations as an array of JSON {"nodeId":"xx","longitude":"xx.xxxx","latitude":"xx.xxxx", "battery":"xx.xx", "date":"YYYY-MM-DD hh:mm:ss", *"phenomenon":"value", "phenomenon":"value",…*}
      - battery value is not present on all nodes
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetLastTemperature**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastTemperature/
    - Description: Return the last temperature observations from any node in the infrastructure equipped with a temperature sensor
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “temperature”:”value”}
      - temperature: (celsius) String representation of float value
      - latitude and longitude are associated to the observation
  - **GetHistoricTemperature**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricTemperature/Date1/Date2
    - Description: Return all the temperature observations within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “temperature”:”value”}
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetHistoricTemperatureByLocation**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricTemperatureByLocation/Date1/Date2/Lat/Lon/Dist
    - Description: Return all the temperature observations less than *Dist* kilometres away from the *<LatLon>* position given within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", *“temperature”:”value”*}
      - latitude and longitude are associated to the observation
  - **GetLastNoise**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastNoise/
    - Description: Return the last noise observations from any node in the infrastructure equipped with a noise sensor
    - Return array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", *“noise”:”value”*}
      - noise: (decibel) String representation of integer value
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetHistoricNoise**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricNoise/Date1/Date2
    - Description: Return all the noise observations within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","nodeId":"xxx", *"noise":"value"*}
      - latitude and longitude are associated to the observation
  - **GetHistoricNoiseByLocation**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricNoiseByLocation/Date1/Date2/Lat/Lon/Dist
    - Description: Return all the noise observations less than *Dist* kilometres away from the *<LatLon>* position given within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","nodeId":"xxx", *"noise":"value"*}
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetLastHumidity**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastHumidity/
    - Description: Return the last relative air humidity observations from any node in the infrastructure equipped with a humidity sensor
    - Return array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “relativeHumidity”:”value”}

      - relativeHumidity: (%) String representation of float value
      - latitude and longitude are associated to the observation
  - **GetHistoricHumidity**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricHumidity/Date1/Date2
    - Description: Return all the relative air humidity observations within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “relativeHumidity”:”value”}

      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetHistoricHumidityByLocation**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricHumidityByLocation/Date1/Date2/Lat/Lon/Dist
    - Description: Return all the relative air humidity observations less than `Dist` kilometres away from the `<LatLon>` position given within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", *“relativeHumidity”:”value”*}
      - latitude and longitude are associated to the observation
  - **GetLastAirQuality**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastAirQuality/
    - Description: Return the last air-quality-related observations from any node in the infrastructure equipped with air pollutants sensors
    - Return array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", *"CO":“value","particles":"value","ozone":"value","NO2":"value"*}
      - CO: (mg/m^3) String representation of float value
      - particles: (mg/m^3) String representation of float value
      - ozone: ($\mu g/m^3$) String representation of float value
      - NO_2: ($\mu g/m^3$) String representation of float value
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetHistoricAirQuality**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricAirQuality/Date1/Date2
    - Description: Return all the air-quality-related observations within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", *"CO":“value","particles":"value","ozone":"value","NO2":"value"*}
      - latitude and longitude are associated to the observation
  - **GetHistoricAirQualityByLocation**
    - http://slem.smartsantander.eu/SENZATIONS/GetHistoricAirQualityByLocation/Date1/Date2/Lat/Lon/Dist
    - Description: Return all the air-quality-related observations less than *Dist* kilometres away from the *<LatLon>* position given within the boundaries of the selected dates
    - Return observations as an array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", *"CO":“value","particles":"value","ozone":"value","NO2":"value"*}
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetLastTrafficSpeed**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastTrafficSpeed
    - Description: Return the vehicle speed related observations in the last 10 minutes from any node in the infrastructure equipped with speed meter sensors. Nodes in vehicles return speed of the vehicle they are mounted on. Fixed nodes in roads detect speed of vehicles going above them
    - Return array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “*average_speed*":“*value*”}

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “speed":“value“,”course”:”value”}

      - average_speed: (Km/h) String representation of float value. Observation generated by fixed node
      - speed: (Km/h) String representation of float value. Observation generated by mobile node
      - course: (degrees) String representation of integer value. 0^{o} represent North. Only present in mobile nodes’ observations
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetLastTrafficSpeedByLocation**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastTrafficSpeedByLocation/Lat/Lon/Dist
    - Description: Return the vehicle speed related observations in the last 10 minutes less than *Dist* kilometres away from the *<LatLon>* position given
    - Return observations as an array of JSON

      {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","nodeId":"xxx", "*average_speed":"value*"}

      {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","nodeId":"xxx", "speed":"value","course":"value"}
      - average_speed: (Km/h) String representation of float value. Observation generated by fixed node
      - speed: (Km/h) String representation of float value. Observation generated by mobile node
      - course: (degrees) String representation of integer value. 0º represent North. Only present in mobile nodes' observations
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

- Methods and parameters:
  - **GetLastTrafficIntensity**
    - http://slem.smartsantander.eu/SENZATIONS/GetLastTrafficIntensity
    - Description: Return the traffic intensity related observations in the last 10 minutes from any node in the infrastructure.
    - Return array of JSON

      {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","nodeId":"xxx", "count":"value","occupancy":"value"}
      - count: (unitless) String representation of integer value. Number of vehicles passing over the node in the last minute
      - occupancy: (%) String representation of float value. Relative assessment of lane occupation
      - latitude and longitude are associated to the observation
  - **GetParkingStatus**
    - http://slem.smartsantander.eu/SENZATIONS/GetParkingStatus
    - Description: Return the status (FREE or OCCUPIED) of the parking lots in the city controlled through car presence detection sensors
    - Return array of JSON

      {"date":"YYYY-MM-DD hh:mm:ss","longitude":"xx.xxxx","latitude":"xx.xxxx","nodeId":"xxx", "*status*":"*value*"}
      - status: String enumeration {FREE, OCCUPIED}
      - latitude and longitude are associated to the observation

Service Level Experimentation Manager

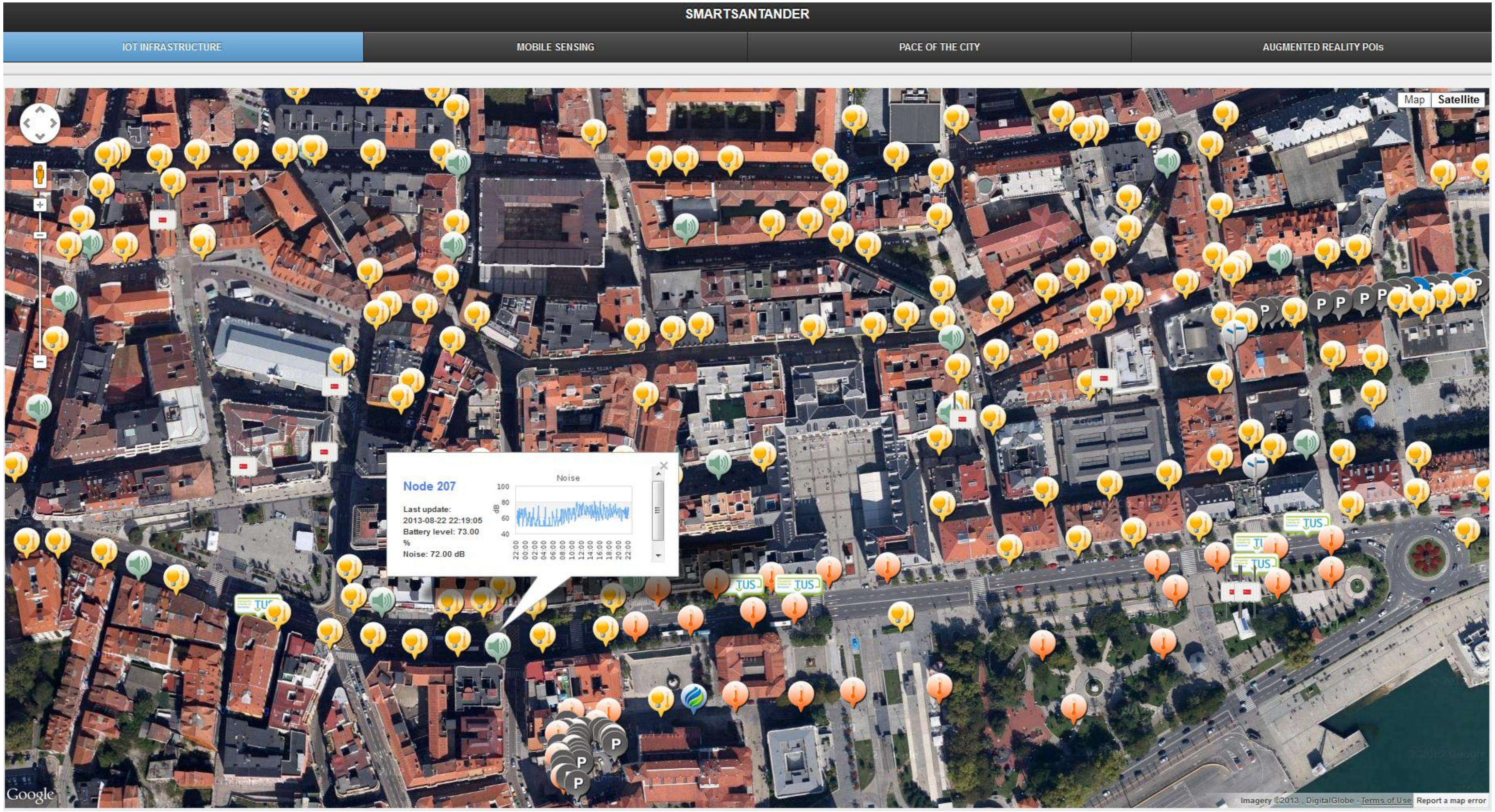
- Methods and parameters:
  - **GetNodesBatteryStatus**
    - http://slem.smartsantander.eu/SENZATIONS/GetNodesBatteryStatus
    - Description: Return the amount of battery remaining on the nodes equipped with rechargeable batteries
    - Return array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “battery":“value"}
      - battery: (%) String representation of float value
      - latitude and longitude are associated to the node
  - **GetNodesBatteryStatusBelowThr**
    - http://slem.smartsantander.eu/SENZATIONS/GetNodesBatteryStatus/Thr
      - **Thr:** Threshold value
    - Description: Return the nodes whose battery remaining is below the specified threshold
    - Return array of JSON

      {"date":“YYYY-MM-DD hh:mm:ss”,"longitude":“xx.xxxx","latitude":“xx.xxxx",“nodeId":“xxx", “battery":“value"}
      - battery: (%) String representation of float value
      - latitude and longitude are associated to the node

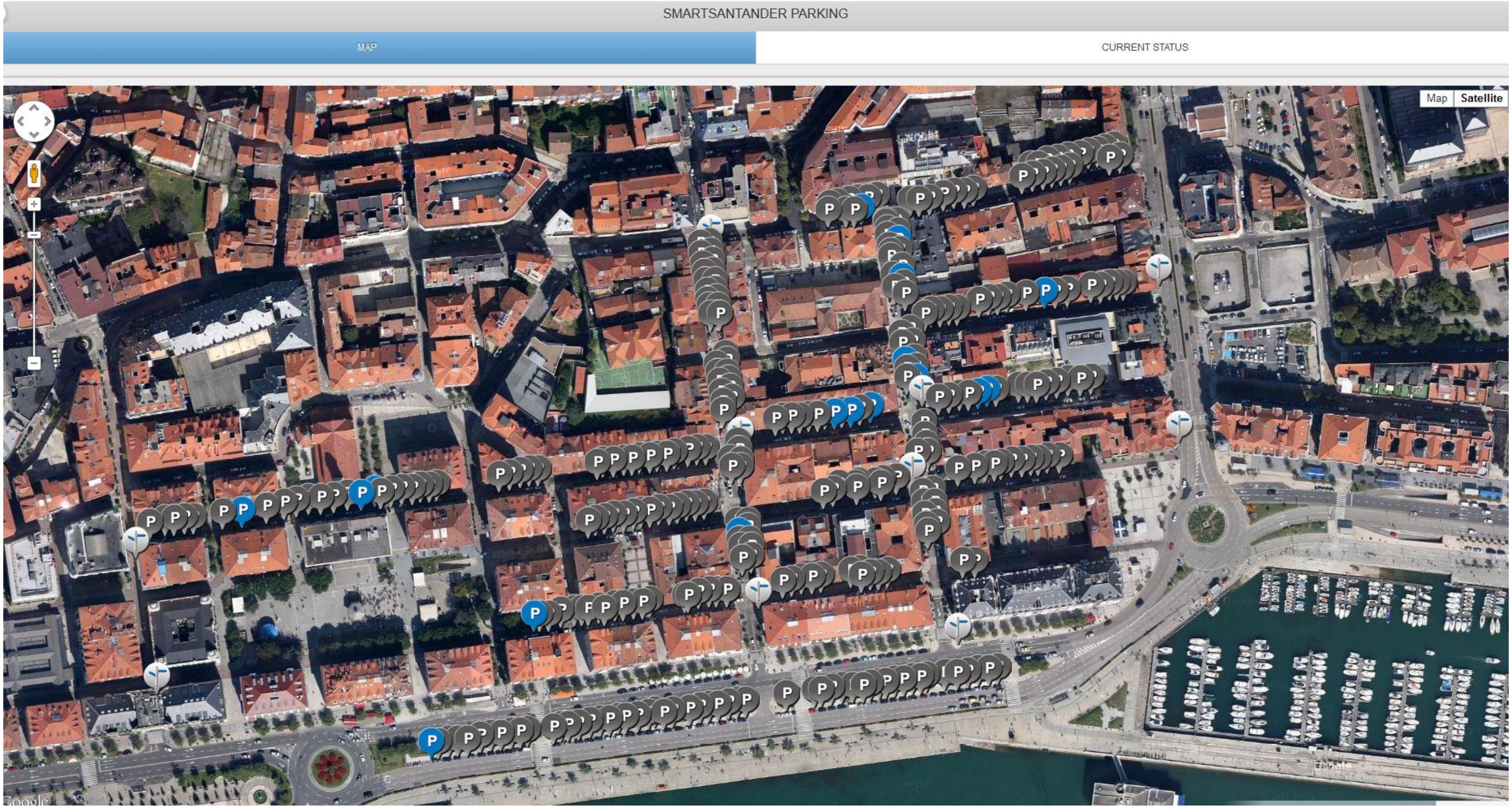
Applicability examples

- Show information and plot last 24 hours activity



Applicability examples

- Available parking lots



Applicability examples

- Noise map of the city

Applicability examples

- Gardens irrigation status map

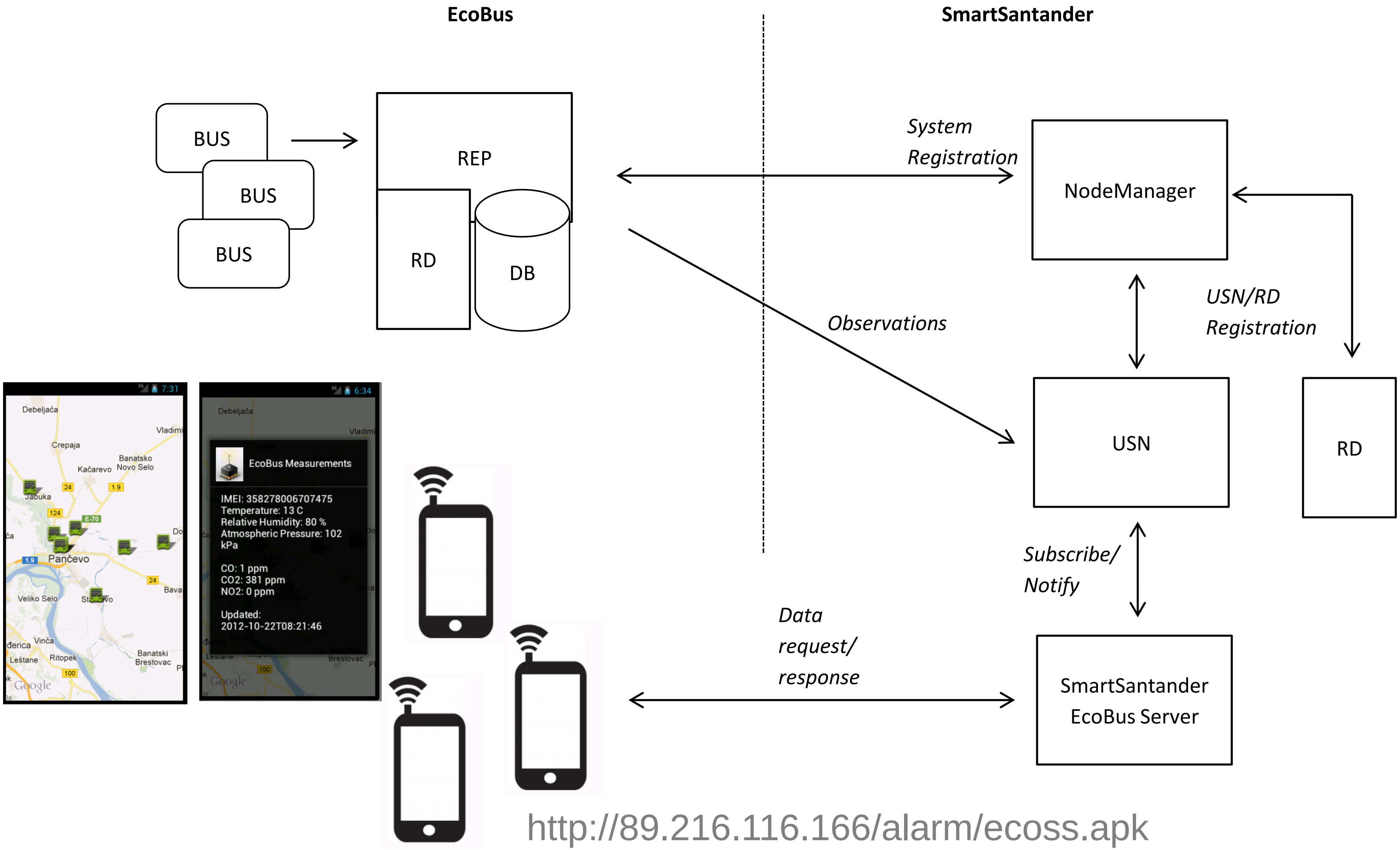
Soil Moisture Tension

Soil Moisture Temperature

SMART SANTANDER

Belgrade testbed service access

Integration – technical details



http://89.216.116.166/alarm/ecoss.apk

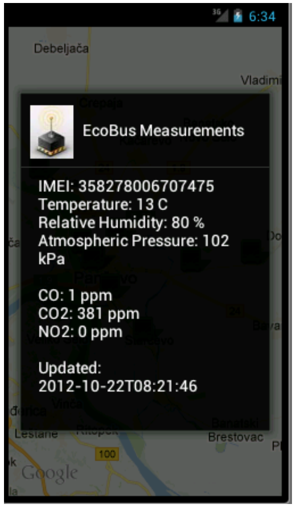
EkoBus service access

- Access to the environmental measurements and bus location
  - http://89.216.116.166/post/post?imei=357467030477053
- Measurements listing
  - http://89.216.116.166/post/post?imei=
- Response is XML formatted message

Data format

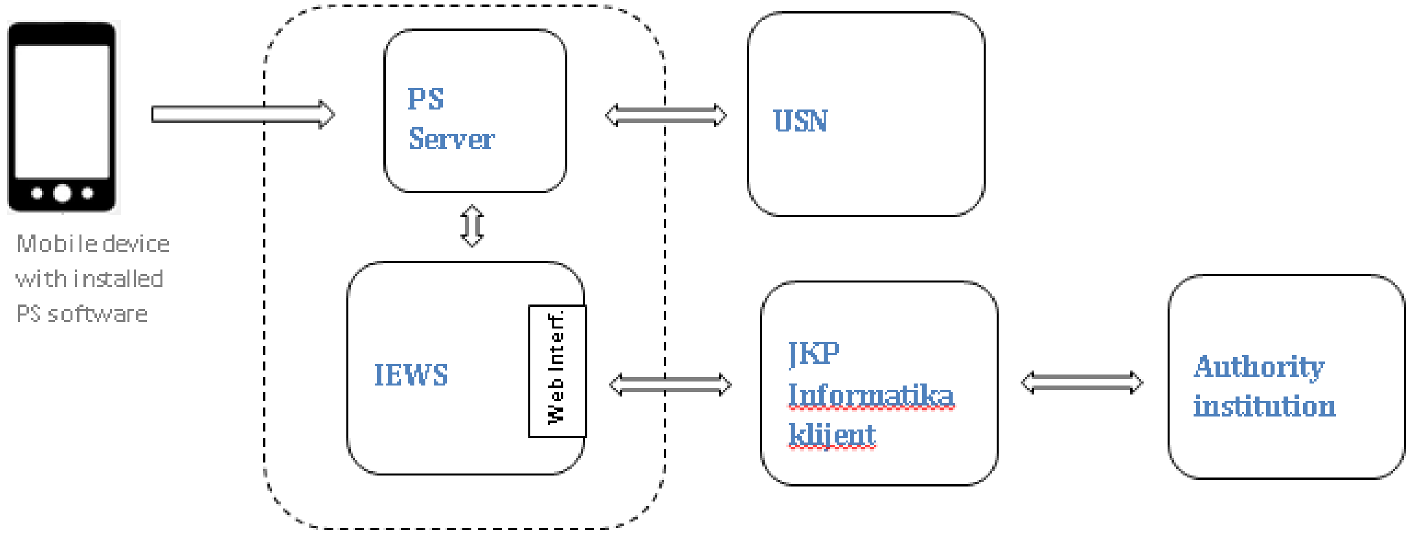
- HTTP GET
  - http://89.216.116.166/post/post?imei=357467030477053
- 357467030477053 is device IMEI
- Response

<ed>
<ie>357467030477053</ie>
<co>0</co>
<co2>834</co2>
<no2>0</no2>
<temp>19</temp>
<hum>50</hum>
<press>102</press>
<lat>44.8648383333333</lat>
<lon>20.7381633333333</lon>
<time>2013-08-30T14:29:00.000000</time>
</ed>



- Empty IMEI parameter returns all devices
- Root element is *<curr>*

Block scheme of the integration



- JKP informatika has services for event authority notifying
- Existing PS server component is extended to communicate with IEWS

Participatory sensing

- IEWS service
  - Listing events
    - http://89.216.116.166/iews/events
    - ../iews/informatikaevents/{crticalRate}/{remove}
      - Only rated events, remove after reading
  - Event rate
    - Rate event (anonymously)
      - http://89.216.116.166/iews/rate/{event_id}/{rate}
        - » Rate is +/-
    - Rate event
      - http://89.216.116.166/iews/ratebyuuid/{event_id}/{deviceUUID-rater}/{rate}
      - Device UUID is assigned by server during the first application start
  - User service http://89.216.116.166/iews/{user}/{deviceUUID}
    - Rest based interface for user management

SEVENTH FRAMEWORK PROGRAMME

SMART SANTANDER

Programming Heterogeneous IoT Platforms
The Wiselib

Learning Goals – Understand This!

```cpp
#include "external_interface/external_interface.h"
#include "util/delegates/delegate.hpp"
#include "util/pstl/map_static_vector.h"
#include "util/pstl/static_string.h"
#include "util/wisebed_node_api/sensors/sensor_controller.h"
#include "util/wisebed_node_api/sensors/managed_sensor.h"

typedef wiselib::OSMODEL Os;
typedef wiselib::MapStaticVector<Os, uint8_t, wiselib::pair<wiselib::StaticString, delegate0<char*> >, 10> sensor_map_t;
typedef wiselib::SensorController<Os, sensor_map_t, wiselib::StaticString> sensor_controller_t;

class SensorTestApplication
{
   public:
      void init( Os::AppMainParameter& value )
      {
         timer_ = &wiselib::FacetProvider<Os, Os::Timer>::get_facet( value );
         debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet( value );
         sensor_controller_.init( *debug_ );
         #ifdef SHAWN
         #endif
         #ifdef ISENSE
            managed_light_sensor_.init_with_facetprovider( value, "light" );
            sensor_controller_.add_sensor( 1, managed_light_sensor_.name(), managed_light_sensor_.sensor_delegate() );
            sensor_controller_.add_sensor( 42, managed_light_sensor_.name(), managed_light_sensor_.sensor_delegate() );
         #endif
         timer_->set_timer<SensorTestApplication, &SensorTestApplication::read_values> (10000, this, 0);
      }
      void read_values( void* )
      {
         debug_->debug( "read sensor values\n" );
         debug_->debug( "has 1: %d, has 2: %d, has 42: %d", sensor_controller_.has_sensor(1),
                  sensor_controller_.has_sensor(2), sensor_controller_.has_sensor(42) );
         debug_->debug( "sensors:\n%s", sensor_controller_.sensors().c_str() );
         char* value = sensor_controller_.value(1);
         debug_->debug( "sensor 1: %d %d %d", value[0], value[1], value[2] );
         value = sensor_controller_.value(42);
         debug_->debug( "sensor 42: %d %d %d", value[0], value[1], value[2] );
         timer_->set_timer<SensorTestApplication, &SensorTestApplication::read_values> (10000, this, 0);
      }
   private:
      Os::Timer::self_pointer_t timer_;
      Os::Debug::self_pointer_t debug_;
#ifdef ISENSE
      ManagedLightSensor managed_light_sensor_;
#endif
      sensor_controller_t sensor_controller_;
};

wiselib::WiselibApplication<Os, SensorTestApplication> application;
// --------------------------------------------------------------------------
void application_main( Os::AppMainParameter& value )
{
   application.init( value );
}
```

(Platform-independent Wiselib application to read sensor values and write them to UART)

Learning Goals

- After this tutorial you will…
  - Understand the basics of C++ template programming (necessary for Wiselib)
  - Understand why Wiselib is *awesome*
  - Understand basic concepts and architecture
  - Know how to write a (very) basic platform-independent Wiselib application
  - Know where to find source code, documentation, and applications
  - … be hungry for more ☺

Learning Goals

- After this tutorial you will not…
  - Know how to write application **X** with Algorithm **Y** for OS **Z**
  - Be an expert in Wiselib

Outline

1. Introduction – What is the Wiselib?
2. Basics: C++ Template Programming
3. Architecture
   OS Facets, Data Structures, Algorithms & Applications
4. Usage Scenarios
5. Message Serialization
6. Callback Mechanism
7. Programming Environments
8. Summary & Resources

Programming Heterogeneous IoT Platforms – The Wiselib

1. INTRODUCTION – WHAT IS THE WISELIB?

1. Introduction – What is the Wiselib?

- The Wiselib is...
  - a library for networked embedded devices
  - contains re-usable code (just like the C++ STL)
  - platform-independent
  - highly efficient
  - implemented using C++ templates (comparable to Boost)

- The Wiselib contains...
  - a collection of algorithms
  - an abstraction of embedded operating systems
  - utility functions and data structures (pSTL, pMP)

1. Introduction – Algorithm Categories

Algorithm Category	Count
Routing	13
Clustering	9
Time Synchronization	4
Localization	6
Energy Saving Schemes	6
Security	9
Graph Algorithms	8
Target Tracking	2
Neighbourhood Discovery	1
Data Collection	1
10 Categories, Total:	59

From: Dissertation Tobias Baumgartner, 2012/07

1. Introduction – Supported Hardware

	OS	Radio	TX Radio	Ext. Data Radio	Timer	Logging	Clock	Settable Clock	Serial Comm.	Random
Contiki	⊗	⊗		•	⊗	⊗	⊗		⊗	
TinyOS	⊗	⊗		•	⊗	⊗	⊗		⊗	
iSense	⊗	⊗	•	•	⊗	⊗	⊗	•	⊗	⊗
ScatterWeb	⊗	⊙			⊗	⊗				
Shawn	⊗	⊗	•	•	⊗	⊗	⊗		⊗	⊗
OpenCom	⊗	⊙			⊙	⊙				
Linux	⊗	⊗			⊗	⊗	⊗			⊗
μkleos	⊗	⊗			⊗	⊗				
TriSOS	⊗	⊗			⊗	⊗	⊗			
iOS	⊗	⊗			⊗	⊗				
Android	⊙	⊙			⊙	⊙				

From: Dissertation Tobias Baumgartner, 2012/07

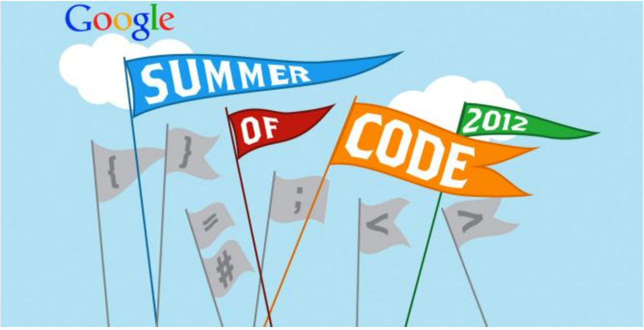
⊗ = Fully Supported

• = Supported, still in testing stage

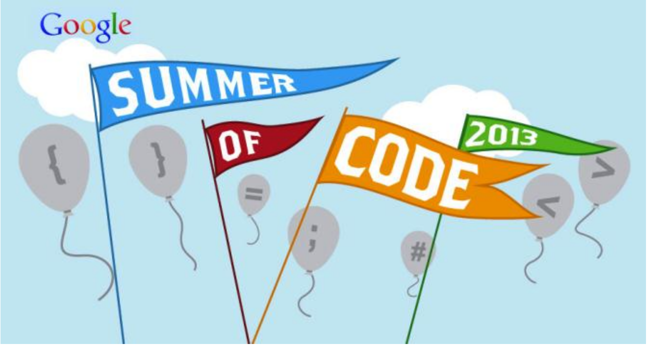
1. Introduction – GSoC Projects

- Wiselib has been/is mentoring organization at Google Summer of Code in 2012 & 2013 (!)

- Projects 2012:
  - OpenWrt Port
  - Android Port
  - Arduino Port
  - 6LoWPAN



- Projects 2013:
  - NS-3 Port
  - IPv6 on Distributed Protocol Stacks
  - Remote-controlled IoT with JS
  - Wisebender Online-IDE
  - Completion of Arduino Port
  - Constrained Application Protocol (CoAP)



1. Introduction

- The Wiselib is heavily based on C++ template programming in order to…
  - achieve platform independence (heterogeneity)
  - produce highly efficient machine code
  - enable programming in C++ for heavily resource constrained (IoT) devices

- So, let’s look at the basics and properties of template programming…

Programming Heterogeneous IoT Platforms – The Wiselib

2. BASICS: C++ TEMPLATE PROGRAMMING

2. Basics: C++ Template Programming

```cpp
template <typename T>
const T& max(const T& a, const T& b)
{
  return a > b ? a : b;
}
```

Parameterized Library Code (max.hpp)

```cpp
#include <iostream>
#include "max.hpp"

int main()
{
  int a = 0;
  int b = 1;
  std::cout << max(a,b) << std::endl;
}
```

Application (main.cpp)

Template
“instantiation”

“triggers”

```cpp
const int& max(const int& a, const int& b)
{
    return a > b ? a : b;
}
```

Generated Code

Compilation

Executable Binary

2. Basics: C++ Template Programming

Function templates are…

- + instantiated and compiled only for functions that are actually called
- + implicitly inline (allows compiler optimizations)
- - instantiated for every type parameter used
- typically located in header files as implementation source is needed for instantiation

Template Specialization

```cpp
template <typename T>
const T& max(const T& a, const T& b)
{
  return a > b ? a : b;
}

template <>
const char* max(const char* a, const char* b)
{
  return (std::strcmp(a,b) > 0) ? a : b;
}
```

Specialization for char*

- + allows optimizations for concrete types
- - sometimes necessary to work correctly with certain types

Advanced example (multiple type parameters)

```
template <typename T1, typename T2>
class FakeMap
{
private:
  T2 fake_value;
public:
  FakeMap() : fake_value(T2())
  {
    // ready :)
  }
  const void put(const T1& key, const T2& value)
  {
    fake_value = value;
  }
  const T2& get(const T1& key)
  {
    return fake_value;
  }
};
```

2. Basics: C++ Template Programming

Partial Specialization (two identical parameters)

```
template <typename T>
class FakeMap<T,T>
{
private:
  T fake_value;
public:
  FakeMap() : fake_value(T())
  {
    // ready :)
  }
  const void put(const T& key, const T& value)
  {
    fake_value = value;
  }
  const T& get(const T& key)
  {
    return fake_value;
  }
};
```

2. Basics: C++ Template Programming

Partial Specialization (one concrete parameter)

```
template <typename T>
class FakeMap<T,int>
{
private:
  int fake_value;
public:
  FakeMap() : fake_value(0)
  {
    // ready :)
  }
  const void put(const T key, const int value)
  {
    fake_value = value;
  }
  const int get(const T& key)
  {
    return fake_value;
  }
};
```

2. Basics: C++ Template Programming

Partial Specialization (two pointer types)

```
template <typename T1, typename T2>
class FakeMap<T1*,T2*>
{
private:
  T2* fake_value;
public:
  FakeMap() : fake_value(NULL)
  {
    // ready :)
  }
  const void put(const T1* key, const T2* value)
  {
    fake_value = value;
  }
  const T2* get(const T1* key)
  {
    return fake_value;
  }
};
```

2. Basics: C++ Template Programming

Default type parameters

```cpp
#include <vector>

template <typename T, typename CONTAINER=std::vector<T> >
class Stack
{
private:
  CONTAINER container;
public:
  Stack()
  {
    // ready :)
  }
  const void push(const T& elem)
  {
    container.push_back(elem);
  }
  const T& pop()
  {
    T& elem = container.back();
    container.pop_back();
    return elem;
  }
};
```

```cpp
#include <iostream>
#include "stack.hpp"

using namespace std;

int main()
{
  Stack<int> int_stack;
  int_stack.push(1);
  int_stack.push(2);
  cout << int_stack.pop() << endl;
  cout << int_stack.pop() << endl;
}
```

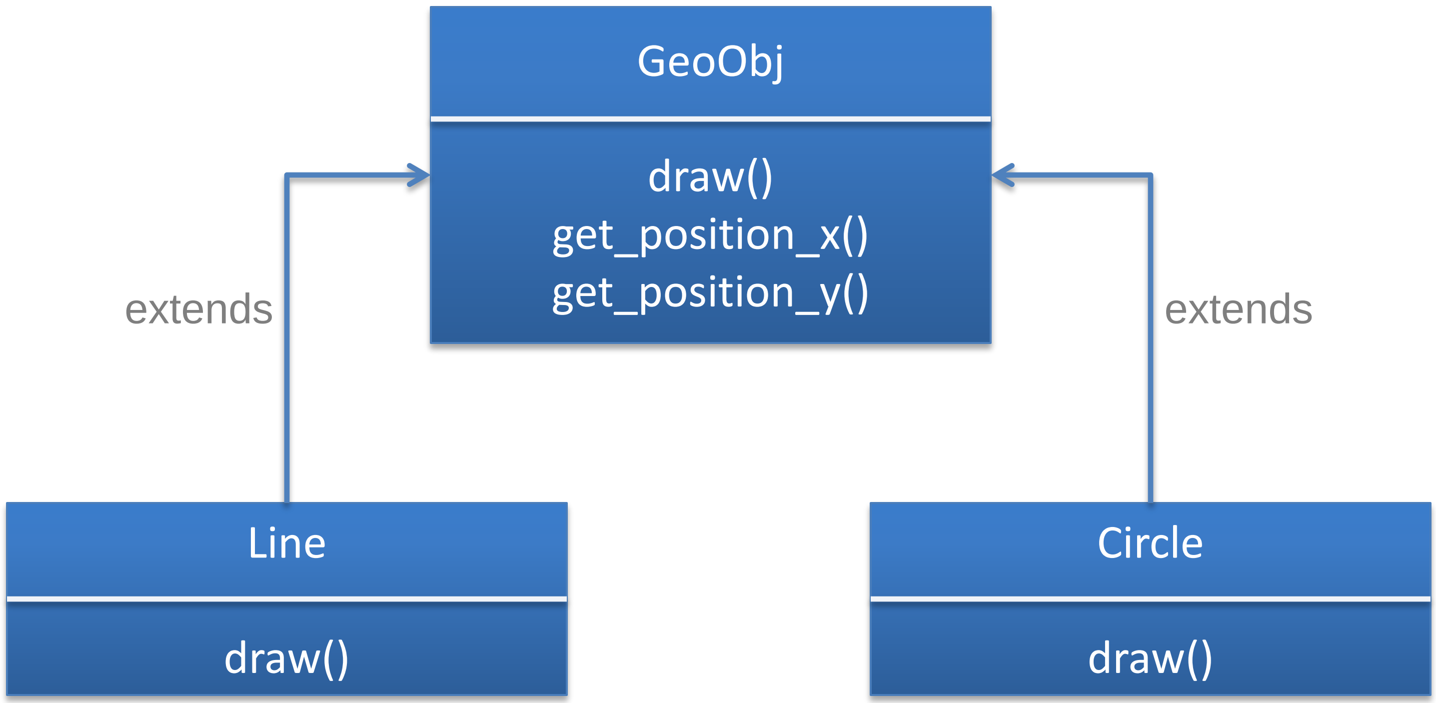

2. Basics: C++ Template Programming

Default value parameters

```
template <typename T, int MAXSIZE=10 >
class Stack
{
private:
  T container[MAXSIZE];
  // ...
};
```

- ➕ dynamic memory management can be avoided (highly efficient)
- only primitive types (not classes) allowed as value parameters

Bound dynamic polymorphism (OO-inheritance)



2. Basics: C++ Template Programming

Bound dynamic polymorphism (OO-inheritance)

```cpp
class GeoObj
{
protected:
  int x;
  int y;
  GeoObj(int x, int y) : x(x), y(y) { }

public:
  virtual void draw() = 0;
  virtual int get_position_x() { return x; }
  virtual int get_position_y() { return y; }
};
```

```cpp
#include <iostream>
#include "geoobj-oo.hpp"

class Line : public GeoObj
{
protected:
  int length;
public:
  Line(int x, int y, int length) : GeoObj(x,y),
                                   length(length) {}
  virtual void draw()
  {
    std::cout << "Line{x=" << x << ",y=" << y <<
        ",length=" << length << "}" << std::endl;
  }
};
```

```cpp
#include <iostream>
#include "geoobj-oo.hpp"

class Circle : public GeoObj
{
protected:
  int radius;
public:
  Circle(int x, int y, int radius) : GeoObj(x,y),
                                     radius(radius) {}
  virtual void draw()
  {
    std::cout << "Circle{x=" << x << ",y=" << y
      << ",radius=" << radius << "}" << std::endl;
  }
};
```

2. Basics: C++ Template Programming

Bound dynamic polymorphism (OO-inheritance)

```
#include <iostream>
#include <vector>

#include "geoobj-oo.hpp"
#include "geoobj-oo_circle.cpp"
#include "geoobj-oo_line.cpp"

using namespace std;

void paint(GeoObj* o)
{
  o->draw();
}

int main()
{

  Line l = Line(0,1,2);
  Circle c = Circle(1,2,3);

  vector<GeoObj*> v = vector<GeoObj*>();
  v.push_back(&l);
  v.push_back(&c);

  for (vector<GeoObj*>::iterator it=v.begin(); it != v.end(); it++)
  {
    paint(*it);
  }
}
```

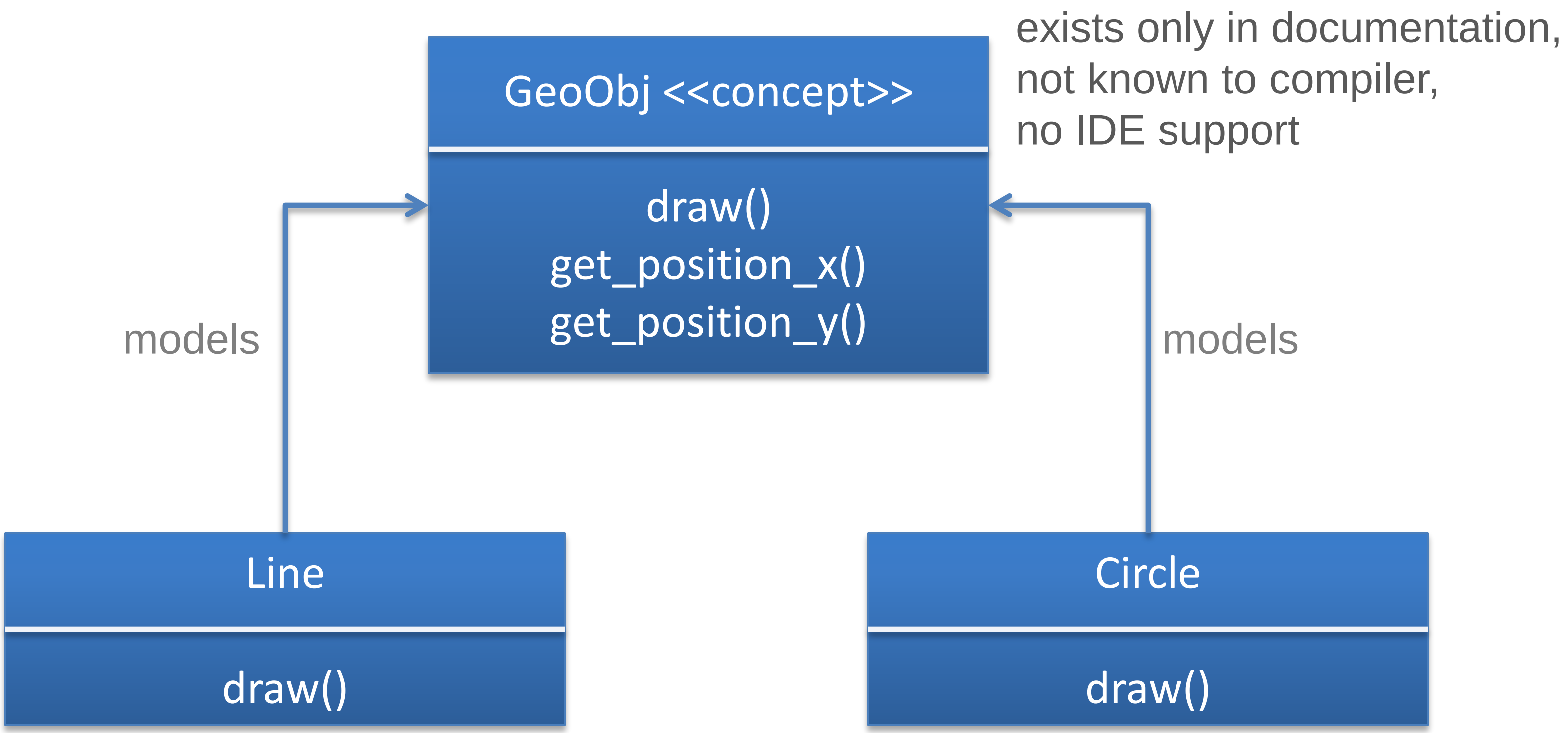
2. Basics: C++ Template Programming

Bound dynamic polymorphism (OO-inheritance)

- \+ Shared code only compiled once
- \+ Allows inhomogeneous sets
  (data structures bound to instances of base class)
- \+ Good compiler error messages

- \- Generates virtual function tables
  (overhead for program memory and runtime)
- \- Virtual methods can not be inlined
  (prevents compiler optimizations)

Unbound static polymorphism (using templates)



2. Basics: C++ Template Programming

Unbound static polymorphism (using templates)

```cpp
#include <iostream>

class Circle
{
private:
  int x;
  int y;
  int radius;

public:
  Circle(int x, int y, int radius) :
      x(y), y(y), radius(radius) {}
  void draw()
  {
    std::cout << "Circle{x=" << x << ",y=" << y
      << ",radius=" << radius << "}" << std::endl;
  }
  int get_position_x() { return x; }
  int get_position_y() { return y; }
};
```

```cpp
#include <iostream>

class Line
{
private:
  int x;
  int y;
  int length;

public:
  Line(int x, int y, int length) :
      x(y), y(y), length(length) {}
  void draw()
  {
    std::cout << "Line{x=" << x << ",y=" << y
      << ",length=" << length << "}" << std::endl;
  }
  int get_position_x() { return x; }
  int get_position_y() { return y; }
};
```

No common base class, but same “interface” (a.k.a. concept)

2. Basics: C++ Template Programming

Unbound static polymorphism (using templates)

```cpp
#include <iostream>
#include <vector>
#include "geoobj-tmpl_circle.hpp"
#include "geoobj-tmpl_line.hpp"
using namespace std;

template <typename GeoObj_P>
void paint(GeoObj_P* o)
{
  o->draw();
}

int main()
{
  Line l = Line(0,1,2);
  Circle c = Circle(1,2,3);
  paint<Line>(&l);
  paint<Circle>(&c);
  // no inhomogeneous sets possible (!)
  // vector<GeoObj*> v = vector<GeoObj*>();
  // v.push_back(&l);
  // v.push_back(&c);
  // for (vector<GeoObj*>::iterator it=v.begin(); it != v.end(); it++)
  // {
  //   paint(*it);
  // }
}
```

2. Basics: C++ Template Programming

Unbound static polymorphism (using templates)

- Concepts exist only in documentation
- Concepts describe “interface” of template class
- Programming: **interfaces and classes**
- Metaprogramming: **concepts and models**
- Multiple inheritance equally possible

Unbound static polymorphism (using templates)

- + Faster & smaller machine code
  (no pointer indirection, no vtables are generated)
- + Enables heavy compiler optimizations (inlining)

- - No inhomogeneous sets possible
  (reason: no common base class, only concepts)
- - Compiler error messages hard to read

Programming Heterogeneous IoT Platforms – The Wiselib

3. ARCHITECTURE

3. Architecture Constraints

Some target architectures are heavily resource-constrained. Wiselib applications must therefore adhere to some restrictions:

- No dynamic memory allocation
  - no *new*, *delete*, *malloc*, *free*, only static allocation (!)
- No STL, use picoSTL
- No Runtime Type Information (RTTI)
  - no dynamic_cast<> -> no type safety checks
- No virtual inheritance

3. Architecture

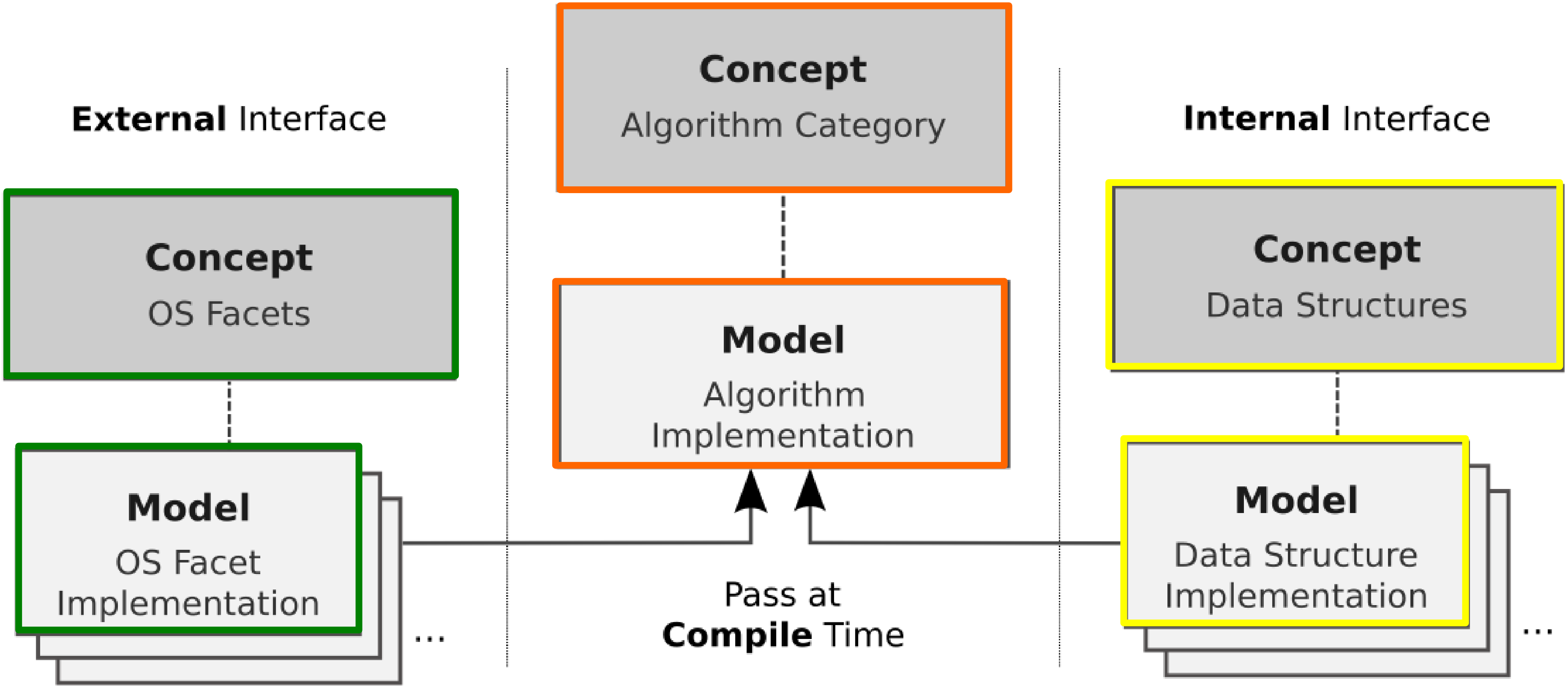


Image Source: Dissertation Tobias Baumgartner, 2012

3. Architecture – Basic OS Concepts

Concept	Description
OS	Describes platform capabilities by providing only type definitions (base for other concepts)
Radio	Send & receive functions, type definitions for node and message IDs (e.g., a node ID may be an 802.15.4 or an IPv6 address…)
Timer	Allows to schedule callbacks
Clock	Allows to read system time
Debug	printf-like logging facility, can print to UART or e.g., forward to sink
Serial	UART, I^2C, …
…	…

More concepts:
http://www.ibr.cs.tu-bs.de/users/tbaum/wiselib/doxygen/testing/html/group__concepts.html

Image Source: Dissertation Tobias Baumgartner, 2012

3. Architecture – Algorithm Example

```
namespace wiselib
{
   template<typename OsModel_P,
            typename RoutingTable_P,
            typename Radio_P = typename OsModel_P::Radio,
            typename Timer_P = typename OsModel_P::Timer,
            typename Debug_P = typename OsModel_P::Debug>
   class DsdvRouting
      : public RoutingBase<OsModel_P, Radio_P>
   {
     public:
      typedef OsModel_P OsModel;
      typedef Radio_P Radio;
      typedef Timer_P Timer;
      typedef Debug_P Debug;
      typedef RoutingTable_P RoutingTable;
      // ...
      int init( Radio& radio, Timer& timer, Debug& debug )
      {
         radio_ = &radio;
         timer_ = &timer;
         debug_ = &debug;
         return SUCCESS;
      }
     private:
      RoutingTable routing_table_;
      // ....
   };
}
```

OS Abstraction Layer

Data Structure

Parameterized Algorithm

Use of typedefs to have a fixed name to reference

static allocation

3. Architecture – Algorithm Example

Instantiated template class (when compiling for iSense)

```
namespace wiselib
{
    class DsdvRouting
        : public RoutingBase<iSenseOsModel, iSenseRadioModel>
    {
        typedef iSenseOsModel OsModel;
        typedef iSenseRadioModel Radio;
        typedef iSenseTimerModel Timer;
        typedef iSenseDebug Debug;
        typedef wiselib::StaticArrayRoutingTable<OsModel, Radio, 8, /* ...*/ > RoutingTable;
        // ...
        int init( iSenseRadioModel& radio, iSenseTimerModel& timer, iSenseDebug& debug )
        {
            radio_ = &radio;
            timer_ = &timer;
            debug_ = &debug;
            return SUCCESS;
        }
        RoutingTable routing_table_;
        // ....
    };
}
```

Use of typedefs to have a fixed name to reference

(probably unprecise and/or partially incorrect, but you should get the idea)

Programming Heterogeneous IoT Platforms – The Wiselib

4. USAGE SCENARIOS

4. Usage Scenarios

The Wiselib can be used in two ways:

1. As an algorithm library
   – Use classes for special purposes, e.g., routing
   – “Embed” Wiselib in your host application

2. As a “generic application”
   – Can be compiled to each supported platform (portable)
   – Limited to Wiselib concepts
     (no direct access to OS functionality)

4. Usage Scenarios – Algorithm Library

```
typedef wiselib::iSenseOsModel Os;
typedef wiselib::StaticArrayRoutingTable<Os, Os::Radio, 8, wiselib::DsdvRoutingTableVa
   DsdvRoutingTable;
typedef wiselib::DsdvRouting<Os, DsdvRoutingTable> dsdv_routing_t;

class iSenseDemoApplication :
   public isense::Application,
   public isense::Task
{
public:
  // ...
private:
  dsdv_routing_t routing_;
};

iSenseDemoApplication::iSenseDemoApplication(isense::Os& os) : isense::Application(os)
{
   routing_.init( os, os, os );
}

// ...

void iSenseDemoApplication::boot(void)
{
   os_.debug("WiselibExample::boot");
   routing_.enable_radio();
   // ...
}

isense::Application* application_factory(isense::Os& os)
{
   return new iSenseDemoApplication(os);
}
```

OS Abstraction Layer
Data Structure
Algorithm

Radio, Timer, Debug concepts all modeled by iSenseOsModel (using multiple inheritance)

Example from $WISELIB_HOME/apps/iapps/wiselib_example

4. Usage Scenarios – Generic App

```
typedef wiselib::OSMODEL Os;
typedef wiselib::StaticArrayRoutingTable<Os, Os::Radio, 8,
      wiselib::DsdvRoutingTableValue<Os, Os::Radio> > RoutingTable;
typedef wiselib::DsdvRouting<Os, RoutingTable> DsdvRouting;

class DsdvRoutingApplication
{
   public:
      void init( Os::AppMainParameter& value )
      {
         radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet( value );
         timer_ = &wiselib::FacetProvider<Os, Os::Timer>::get_facet( value );
         debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet( value );

         routing_.init( *radio_, *timer_, *debug_ );
         // ...
      }
      // ...
   private:
      DsdvRouting routing_;
      Os::Radio::self_pointer_t radio_;
      Os::Timer::self_pointer_t timer_;
      Os::Debug::self_pointer_t debug_;
};

wiselib::WiselibApplication<Os, DsdvRoutingApplication> routing_app;

void application_main( Os::AppMainParameter& value )
{
   routing_app.init( value );
}
```

OS Abstraction Layer
Data Structure
Algorithm

FacetProvider hides initialization details

Reference to internal typedefs makes code more readable

References to typedefs declared in OS models

static allocation

Example from $WISELIB_HOME/apps/generic_apps/routing_test

4. Usage Scenarios – Compiling

```
# ----------------------------------------
# Environment variable WISELIB_PATH needed
# ----------------------------------------

# all: shawn
# all: scw_msb
# all: contiki_msb
# all: contiki_sky
# all: contiki_micaz
all: isense
# all: tinyos-tossim
# all: tinyos-micaz

export APP_SRC=routing_test.cpp
export BIN_OUT=routing_test

include ../Makefile
```

Makefile from $WISELIB_HOME/apps/generic_apps/routing_test

Programming Heterogeneous IoT Platforms – The Wiselib

5. CALLBACK MECHANISM

5. Callback Mechanism

Some concepts require to register callbacks (e.g., radio, timer)…

```
class ExampleApplication
{
    public:
        void init( Os::AppMainParameter& value )
        {
            radio_ = &wiselib::FacetProvider<Os, Os::Radio>::get_facet( value );
            timer_ = &wiselib::FacetProvider<Os, Os::Timer>::get_facet( value );
            debug_ = &wiselib::FacetProvider<Os, Os::Debug>::get_facet( value );
            radio_->enable_radio(1);
            radio_->reg_recv_callback<
                ExampleApplication,
                &ExampleApplication::receive_radio_message >( this );
            timer_->set_timer<
                ExampleApplication,
                &ExampleApplication::start >( 5000, this, 0 );
        }
        void receive_radio_message( Os::Radio::node_id_t from,
                                    Os::Radio::size_t len,
                                    Os::Radio::block_data_t *buf )
        {
            // ...
        }
        void start( void* )
        {
            // ...
        }
    private:
        // ...
};
```

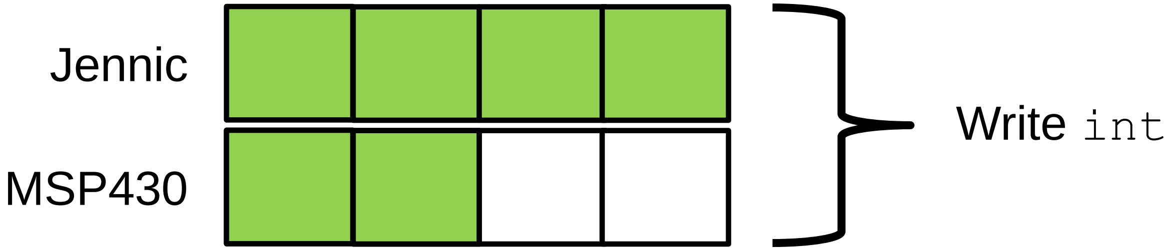
Makefile from $WISELIB_HOME/apps/generic_apps/example_app

Programming Heterogeneous IoT Platforms – The Wiselib

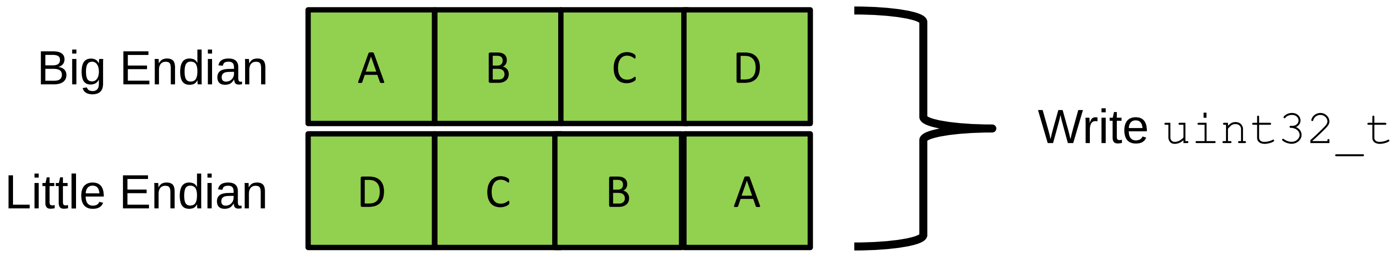
6. MESSAGE SERIALIZATION

6. Message Serialization

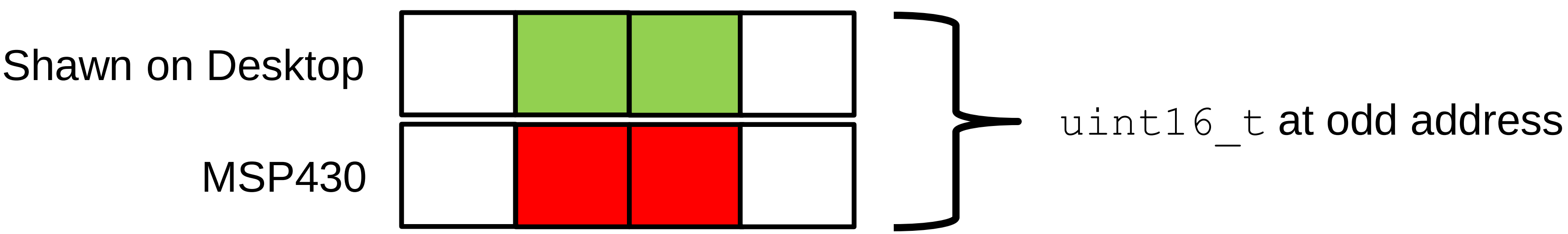
1. Word Width



2. Byte Order



3. Alignment



257

6. Message Serialization

Solution: Templated Serialization provided by Wiselib

– Can be specialized for **each system** and **each data type**

```
template<typename OsModel_P,
         typename BlockData_P,
         typename Type_P>
inline Type_P read( BlockData_P *target )
{ return Serialization<OsModel_P, BlockData_P, Type_P>::read( target ); }

template<typename OsModel_P,
         typename BlockData_P,
         typename Type_P>
inline typename OsModel_P::size_t write( BlockData_P *target, Type_P& value )
{ return Serialization<OsModel_P, BlockData_P, Type_P>::write( target, value ); }
```

6. Message Serialization

```
namespace wiselib
{
  template<typename OsModel_P, typename Radio_P>
  class DsdvRoutingMessage
  {
    private:
      uint8_t buffer[Radio::MAX_MESSAGE_LENGTH];
    public:
      typedef OsModel_P OsModel;
      typedef Radio_P Radio;
      typedef typename Radio::block_data_t block_data_t;
      typedef typename Radio::node_id_t node_id_t;
      typedef typename Radio::message_id_t message_id_t;

      inline message_id_t msg_id()
      { return read<OsModel, block_data_t, message_id_t>( buffer ); };

      inline void set_msg_id( message_id_t id )
      { write<OsModel, block_data_t, message_id_t>( buffer, id ); }

      inline node_id_t source()
      { return read<OsModel, block_data_t, node_id_t>(buffer + SOURCE_POS); }

      inline void set_source( node_id_t src )
      { write<OsModel, block_data_t, node_id_t>(buffer + SOURCE_POS, src); }

      // ...

      enum data_positions
      {
         MSG_ID_POS   = 0,
         SOURCE_POS   = sizeof(message_id_t),
         // ...
      };
   };
}
```

Programming Heterogeneous IoT Platforms – The Wiselib

7. PROGRAMMING ENVIRONMENTS

8. Programming Environments

- **Wisebender** is an Online-IDE for Wiselib-Development
  - Fork of the popular (Arduino) Codebender IDE http://codebender.cc/
  - Currently being developed in GSoC 2013
  - Edit and compile code without *any* toolchain installation (!!!)

- Wisebender Beta Access: http://wisebender.cti.gr/app_dev.php/

8. Programming Environments

- **The WISEBED VM** is a pre-configured Linux Development Environment containing:
  - Compilers for various IoT hardware platforms
  - Various sensor node operating systems
  - The Shawn network simulator
  - The Wiselib ☺

- Get it from: wisebed.eu/site/application-development/virtual-machine/
- Or from the USB sticks I brought!

Programming Heterogeneous IoT Platforms – The Wiselib

8. SUMMARY & RESOURCES

8. Summary

- Code library of algorithms
  - Write once, compile everywhere
  - Highly efficient, portable libraries and applications
  - Variety of well-known algorithms implemented
  - Open-Source
- Supports a variety of hardware platforms and simulators
  - Easily extensible, e.g., to other (C/C++/nesC) platforms
  - Helps to avoid vendor lock-in
- Growing community of contributors
  - Benefit from others, contribute for others

8. Resources

```
bimschas@opium [~/wiselib] (master) $ tree -d -L 2
.
├── apps
│   ├── arduino_apps
│   ├── generic_apps
│   ├── iapps
│   ├── ns3
│   ├── pc_apps
│   └── shawn_apps
├── doc
│   ├── concepts
│   └── pages
├── util
│   ├── compile
│   ├── isense
│   ├── pyvisor
│   ├── rdf
│   └── spyglass
├── wiselib.stable
│   ├── algorithms
│   ├── external_interface
│   ├── internal_interface
│   └── util
└── wiselib.testing
    ├── algorithms
    ├── external_interface
    ├── intermediate
    ├── internal_interface
    ├── radio
    └── util
```

Lots of Sources & Examples in Repo

git clone git@github.com:ibr-alg/wiselib.git

Read Documentation

https://github.com/ibr-alg/wiselib/wiki

Join Mailing List

wiselib-subscribe@wiselib.org

Apply for GSoC 2014 :)

SMART SANTANDER

Experimenting with Guildford Testbed

Outline

- Development environment
  - VM installation and usage
- TinyOS programming
  - TinyOS Fundamentals
  - SmartPlogg libraries
- TMON
  - Basic Functionalities
  - Plug-in develop
- Build a Service
  - ExprDB Access
  - Export Your Data -> The “Xively” Example

Disclaimer

- You won't learn how to build a TinyOS application from scratch!!
  - TinyOS has a very steep learning curve
- but
  - You will know the basics and how to read and understand TinyOS code (requirement to build good TinyOS application)
  - You will know all the components/actions involved in experimenting with SmartSantander
- Presentation: http://tinyurl.com/nsy95uo
  - Please help me to correct/improve it

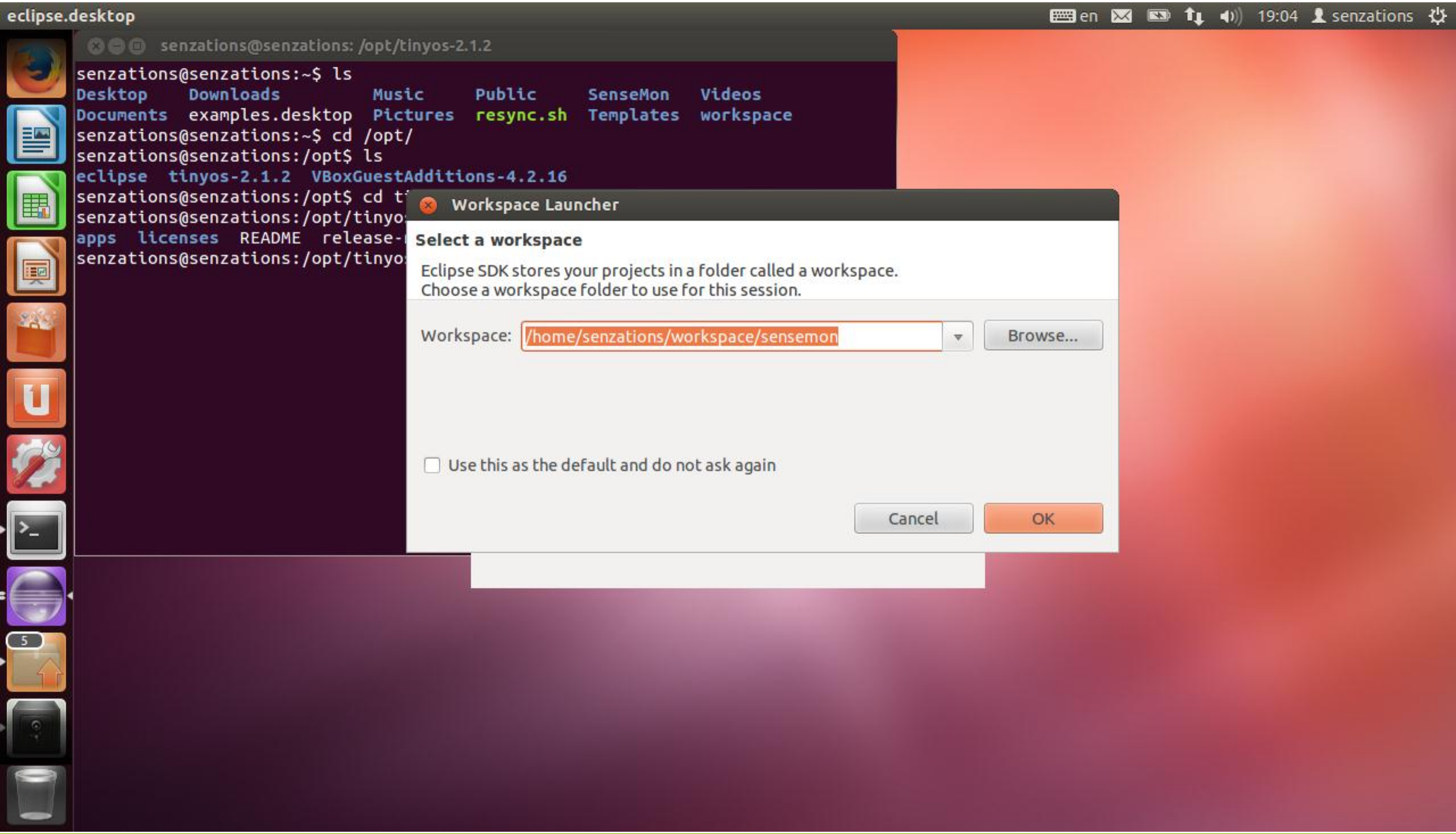
Senzations13 VM

DEVELOPMENT ENVIRONMENT

Senzations13 VM

- Install VirtualBox
- Import the Senzations13 Appliance
  - Senzations13.ova
    - Username: senzations
    - Password: senzations13
- TinyOS-2.1.1 path
  - /opt/tinyos-2.1.1
- Eclipse workspaces
  - TOS -> TinyOS Examples with YETI Plugin
  - sensemon -> sensemon source code

Senzations13 Walk-through



Fix VM synchronization issue

DONE!!!

- Guest machine
  - Devices menu (Virtua … Install Guest Additions”
  - Select and ins … uest Additions
  - Restart the … achine
- Host machine
  - Go to VirtualBox installation folder (VBoxManage)
    - VBoxManage guestproperty set Senzations13 "VirtualBox/GuestAdd/VBoxService/-timesync-interval" 1000
    - VBoxManage guestproperty set Senzations13 "VirtualBox/GuestAdd/VBoxService/-timesync-set-threshold" 1000

Basic TinyOS concepts

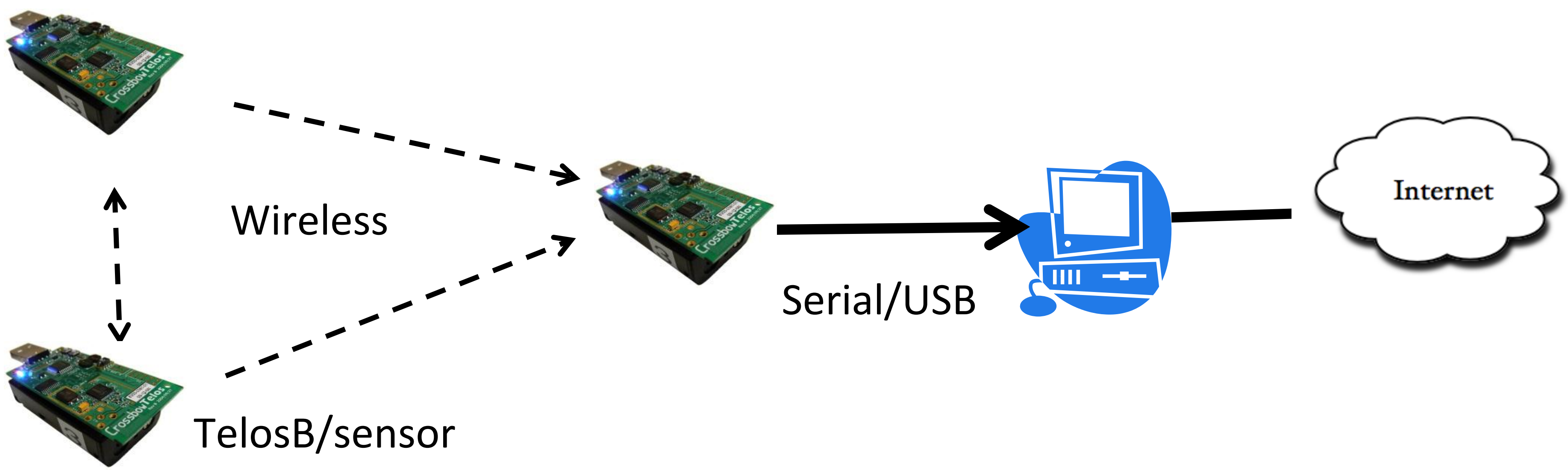
INTRODUCTION TO WSN PROGRAMMING

WSN/Testbed Architecture

Sensor code (nesC/TinyOS)

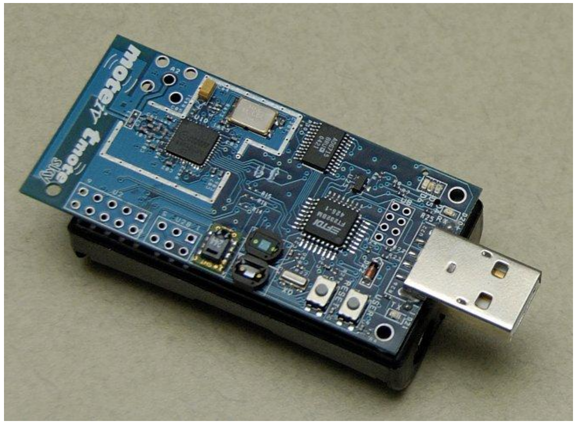
Base station code (nesC/TinyOS)

Gateway code (Java, c, …)



HW Specification

- Mote/TelosB
  - MSP430 (MCU)
    - 48KB of ROM
    - 10KB of RAM
  - CC2420 (RADIO)
    - 802.15.4, 250KB/s
  - Embedded sensors (light, temp, humidity)



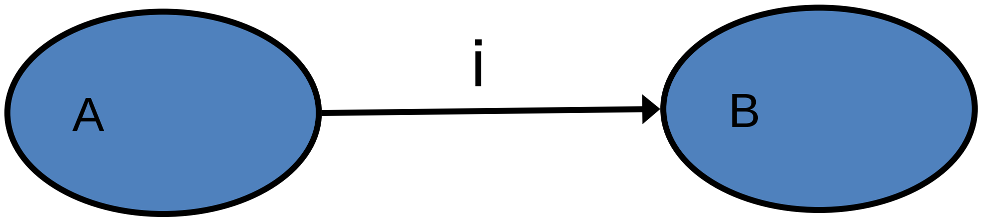
- Mote/General
  - Low memory and speed
  - Not all OS services, drivers etc can be installed

TinyOS General

- OS for low power, embedded, wireless devices
  - Typically only one task is needed
  - Component based OS (memory)
  - Application is embedded in the OS (speed)
- Documentation
  - *http://www.tinyos.net/tinyos-2.x/doc/pdf/tinyos-programming.pdf*
  - TEP (TinyOS Enhancement Proposals)

TinyOS Components

- TinyOS and its applications are in nesC
  - C dialect with extra features
- Basic unit of nesC code is a component
- Components connect via interfaces
  - Connections called “wiring”

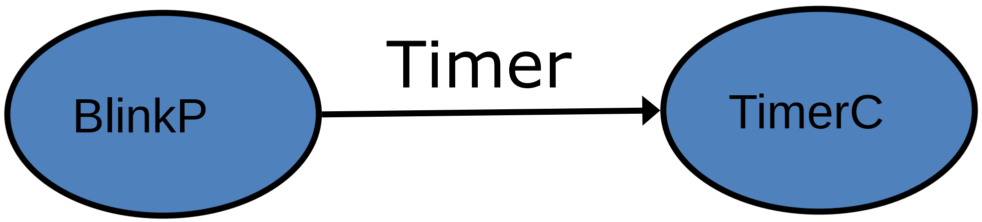


Components

- A component is a file
  - names must match
- Modules are components with
  - Variables
  - Executable code
- Configurations are components that
  - wire other components together
  - using interfaces

Component Example

- BlinkC wires BlinkP.Timer to TimerC.Timer




```
module BlinkP { … }
implementation {
  int c;
  void increment() {c++;}
}
```

```
configuration BlinkC { … }
implementation {
  components new TimerC();
  components BlinkC;

  BlinkC.Timer -> TimerC;
}
```

Singletons and Generics

- Singleton components are unique:
  - they exist in a global namespace
- Generics are instantiated:
  - each instantiation is a new, independent copy

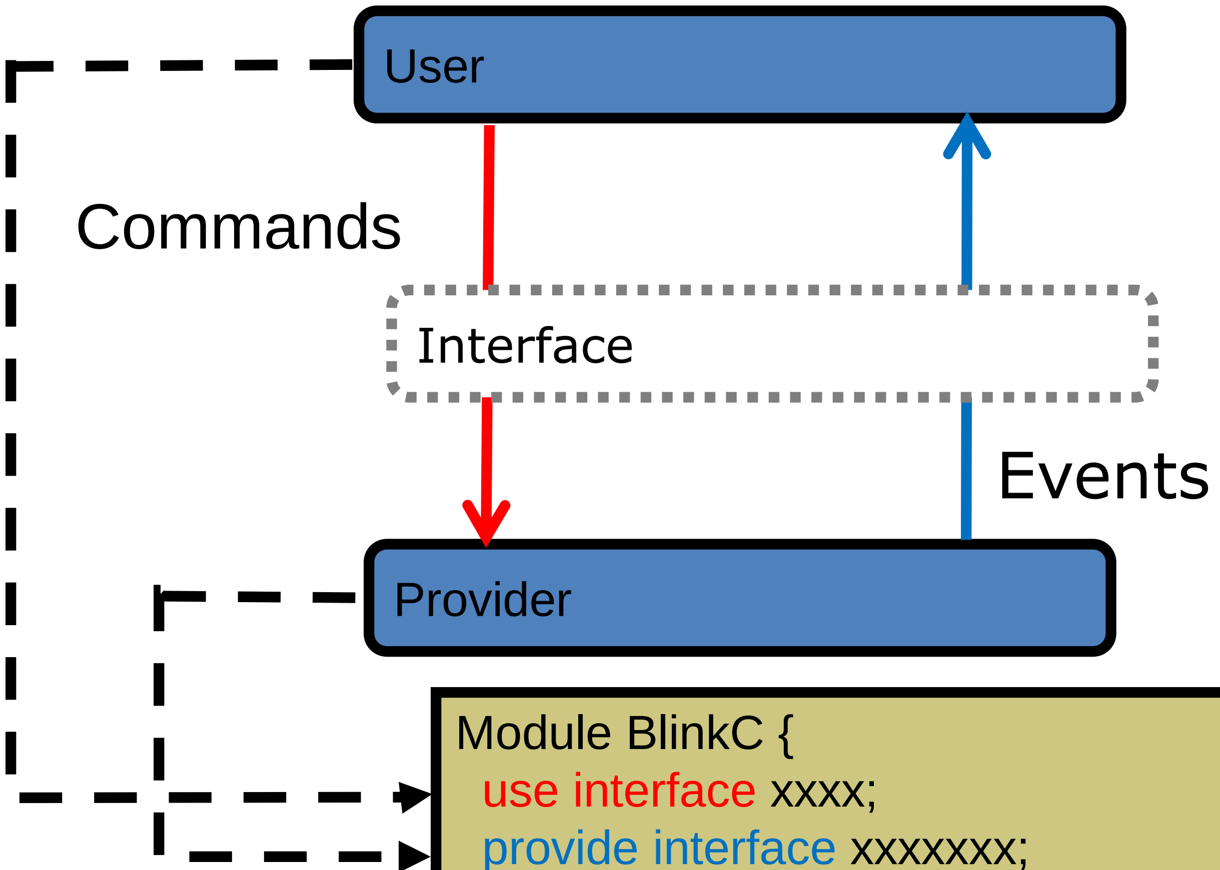
```
configuration BlinkC { … }
implementation {
  components new TimerC();
  components BlinkC;

  BlinkC.Timer -> TimerC;
}
```

Interfaces

- Collections of related functions
- Define how components connect
- Interfaces are bi-directional: for A->B
  - Commands are from A to B
  - Events are from B to A
- Can have parameters (types)

```
interface Timer<tag> {
  command void startOneShot(uint32_t period);
  command void startPeriodic(uint32_t period);
  event void fired();
}
```

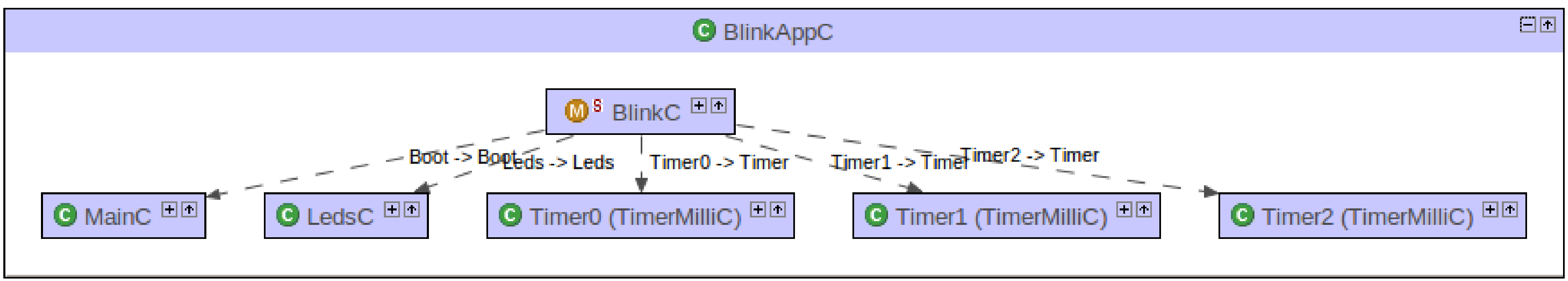
Interface (provide and use)




```
Module BlinkC {
  use interface xxxx;
  provide interface xxxxxxx;
  .........
}
```

Blink Application

- Toggle LEDs
- Update LEDs status based on a Timer
- Each LED is controlled by a different Timer



Blink Application

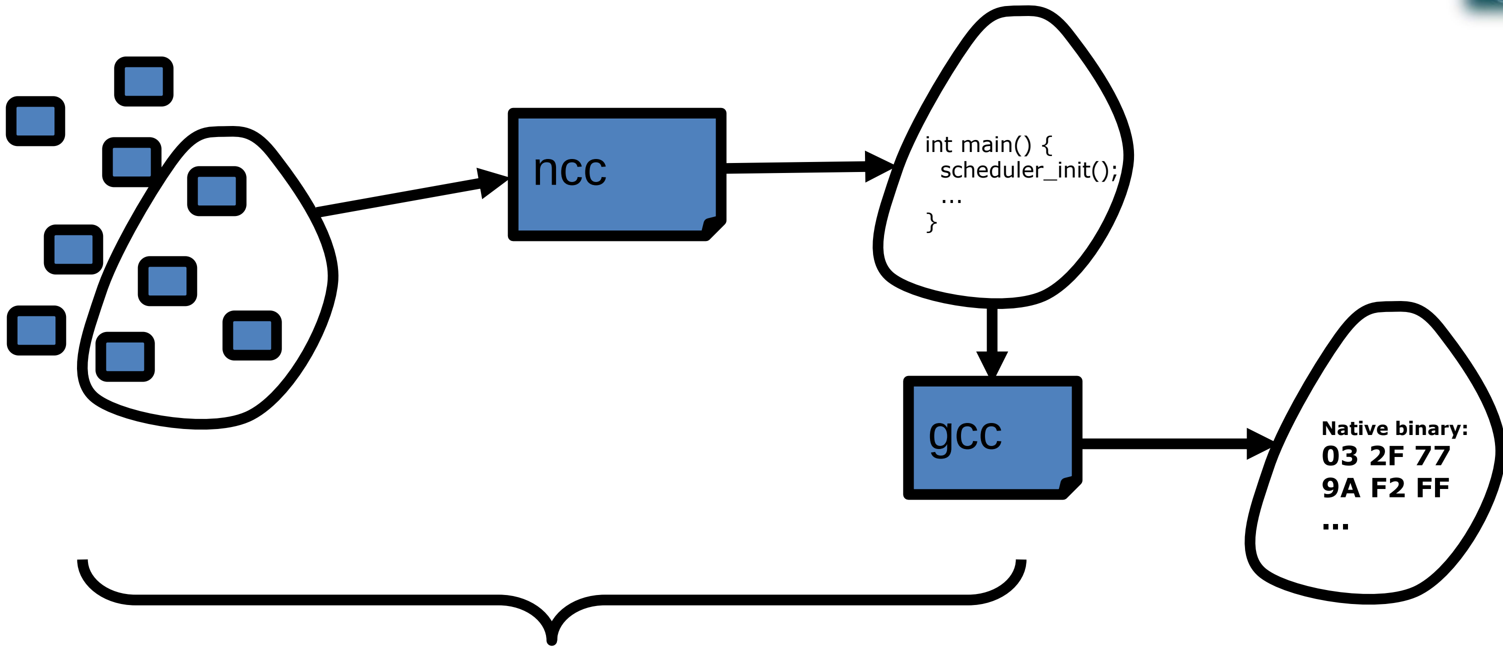
Configuration: BlinkAppC.nc

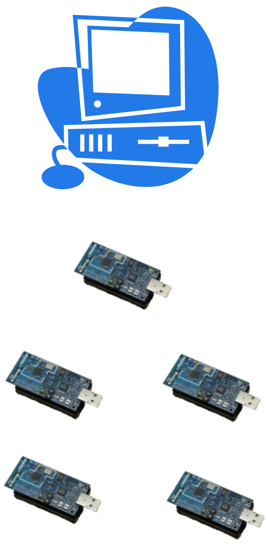
```
configuration BlinkAppC {
}
implementation {
        components MainC,
        BlinkC, LedsC;
        components new
        TimerMilliC() as Timer0;

        BlinkC.Boot > MainC.Boot;
        BlinkC.Timer0 > Timer0;
        BlinkC.Leds > LedsC;
}
```

Module: BlinkC.nc

```
module BlinkC {
        uses interface Timer<TMilli> as
        Timer0;
        uses interface Leds;
        uses interface Boot;
}
implementation {
        event void Boot.booted() {
                call
        Timer0.startPeriodic( 1000 );
        }
        event void Timer0.fired() {
                call
Leds.led0Toggle();
        }
```

Make Toolchain



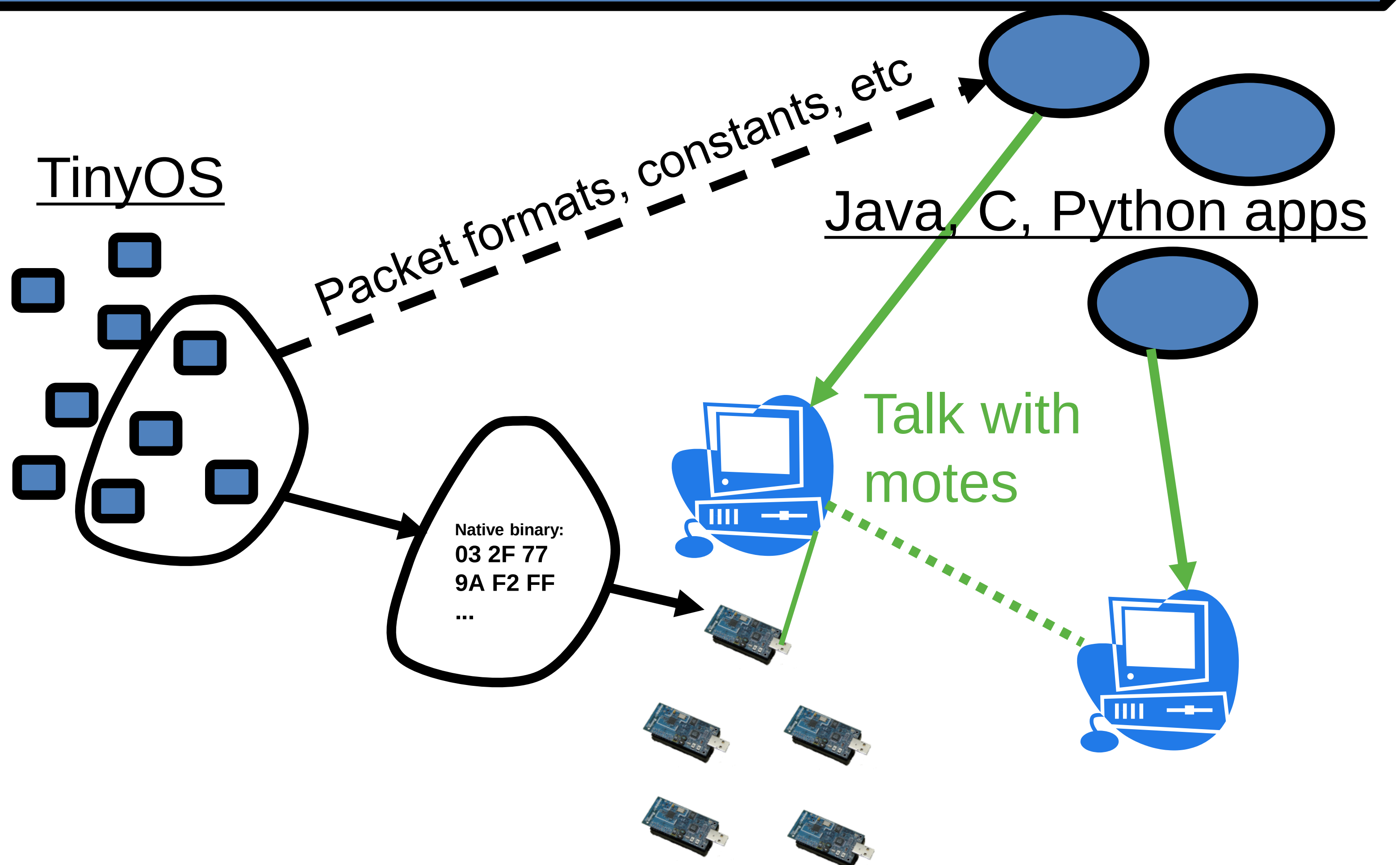


make TARGET
make TARGET **install**,ID bsl,PORT
make telosb install,X bsl,/dev/ttyS0

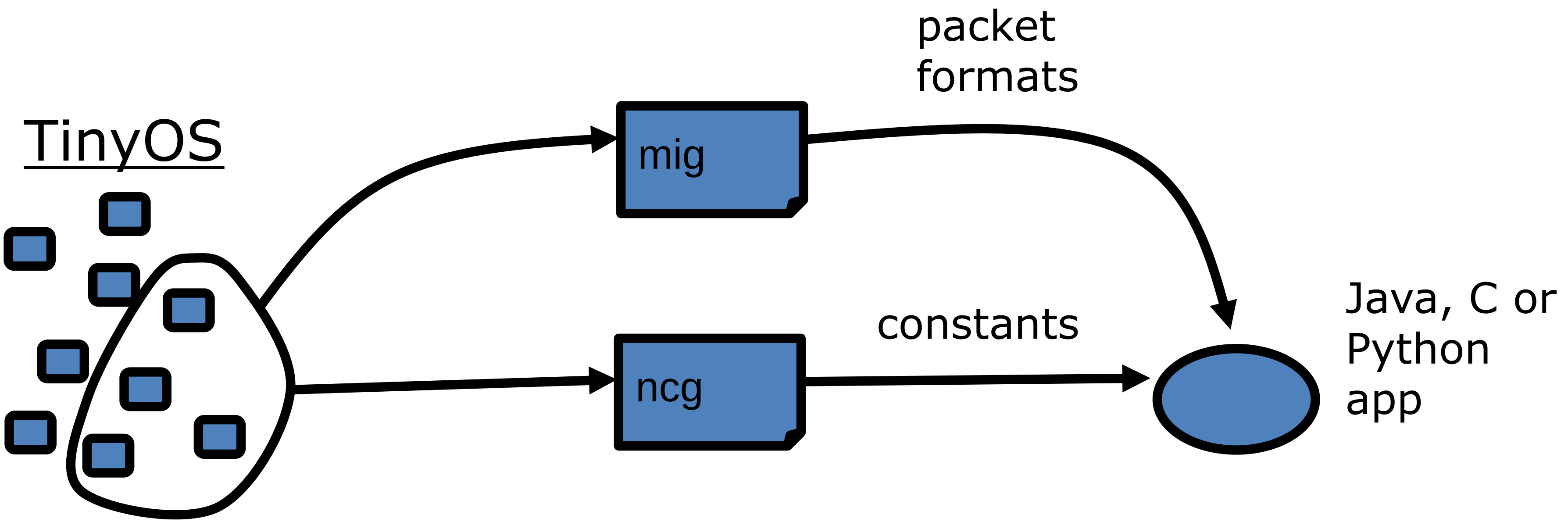
TARGET: **telosb**, tmote, **XM1000**, micaz
ID: 16 bit number (0 to 65536)
PORT: motelist

PC Application: Java

java classname -comm serial@PORT:TARGET params
net.tinyos.tools.Listen [-comm <source>] (raw packets)
net.tinyos.tools.MsgReader [-comm <source>] message-class (formatted packets)



PC Applications: MIG

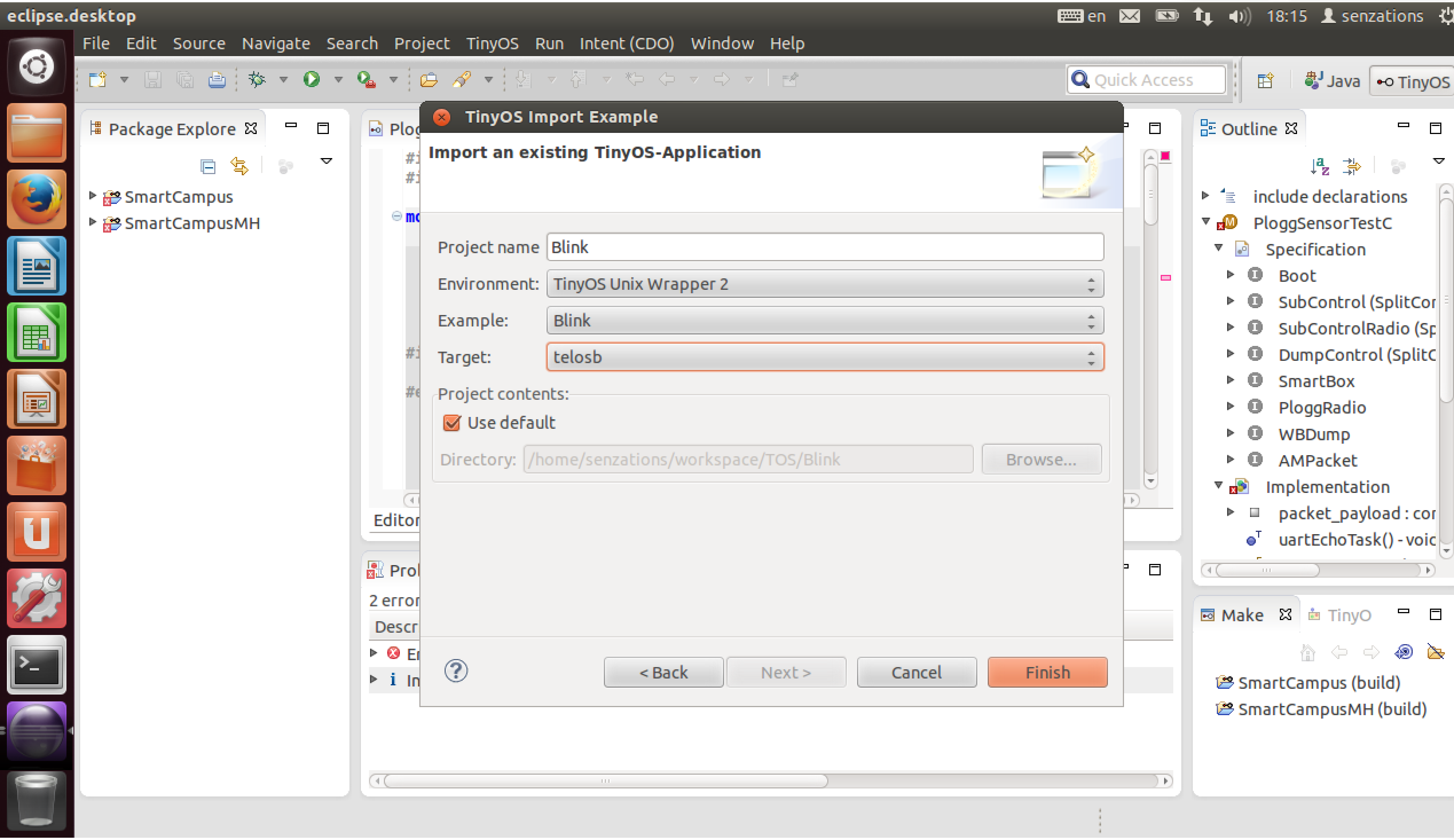


mig java -java-classname=[java_class] [header] [c_struct_name]
-o [java_class].java

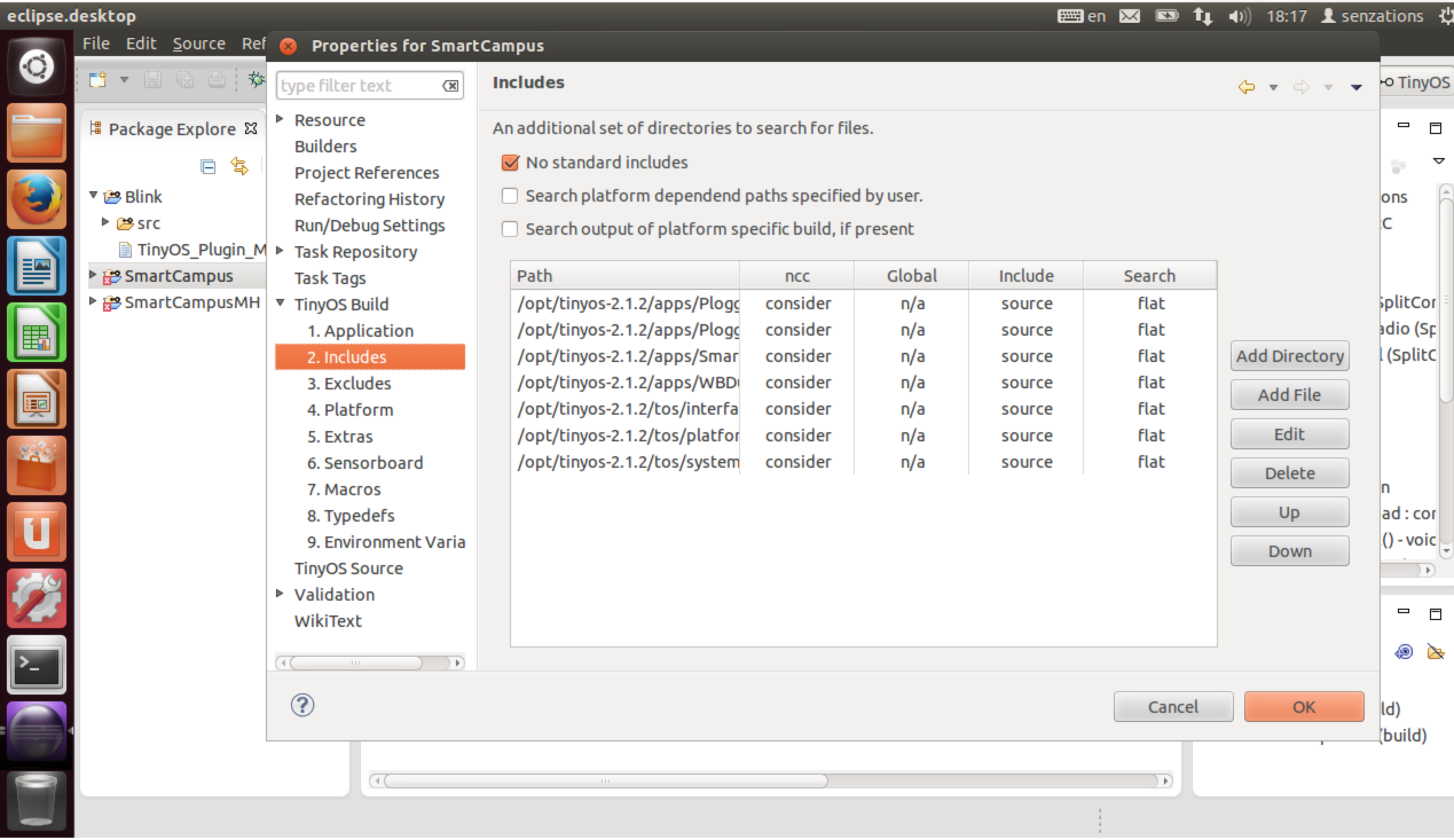
For instance:

mig java -java-classname=CountMsg CountMsg.h CountMsg
-o CountMsg.java

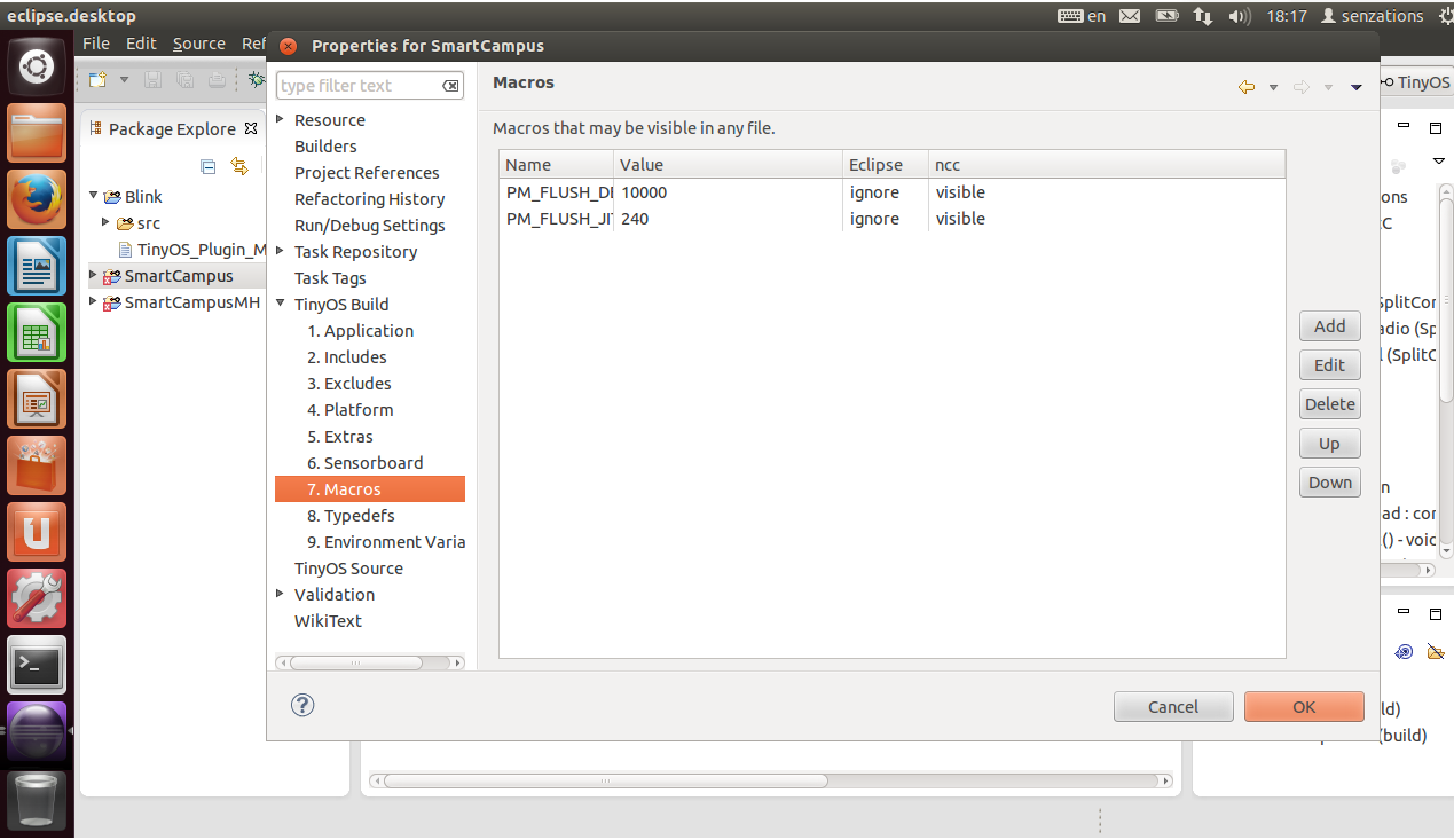
TOS and Eclipse: YETI Plug-in



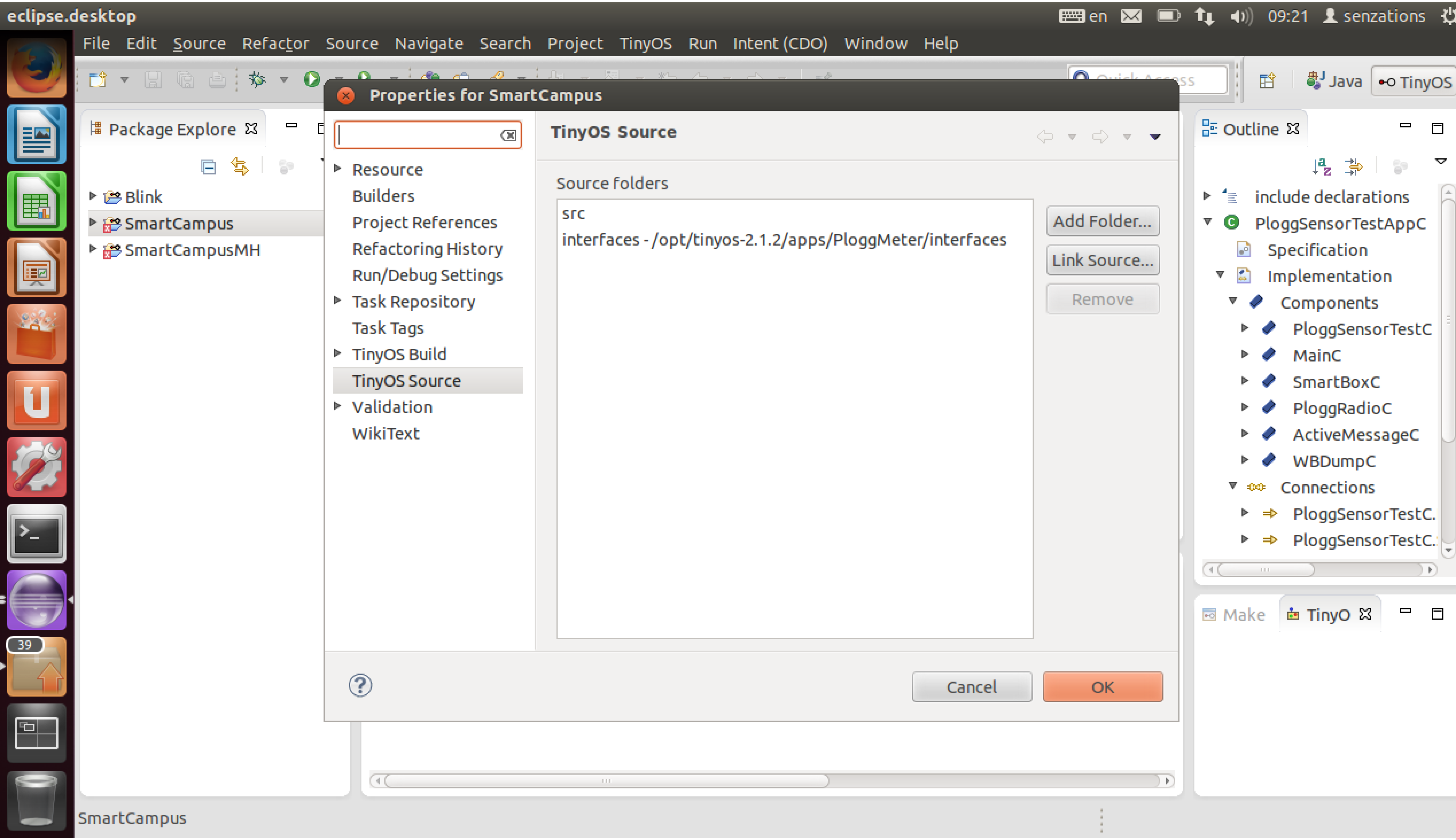
TOS and Eclipse: YETI Plug-in



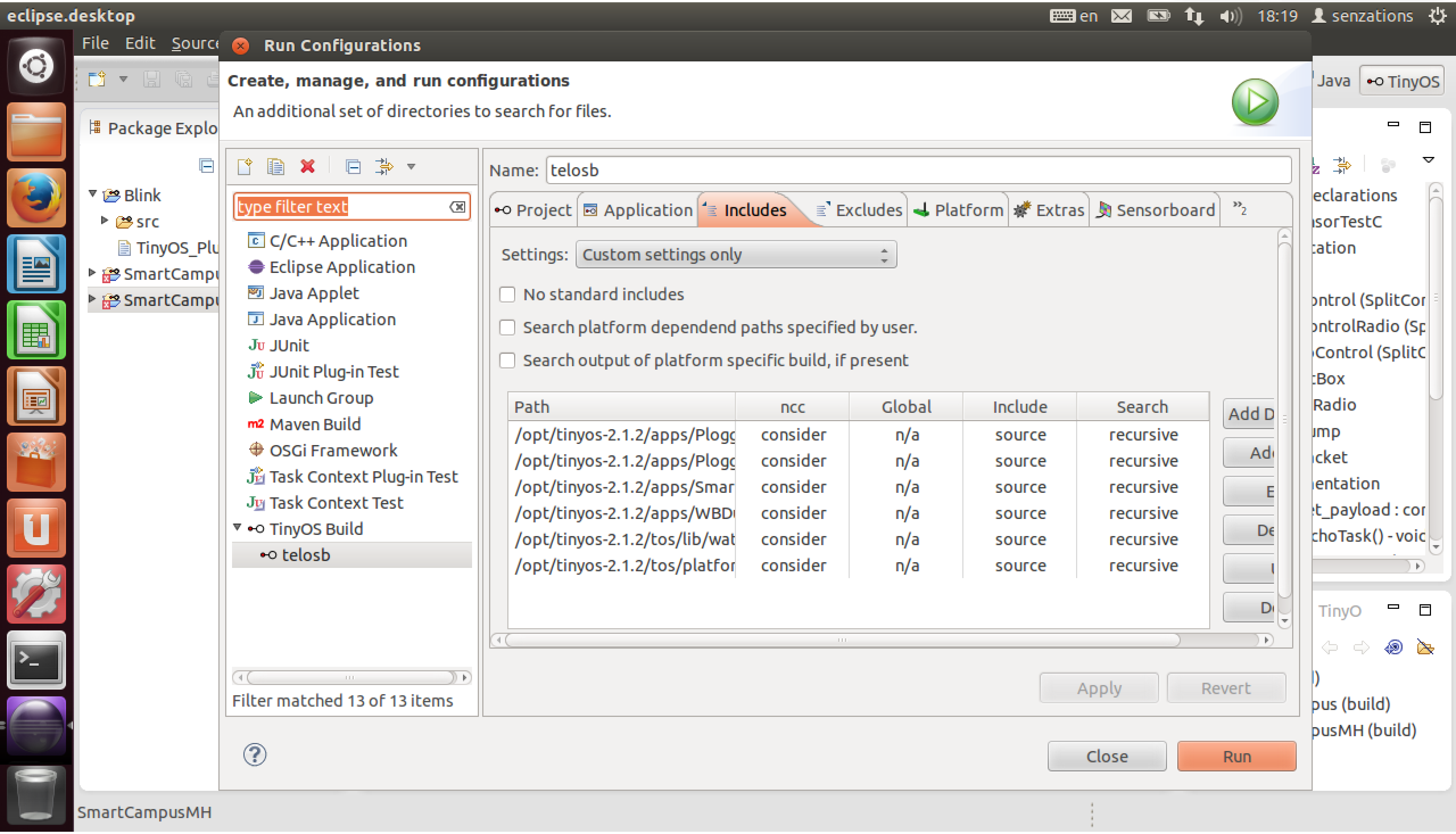
TOS and Eclipse: YETI Plug-in



TOS and Eclipse: YETI Plug-in



TOS and Eclipse: YETI Plug-in



Hans-on 1 – Blink

- Import Blink into Eclipse TOS workspace
- Compile Blink
  - Using YETI
  - Via CMD line
- Install Blink
  - Via CMD line

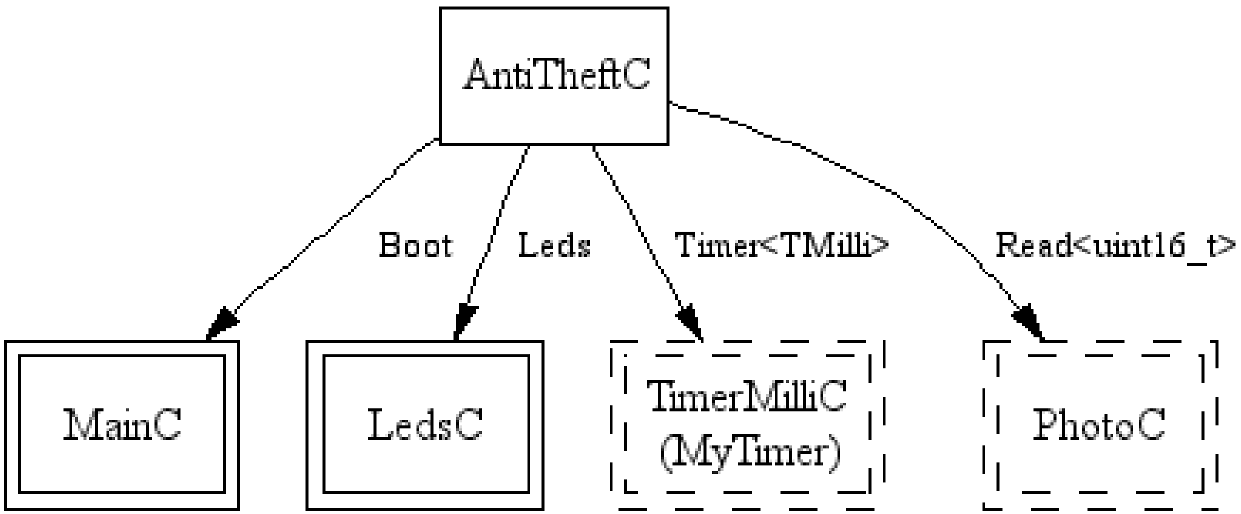
Sensing and Tasks

ADVANCED TOPICS

AntiTheft Application

- Goal: write an anti-theft device
- Two parts
  - Detecting theft
    - Assumption: thieves put the motes in their pockets
    - So, a "dark" mote is a stolen mote
    - Every N ms check if light sensor is below some threshold
  - Reporting theft
    - Assume: bright flashing lights deter thieves
    - Algorithm: light the **red LED** for a little while!

AntiTheft – Components



AntiTheft – Module

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<TMilli> as Check;
  uses interface Read<uint16_t>;
}
implementa
  event voi
    call Che
  }
  event voi
    call Rea
  }
  event void Read.readDone(error_t
    if (ok == SUCCESS && val < 200)
      theftLed();
  }
}
```

```
interface Boot {
  /* Signaled when OS booted */
  event void booted();
}
```

```
interface Timer<tag> {
  command void startOneShot(uint32_t period);
  command void startPeriodic(uint32_t period);
  event void fired();
}
```

Components start with a *signature* specifying

- the interfaces *provided* by the component
- the interfaces *used* by the component

A module is a component implemented in C

- with functions implementing commands and events
- and extensions to call commands, events

AntiTheft – Split-Phase

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<TMilli> as Check;
  uses interface Read<uint16_t>;
}
implementation {
  event void Boot.booted() {
    call Check.startPeriodic(1000);
  }
  event void Check.fired() {
    call Read.read();
  }
  event void Read.readDone(error_t ok, uint16_t val) {
    if (ok == SUCCESS && val < 200)
      theftLed();
  }
}
```

In TinyOS, all long-running operations are split-phase:

- A command starts the op: read
- An event signals op completion: readDone

```
interface Read<val_t> {
  command error_t read();
  event void readDone(error_t ok, val_t val);
}
```

AntiTheft – Split-Phase

```
module AntiTheftC {
  uses interface Boot;
  uses interface Timer<TMilli> as Check;
  uses interface Read<uint16_t>;
}
implementation {
  event void Boot.booted() {
    call Check.startPeriodic(1000);
  }
  event void Check.fired() {
    call Read.read();
  }
  event void Read.readDone(error_t ok, uint16_t val) {
    if (ok == SUCCESS && val < 200)
      theftLed();
  }
}
```

In TinyOS, all long-running operations are split-phase:

- A command starts the op: read
- An event signals op completion: readDone

Errors are signalled using the error_t type, typically

- Commands only allow one outstanding request
- Events report any problems occurring in the op

```
interface Read<val_t> {
  command error_t read();
  event void readDone(error_t ok, val_t val);
}
```

AntiTheft – Configuration

```
configuration AntiThe
implementation
{
  components An

  AntiTheftC.Boc
  AntiTheftC.Led

  components new TimerMilliC() as MyTimer;
  AntiTheftC.Check -> MyTimer;

  components new HamamatsuS1087ParC() as PhotoC();
  AntiTheftC.Read -> PhotoC;
}
```

```
generic configuration TimerMilliC() {
  provides interface Timer<TMilli>;
}
```

```
generic configuration HamamatsuS1087ParC() {
  provides interface Read;
}
implementation { ... }
```

A configuration is a component built out of other components.
It *wires* “used” to “provided” interfaces.
It can instantiate *generic* components
It can itself provide and use interfaces

AntiTheft – Improved

- Goal: avoid false positive in the theft detection
- Two parts
  - Detecting theft
    - Assumption: thieves could put the motes out of their pockets for a while
    - So, a "dark" mote is not always a stolen mote
    - Every N samples check if the light samples variance is below some threshold
  - Reporting theft
    - Assume: bright flashing lights deter thieves
    - Algorithm: light the **red LED** for a little while!

AntiTheft – Tasks

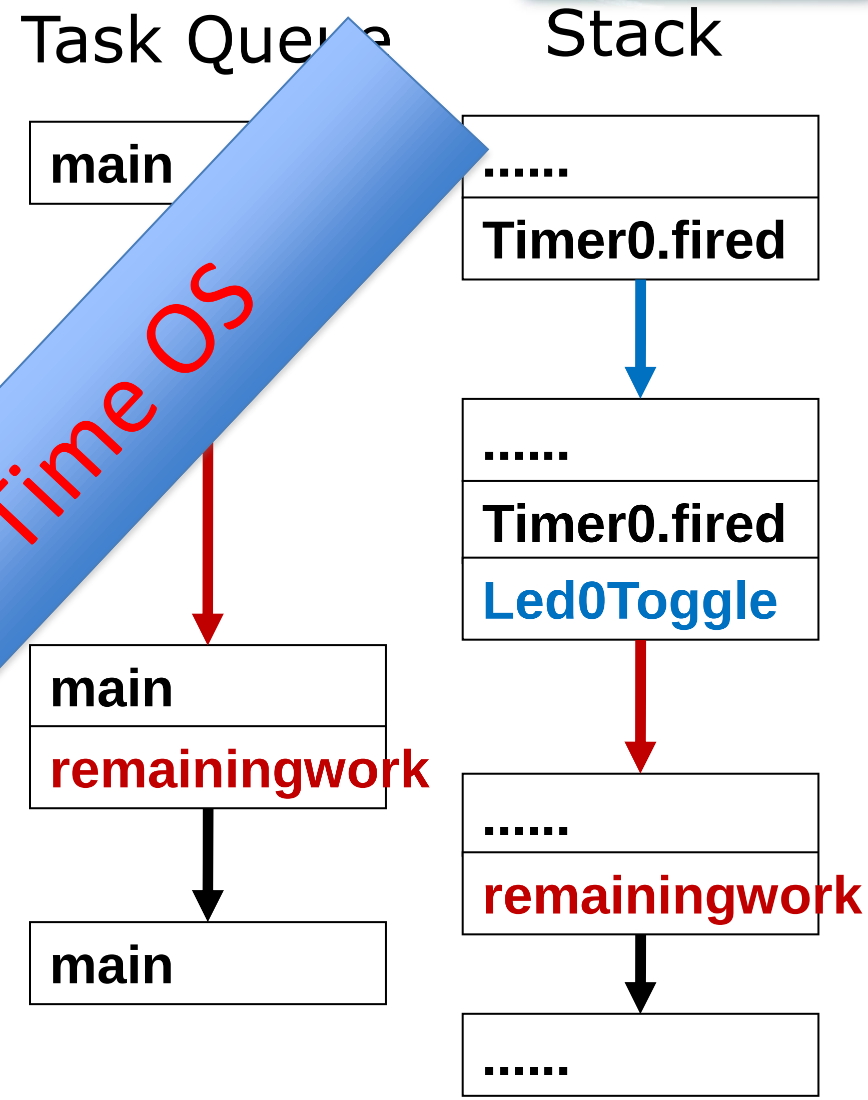
```
uint16_t lightSamples[SAMPLES];
uint8_t numSamples;
event void Read.readDone(error_t ok, uint16_t val) {
  if (ok == SUCCESS) {
     if (numSamples < NUM_SAMPLES) {
         lightSamples[numSamples] = val;
         numSample ++;
     } else {
         post checkVariance();
     }
}
task void checkAcceleration() {
   uint16_t i, avg, var;

   for (avg = 0, i = 0; i < SAMPLES; i++)
     avg += lightSamples[i];
   avg /= NUM_SAMPLES;
   for (var = 0, i = 0; i < NUM_SAMPLES; i++) {
       int16_t diff = lightSamples[i] - avg;
       var += diff * diff;
     }
   if (var < SOME_THR) theftLed();
}
```

Tasks

- TinyOS has a single stack
  - long-running computation can reduce responsiveness
- Tasks: mechanism to defer computation
  - Tells TinyOS "do this later"
- Tasks run to completion
  - TinyOS scheduler runs them one by one in the order they post
  - Must be short!
- Interrupts run on stack, can post tasks

TinyOS Execution Model

```
Xxxxxx;
event void Timer0.fired()
  {
     xxxxxx;
     xxxxxx;
     xxxxxx;
     xxxxxx;
     call Leds.led0Toggle();
     xxxxxx;
     xxxxxx;
     post remainingwork(
  }
  xxxxx;
  remainingwork(){
  xxxxx;
```



TinyOS/nesC Summary

- Components and Interfaces
  - Programs built by writing and wiring components
    - modules are components implemented in C
    - configurations are components written by assembling other components
- Execution model
  - Execution happens in a series of tasks (atomic with respect to each other) and interrupt handlers
  - No threads
- System services: startup, timing, sensing
  - (Mostly) represented by instantiable generic components
    - This instantiation happens at compile-time! (think C++ templates)
  - All slow system requests are split-phase

Where are we…

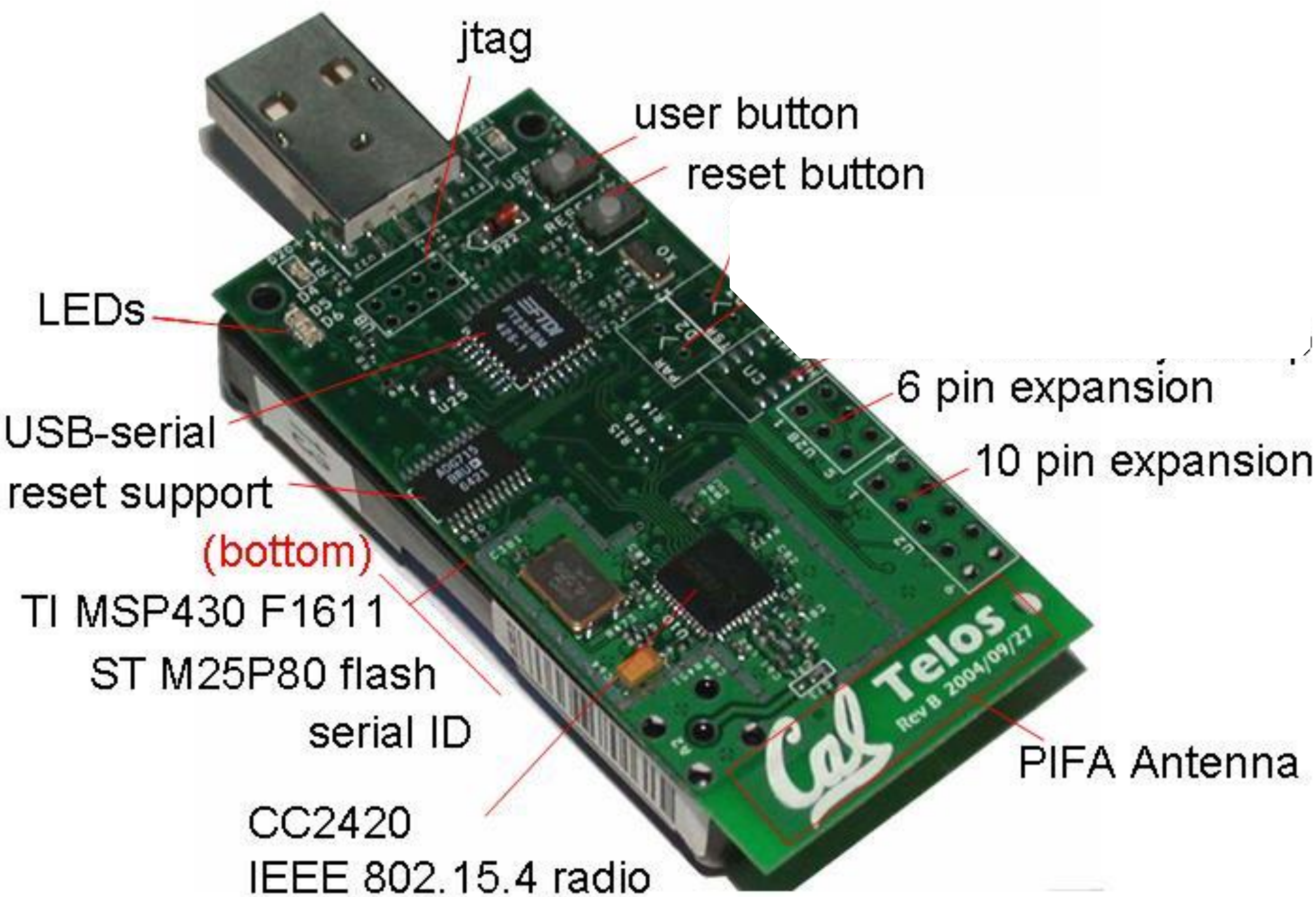
- We learnt a bit of TinyOS
  - Components and interface
  - Split-phase paradigm
  - Tasks
  - Execution model
- There is a lot missing
  - Concurrency model
  - Serial communication
  - Radio access/communication
- What we want to do
  - Run experiment on SmartSantander testbeds
  - Exploit additional HW
  - TinyOS components based structure can help

SmartPlogg TOS libraries

TOS APPLICATION FOR SMARTSANTANDER TESTBED

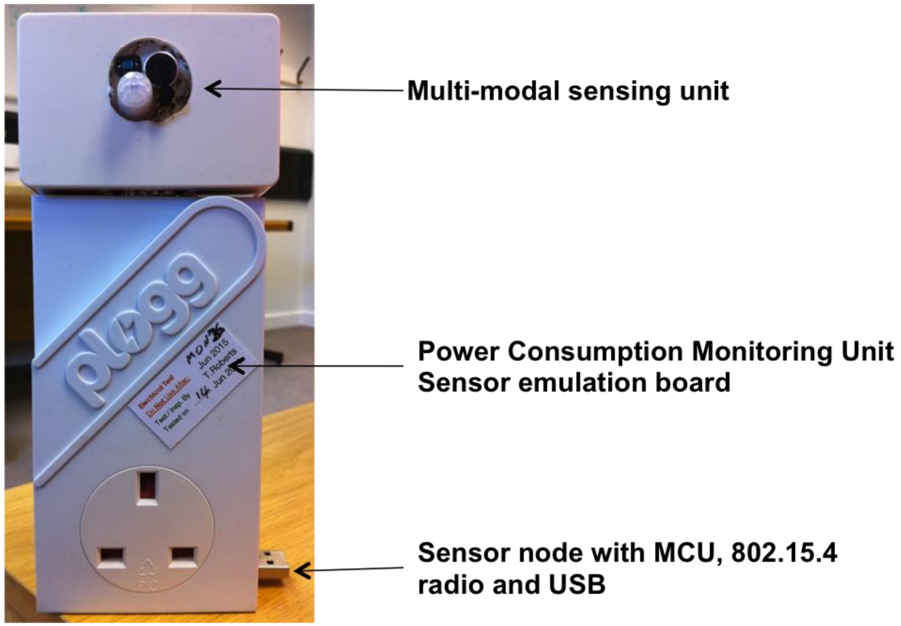
Testbed HW - Motes

- TelosB mote platform (≈ 200)
  - 16 bit MCU, 802.15.4 radio in 2.4GHz ISM band
  - Max transmit power is 1mW (Wifi cards ~100mW, Mobiles up to 1W)



Testbed HW - SmartPlogg

- Plogg+TelosB+Sensor board platform (≈ 200)
- Plogg energy meter
  - Monitoring: power, current, ...
  - Basic actuation (on/off)
  - Max sampling rate 1 sample every 2 sec
- Sensor board
  - Temperature, Light, Noise level, PIR, Vibration
  - LED for signaling
- Driver in
  - $TOSROOT/tos/platforms/telosa/chips/a500



Tutorial HW

- Similar architecture
  - Same sensors
  - Better MCU (116KB ROM)
  - Not all the HW is available in TinyOS
    - internal USART
  - Cannot run SmartPlogg code
- Can compile with its own target: XM1000

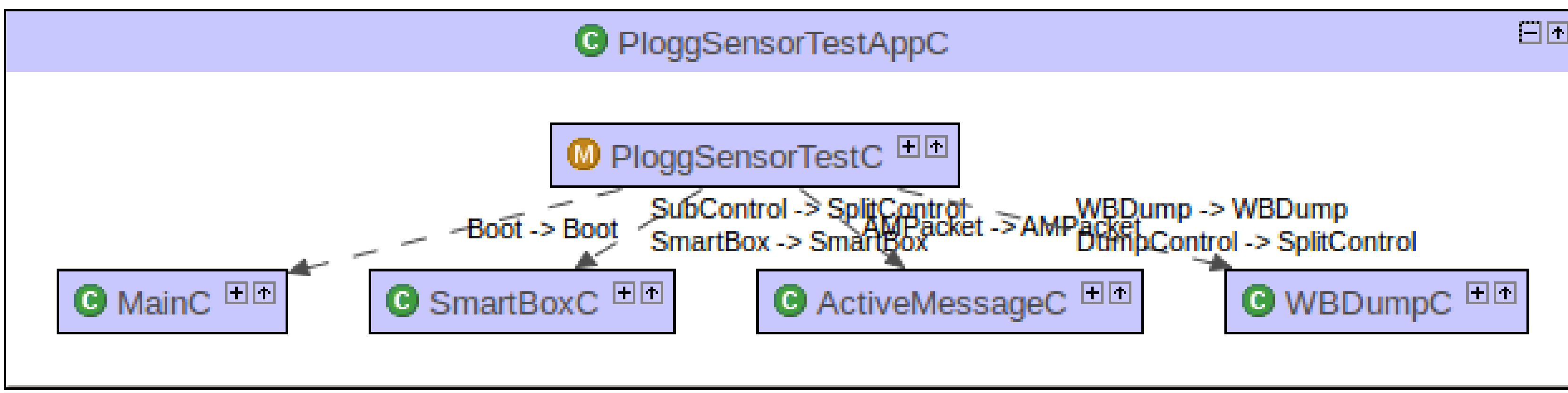
✖ For this tutorial “No Access to Energy Data”

Code Structure

- $TOSROOT/apps/
  - PloggDevelop → Application for periodic readings over serial
  - PloggDevelopMH → Application for periodic readings over radio
  - SmartBox → High-level component for sensing subsystem access
  - WBDump → Component for sending TBR formatted packets
  - PloggRadio → High-level component for radio subsystem access
  - PloggMeter → Low-level driver for energy meter access

Send Readings over the Serial

- Read the SmartPlogg sensors
- Write readings on Serial
  - Eventually transmit them to the backend (using TBR)
- PloggSensorTestAppC



SmartBox Interface

```
interface SmartBox {
        command void ledToggle();
        command void ledOn();
        command void ledOff();
        command void
        enableOffTimers(uint8_t on);
        command void
        setSamplingPeriod(uint16_t
period);
        event void
        samplesReady(context_msg_t*
payload);
}
```

- Access to LED
- Enable Plogg Timers for on/off actuation
- Sensors and Plogg sampling in split-phase mode

WBDump Interface

```
interface WBDump {
        command error_t     put(uint8_t type, uint8_t am_id, uint8_t len, void * payload);
        command uint8_t     isFlushing();
        async command error_t flush();
        async command error_t stopFlush();
}
```

- put command to enqueue messages for serial transmission
  - Works for different message types
  - Receives message payload
  - Automatic flush
- async command can be preempted

PloggSensorTestC – Module

```
event void Boot.booted() {
    call SubControl.start();
}
```

- Trigger periodic read from sensors and energy meter

```
event void SmartBox.samplesReady(context_msg_t* payload) {
        memcpy(&packet_payload, payload, sizeof(context_msg_t));
        post uartEchoTask();
}
```

- Event is generated when readings are ready

```
task void uartEchoTask() {
    if (call WBDump.put(10,
        AM_CONTEXT_MSG,
        sizeof(context_msg_t),
        &packet_payload) !=
        SUCCESS)
        post uartEchoTask();
}
```

- Copy of received packet becomes the payload of the serial message
  - 1 packet buffer
- put fails if
  - buffer is full
  - payload exceeds the maximum length
- Only task per type at time

ContextMsgWB

ContextDataWB.h

```
typedef nx_struct context_msg {

    nx_uint8_t     PIR;
    nx_uint8_t     vib;
    nx_uint16_t    mic;
    nx_uint16_t    temp;
    nx_uint16_t    light;
    nx_uint16_t    source;
    nx_uint16_t    packet_id;
    nx_uint32_t    watt;
    nx_uint32_t    frequency;
    nx_uint32_t    rms_voltage;
    nx_uint32_t    rms_current;
    nx_uint32_t    plogg_on_time;
    nx_uint32_t    reactive_power;
    nx_uint32_t    phase_angle;
    nx_uint32_t    time_on;
} context_msg_t;

enum {
    AM_CONTEXT_MSG          = 122,
};
```

MIG

ContextMsgWB.java

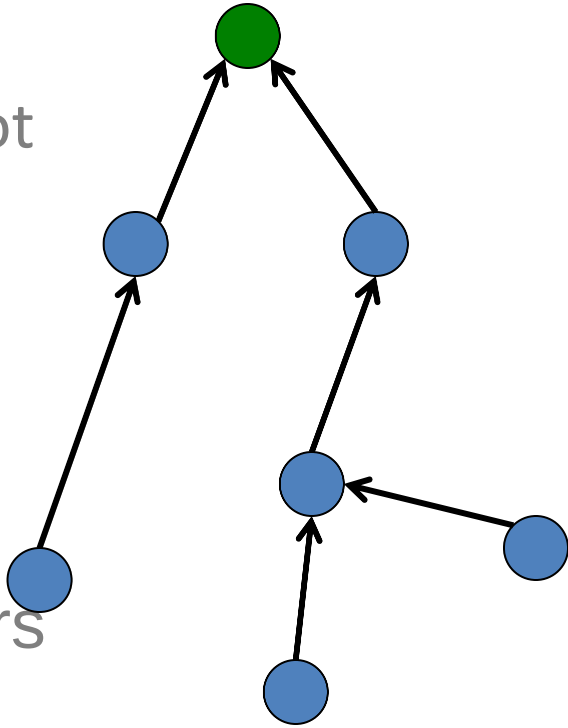
- Automatically generated by MIG
  - Constructor to build the packet
    - No need to manually incapsulate the message
  - setter and getter methods to each message field
    - No need to know field offset
  - toString() method to print the message field
- Statistics can be generated in the same way

Hands-on 2 - PloggDumpWD

- $TOSROOT/apps/PloggDevelop
  - Compile and install on one mote
  - Comment out (no plogg connected): PFLAGS+=-DPM_PLOGG_CONNECTED
  - Adjust the reading period: PFLAGS+=-DPM_FLUSH_DELAY=10000
- Listen raw packets using Listen tool
- Generate ContextMsgWB.java via MIG
  - Header in $TOSROOT/apps/PloggMeter/interface
  - Tip: change the AM_MSG_TYPE to 10
- Listen parsed packets using MsgReader tool

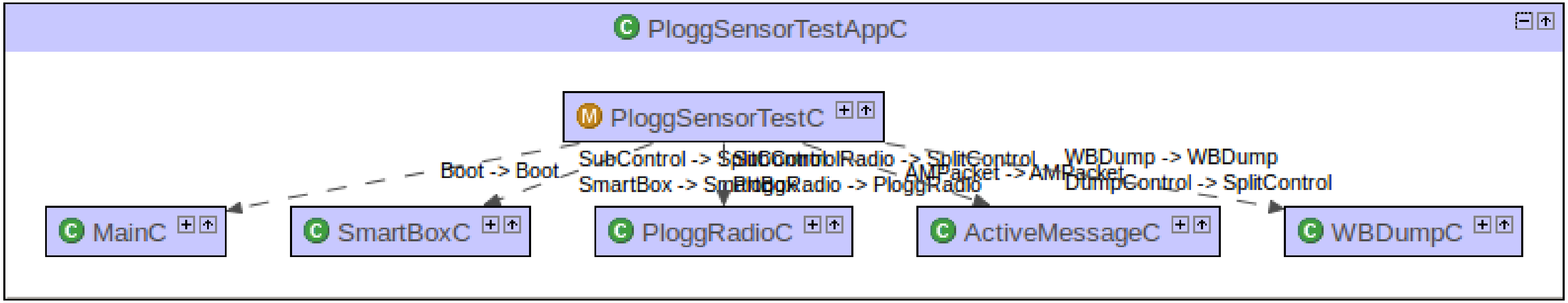
Collection Tree Protocol

- Collect data from the network to one or a small number of roots
  - Build a tree rooted on the selected root
    - call RootControl.setRoot();
  - Use ETX to select forwarder
  - $TOSROOT/tos/lib/net/ctp
- Often used with dissemination
  - Send small data to all the nodes
  - Command, configuration parameters
    - sampling period, …
  - DRIP: $TOSROOT/tos/lib/net/drip
    - call DisseminationUpdate<uint16_t>.change(&value);
    - event void DisseminationValue<uint16_t>.changed() {}



Send Readings over the Radio

- Read the SmartPlogg sensors
- Write readings on Radio
  - Transmit the readings to a selected root (Sink)
- Sink writes reading on Serial
- PloggSensorTestAppC



PloggRadio Interface

```
interface PloggRadio {

    command void sendMessage(uint8_t len, void * payload);
    command void sendUpdate(uint16_t update);

    event void receiveMessage(uint8_t len, void * payload);
    event void receiveUpdate(uint16_t update);

}
```

- ROOT (Sink)
  - Command sendUpdate() to disseminate sampling period
  - Event receiveMessage() to receive network generated message
- Network Nodes (Tree nodes)
  - Command sendMessage() to inject packet in network
  - Event receiveUpdate() to manage disseminated update
- Network operation follow the same split-phase paradigm

PloggSensorTestC – Module

Network Nodes

```
event void SmartBox.samplesReady(context_msg_t* payload) {
		if ( PM_SINK_NODE != TOS_NODE_ID) {
			call PloggRadio.sendMessage(sizeof(context_msg_t), payload);
   }
}
```

- New readings are sent over radio (previously post uartEchoTask())

```
event void PloggRadio.receiveUpdate(uint16_t update) {
		if ( PM_SINK_NODE != TOS_NODE_ID ) {
#ifndef PM_IS_RELAY
		call SmartBox.setSamplingPeriod(update);
#endif
		}
}
```

- Dissemination used to update sampling period (currently not used)

Network

Network

Sink

```
event void PloggRadio.receiveMessage(uint8_t len, void * payload) {
	memcpy(&packet_payload, (context_msg_t*)payload, len);
	post uartEchoTask();
}
```

- Message from network are sent to serial
- Replace behavior of event void SmartBox.samplesReady()
- Payload is the same
- task void uartEchoTask() doesn't change

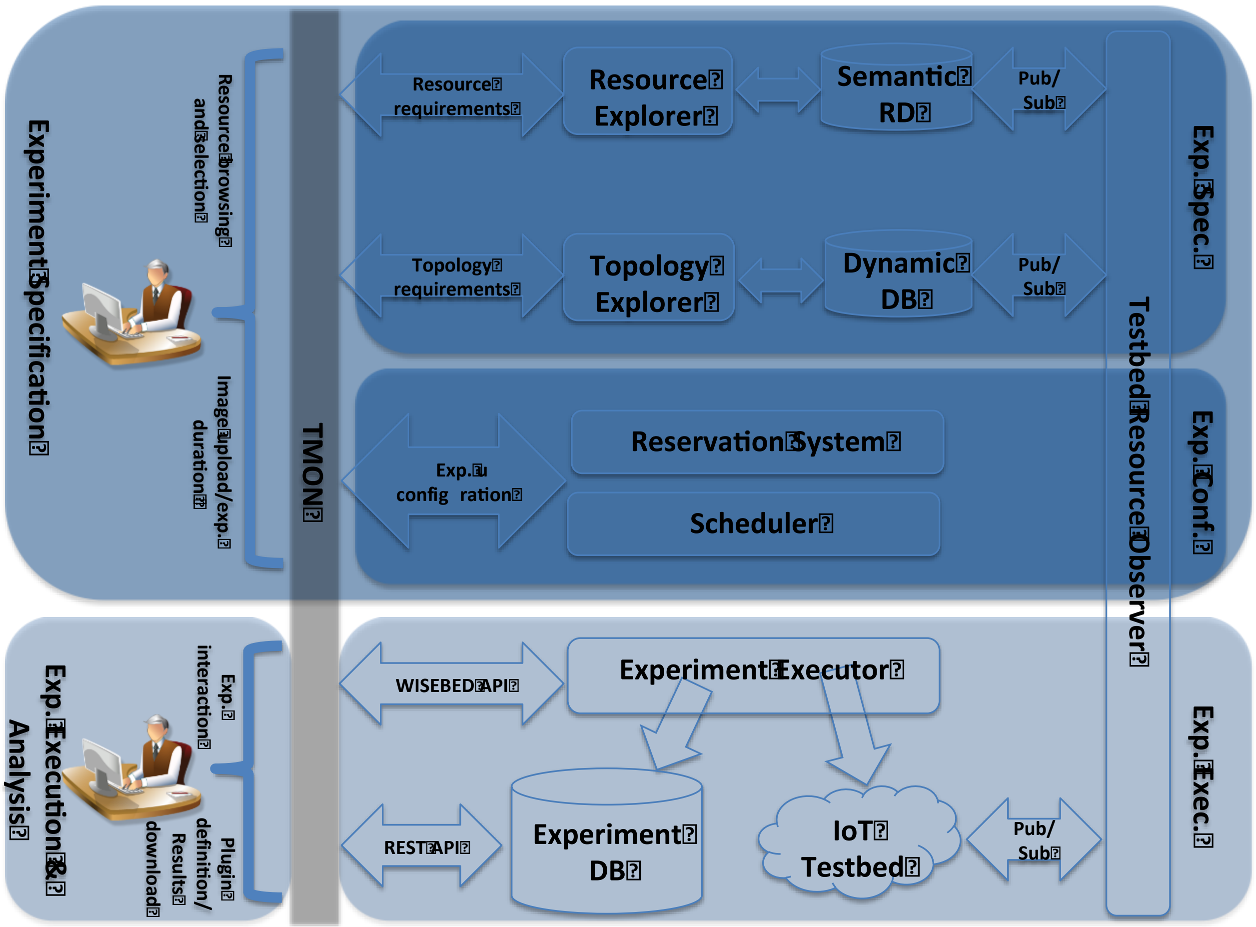
Hands-on 3 - PloggDumpWDMH

- $TOSROOT/apps/PloggDevelopMH
  - Compile and install on two mote
  - **Tip**: Avoid to interfere with your neighbours
  - Sink: set PFLAGS+=-DPM_SINK_NODE=204
  - Adjust sampling period
  - Tip: relay cannot access sensor readings using SmartBox
    - XM1000 motes MUST be relay (why?)
    - Comment in: PFLAGS+=-DPM_IS_RELAY
  - Listen packets at Sink
- In-network processing to reduce the data sent
  - Sent the temperature as average over 10 samples
  - **Tip**: use task

Sensemon (sensor network monitor)

EXPERIMENT WITH REAL SENSORS DEPLOYMENT (SMARTCAMPUS)

Testbed Architecture

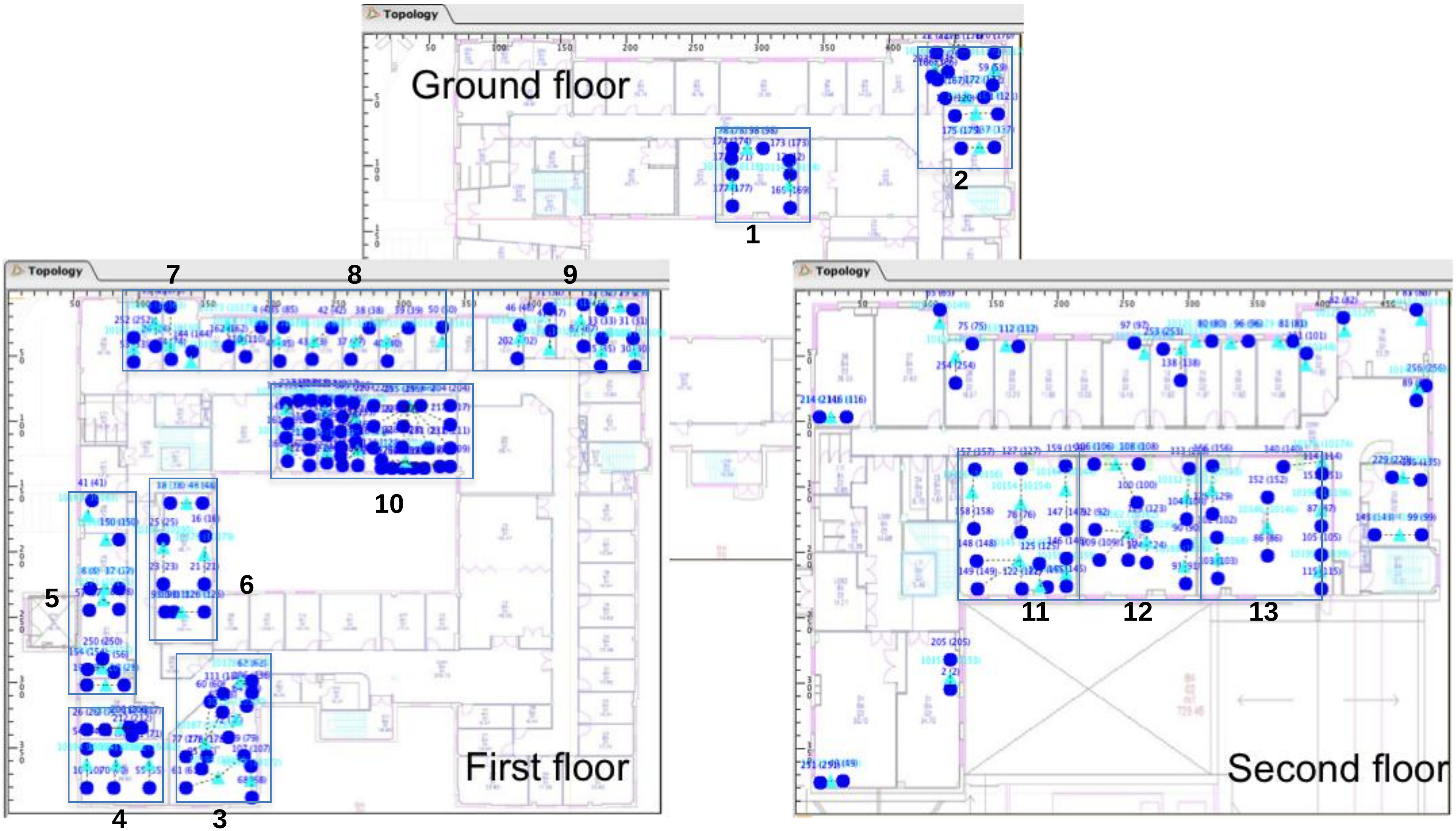


- RD as SSN extension
- Sesami RDF

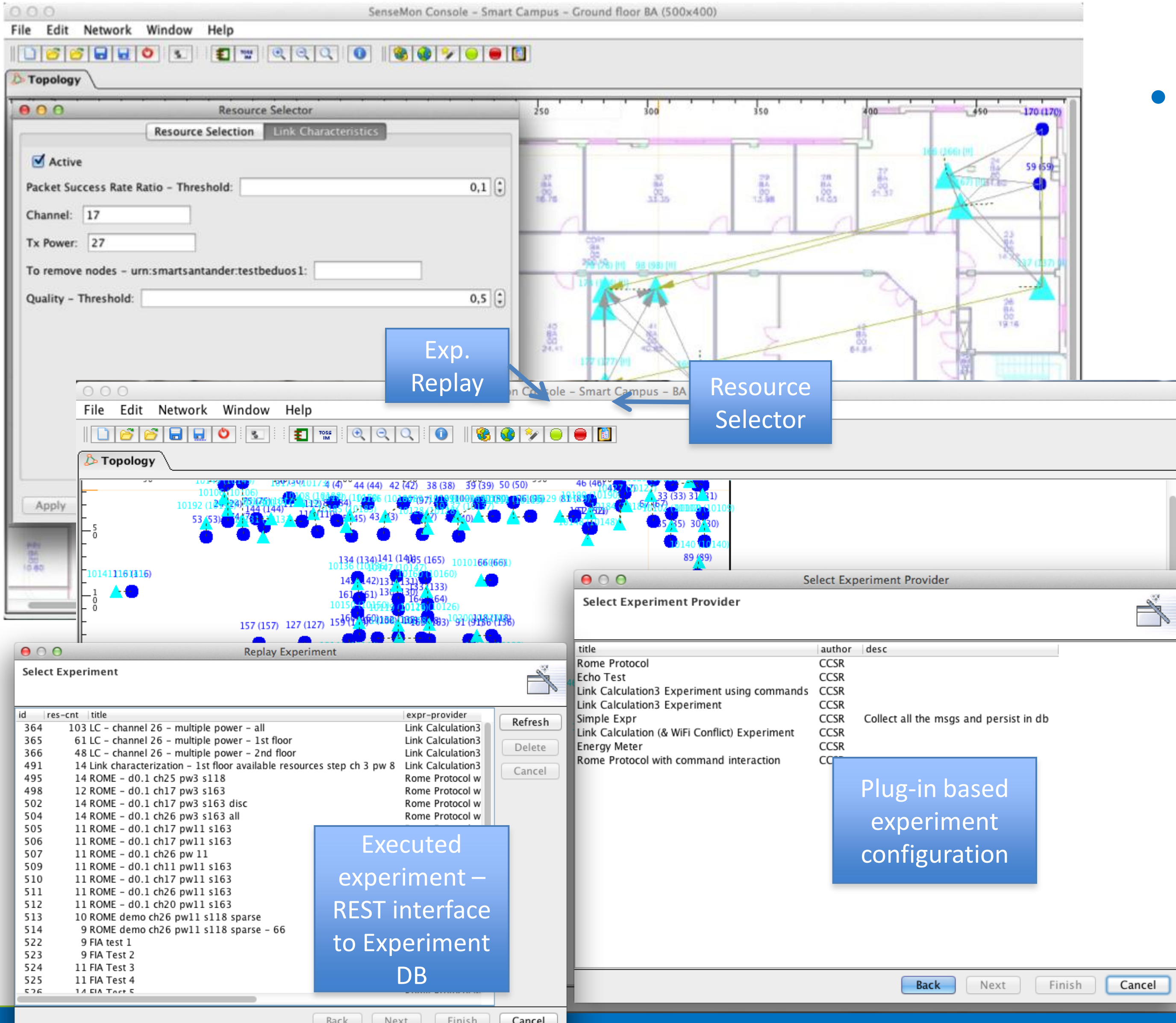
- SmartSantander WISEBED TBR extension

- Big IoT data provisioning (under development)

Deployment overview



TMON – Features



- TMON
  - Semantic resources selection
  - Topology exploration
  - Interference analysis
  - Experiment start/stop/replay/commands injection
  - Plug-in based advanced visualization features

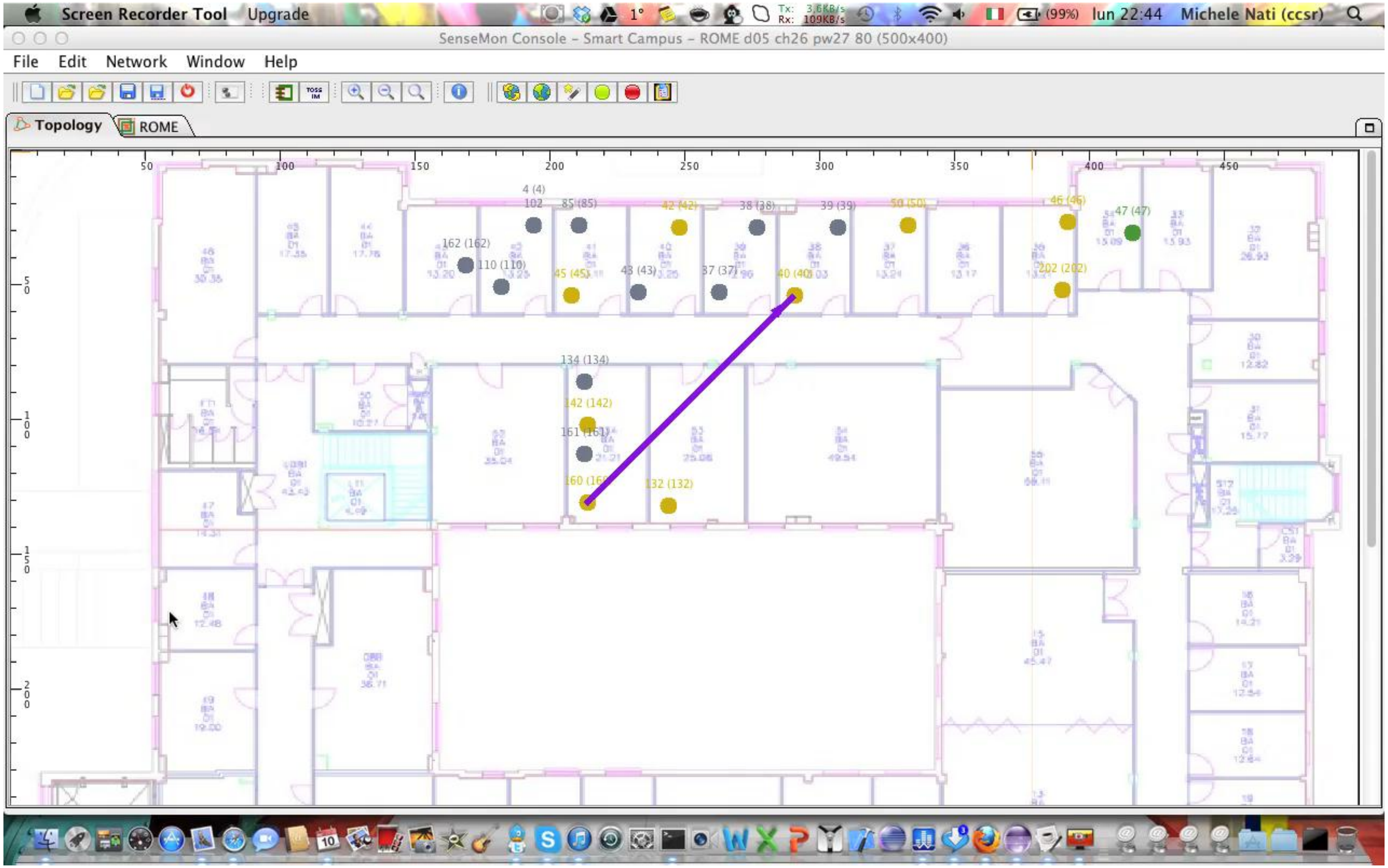
TMON – Walk-through

- Testbed selection
  - Possibility to compose testbeds
- Login
- Resources selection
- Links and interference inspection
- Resources reservation
- Plug-ins presentations
- Replay features

TMON – Available Plug-ins

- Simple Expr
  - Reset/Flash node
  - Receive messages over serial
- Echo Test
  - Send commands/values and receive answers
  - https://www.dropbox.com/s/a5bv8oy4gfq3obp/TestECHO.tar.gz
- Echo Test Lubeck
  - Similar but works with Lubeck backend TBR
- Energy Meter
  - Receive and visualize energy readings
- Rome Protocol
  - Live visualization of WSN routing protocol with real-time statistics computations
- Link Calculation
  - Characterize links characteristics (PER, RSSI, LQI) for selected channel and transmission power values
  - https://www.dropbox.com/s/6492daoj8di2pax/TestChannelsTO.tar.gz

ROME Protocol



Plug-in Develop – Energy Meter

- Sensor Network Application
  - Selected nodes periodically generate sensing info:
    - Presence
    - Temperature
    - Noise
    - Light
    - Energy metering (Power, RP, RMS current, RMS voltage)
  - Transmission over the serial
- Extract and visualize the information

Plug-in Develop – Energy Meter

- uk.ac.surrey.ccsr.tmon.plugin.smartcampus.tr.expr.em
- Extend the AbstractTrExprProvider
  - Define plug-in features/inputs

```
public class ExprEnergyMeterRun extends AbstractTrExprProvider {
........
	@Override
	public JComponent getConfigPanel(boolean isReplay) {
		if (isReplay) {
			return null;
		} else {
			InputPanel pnlConfig = new InputPanel();
			//
			fileImage = new FileInputBox("App Image: ", 20);
			chkFlashTheNode = new BooleanInputBox("Flash the nodes");
			chkResetTheNode = new BooleanInputBox("Reset the nodes");
			//
			pnlConfig.addField(fileImage, true);
			pnlConfig.addField(chkFlashTheNode, false); pnlConfig.nextLine();
			pnlConfig.addField(chkResetTheNode, false); pnlConfig.nextLine();
			//
			//fileImage.setFileName("D:\\images\\images\\main.exe-ROME");
			return pnlConfig;
		}
		//
	}
........
}
```

Plug-in Develop – Energy Meter

- Extend the AbstractTrExprProvider
  - Define experiment execution

```
@Override
protected void runExpr() throws Exception {
		// flash the nodes
		if (chkFlashTheNode.getBooleanValue()) {
			BinaryImage binaryImage = BinaryImage.loadFromFile(fileImage.getFileName(), "main");
			if (binaryImage != null) {
				logger.debug("flashNodes() with " + binaryImage.getFileName());
				try {
			experimentationServiceImpl.flashExperimentImage(secretReservationKeys,
binaryImage.getContent(), nodeUrns);
				} catch (ExperimentationException ex) {
					throw new TrException(ex);
				}
			} else {
				throw new TrException("image not found");
			}
		}
		// reset the nodes
		if (chkResetTheNode.getBooleanValue()) {
			experimentationServiceImpl.resetExperimentNodes(secretReservationKeys, nodeUrns);
		}
}
```

Plug-in Develop – Energy Meter

- Managing the packet payload
  - TinyOS mig interpreter

```
/**
  * Create a new ContextMsgWB using the given byte array
  * as backing store.
  */
 public ContextMsgWB(byte[] data) {
   super(data);
   amTypeSet(AM_TYPE);
 }
/**
  * Return the value (as a short) of the field 'PIR'
  */
 public short get_PIR() {
   return (short)getUIntBEElement(offsetBits_PIR(), 8);
 }
```

  - Custom interpretation of binary payload
- Interaction with the topology/chart objects
  - Real-time representation of the collect information
  - Accordingly changing the color of the nodes

Plug-in Develop – Energy Meter

- Extend the AbstractTrExprProvider
  - Manage received data

```
@Override
public void onNewData(ExperimentResult event) {
		super.onNewData(event);
		//
		if (event.getEtype() == EventType.MSG) {
			try {
				Message tinyOsMessage = extractTinyOsMessage(event);
			}
			if (tinyOsMessage.amType() == ContextMsgWB.AM_TYPE) {
				ContextMsgWB contextMsg = (ContextMsgWB) extractTinyOsMessage(event,
ContextMsgWB.class);
				int nodeId = contextMsg.get_source();
				Color nodeColor = Color.blue;
				if (contextMsg.get_watt() > 0) {
					nodeColor = Color.red;
				} else if (contextMsg.get_PIR() > 0) {
					nodeColor = Color.green;
				} else if (contextMsg.get_light() > 100) {
					nodeColor = Color.yellow;
				}

				Node node = topology.getNodeById(nodeId);
				node.setColor(nodeColor);
				....
```

Sending Commands 1/2

- Link Characterization
  - Algorithm:
    - 1 sender per round
    - Round end is communicated
    - New round start and new sender/parameters are communicated
- Need to send messages
  - TosMsgUtility provides tools for managing TOS packets

Sending Commands 2/2

- uk.ac.surrey.ccsr.tmon.plugin.smartcampus.tr.expr.lc3cmd

```
public void sendCmdMsg(String source, List<String> urns, int channel, int power)
throws ExperimentationException {
        TestMsg test = new TestMsg();
        String[] sourceValues = source.split(":");
        short sourceId = Integer.valueOf(sourceValues[sourceValues.length-1]).shortValue();
        test.set_channel_ID((short)channel);
        test.set_pot((short)power);
        test.set_num_pkts(numPkts);
        test.set_len((short)46);                test.set_source(sourceId);

        byte[] msgToSend = new TOSMsgUtility(sourceId, test).getTOSMessage();

        experimentationServiceImpl.sendMessageExperimentNodes(secretReservationKeys, urns, msgToSend);

}
```

Hands-on 4 - TMON

- Characterize links for your group of nodes
- Select a
  - Tools
- Collect
  - Run e
- Create a … readings exceedi
  - Tip: C … by Collector component provides a *getOrigin* command

Group/Nodes	Channel
1	11
2	12
3	13
4	14
5	15
6	16
7	17
8	18
9	19
10	20
11	21
12	22
13	23

Access experiment results

EXPERIMENT DB

ExprDB REST APIs

- List of experiments
  - GET http://131.227.23.2:8081/sc-repo/rest/api/1.1/expr
- Experiment resources
  - GET http://131.227.23.2:8081/sc-repo/rest/api/1.1/expr/828/resources
- Experiment attributes (duration, image, …)
  - GET http://131.227.23.2:8081/sc-repo/rest/api/1.1/expr/828/attrs
- Experiment results
  - GET http://131.227.23.2:8081/sc-repo/rest/api/1.1/expr/828/results?start=0&limit=100

Fetch Experiment Results

- uk.ac.surrey.ccsr.tmon.plugin.smartcampus.tr.expr.utils
  - ExprDBAccess.java

```
public Vector<ExperimentResult> getResults(Experiment exp) {

		int cursor = 0;
		int bufferCursor = 0;

		Vector<ExperimentResult> buffer = new Vector<ExperimentResult>(DEFAULT_BATCH_SIZE, DEFAULT_BATCH_SIZE);


		try {
			List<ExperimentResult> results = daoExperiment.findExperimentResults(exp, 0, DEFAULT_BATCH_SIZE);

			while(results.size() != 0) {
				buffer.addAll(bufferCursor, results);
				cursor = bufferCursor += results.size();
				results = daoExperiment.findExperimentResults(exp, cursor, DEFAULT_BATCH_SIZE);

			}

		} catch (DaoException de) {

		}
		return buffer;

}
```

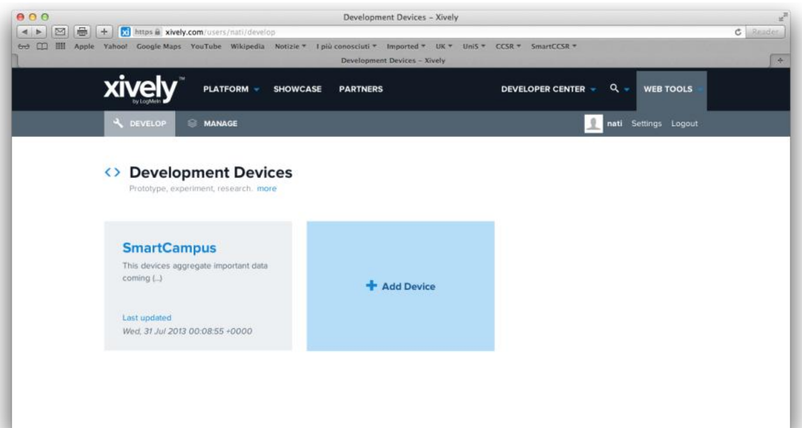
The Xively integration

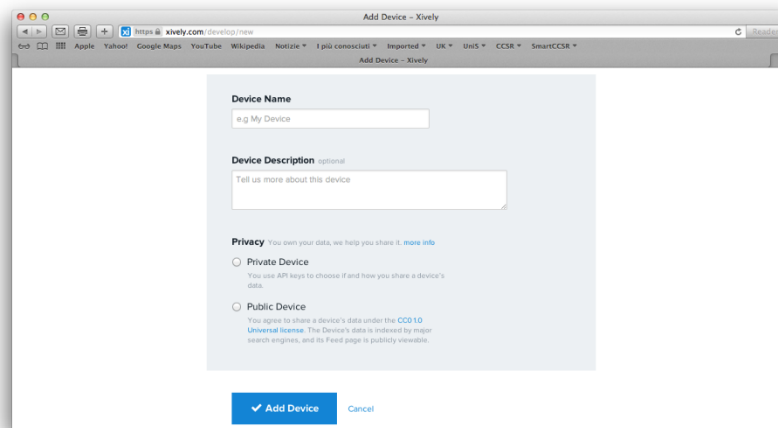
EXPORT AND INTEGRATE YOUR DATA

What is Xively?

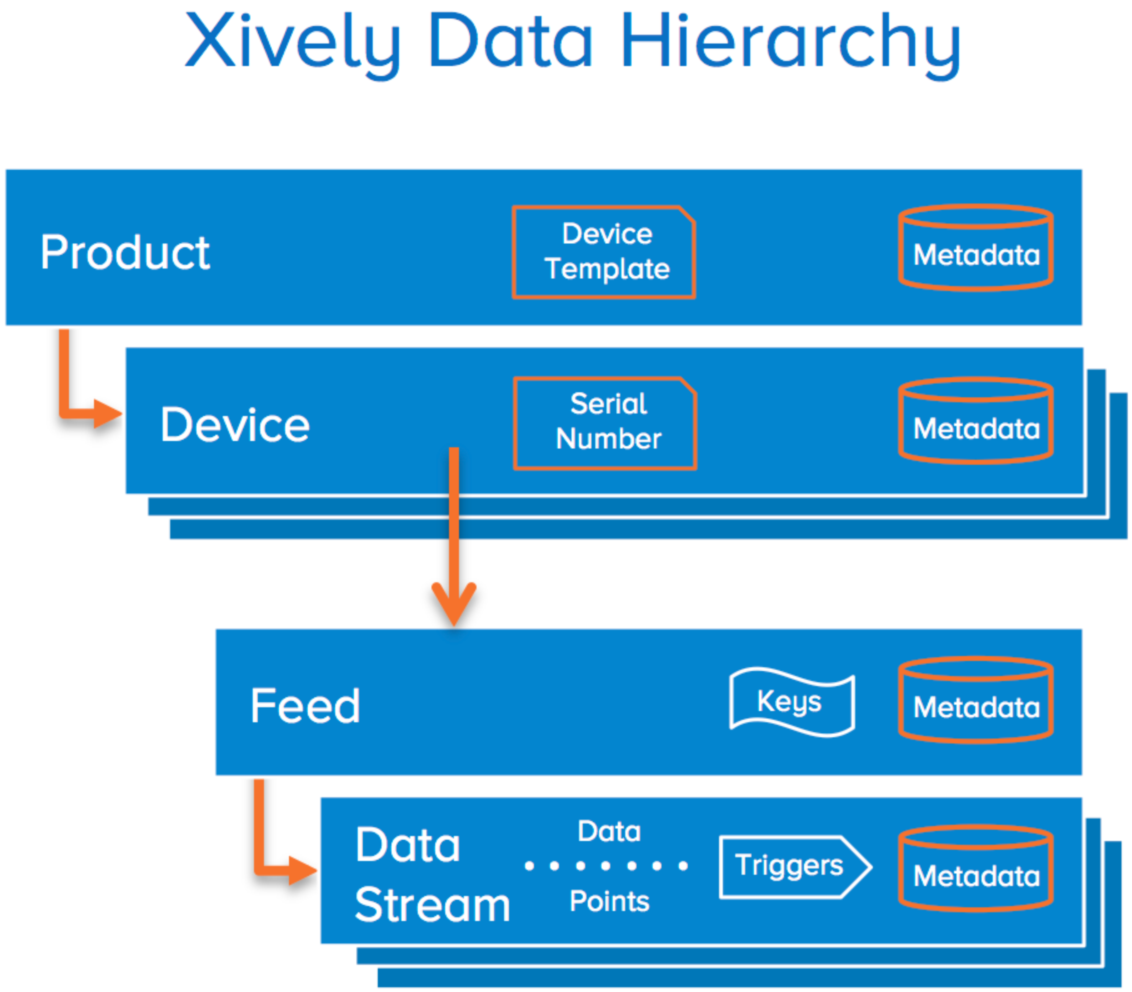
- Xively "*provides the* ***platform, tools, services and partners*** *that simplify and accelerate the creation of compelling connected offerings. With Xively, you're free to* ***focus on innovation instead of infrastructure***."
- What does it mean in practice?
- Get an account
  - https://xively.com/get_started/
  - Free developer account

Create a Device





Xively Data



Xively Data Hierarchy
Product
Device Template
Metadata
Device
Serial Number
Metadata
Feed
Keys
Metadata
Data Stream
Data Points
Triggers
Metadata

Write Data to Xively

- Write a single Datapoint to a single Datastream (for example, the current value from a single sensor on a device).
- Write single Datapoints to multiple Datastreams (for example, the current value from multiple sensors on one or more devices).
- Write multiple Datapoints to a single Datastream (for example, buffered historical values from a single sensor on a device).
- Write multiple Datapoints to multiple Datastreams (for example, multiple sets of buffered historical values, each from a different sensor on one or more devices).

Multiple datapoints, single datastream

- **Parameters**
  - **Method** PUT
  - **Base URL** https://api.xively.com
  - **API Endpoint** /v2/feeds/FEED_ID_HERE
- **Headers**
  - **X-ApiKey** API_KEY_HERE

PUT/v2/feeds/FEED_ID_HERE.json

```
{
 "version":"1.0.0",
  "datastreams" : [ {
     "id" : "example",
     "datapoints":[
     {"at":"2013-04-22T00:35:43Z","value":"42"},
     {"at":"2013-04-22T00:55:43Z","value":"84"},
     {"at":"2013-04-22T01:15:43Z","value":"41"},
     {"at":"2013-04-22T01:35:43Z","value":"83"}
     ],
     "current_value" : "40"
   }
  ]
}
```

Read Data from Xively

- Historical data
  - Method: GET
  - URL: https://api.xively.com/v2/feeds/***feed_id***?***range***
  - ***range*** is one of the following:
    - start=***timestamp***
    - end=***timestamp***
    - start=***timestamp***&end=***timestamp***
    - start=***timestamp***&duration=***time_unit***
    - ***timestamp*** is an ISO 8601 formatted date

Xively Utility for TMON

- uk.ac.surrey.ccsr.tmon.plugin.smartcampus.tr.expr.utils
  - RESTfulClient.java
    - Constructor → (url, username, password)
    - get() → to read
    - put(JSON string) → to write
  - DataJSONizer.java (currentyl only one stream)
    - Java Object ←→ JSON
    - Two Constructors
      - new DataJSONizer(id, current_value)
      - New DataJSONizer(id)
        - » addDataPointsValue
        - » setDataStreamsCurrentValue

Hands-on 5 – Export data

- Export data collected via Energy Meter to Xively
  - uk.ac.surrey.ccsr.tmon.plugin.smartcampus.tr.expr.emx
- More on
  - https://xively.com/dev/docs/api/
  - https://xively.com/dev/tutorials/
- Visualize data or integrate them with other sources

SEVENTH FRAMEWORK PROGRAMME

SMART SANTANDER

Part 4- Setting up your own Testbed

Learning Goals

- After this session you will know…
  - how to set up your own
    - desktop testbed
    - lab testbed
  - what the Testbed Runtime project is
  - where to find relevant documentation
  - how to federate testbeds

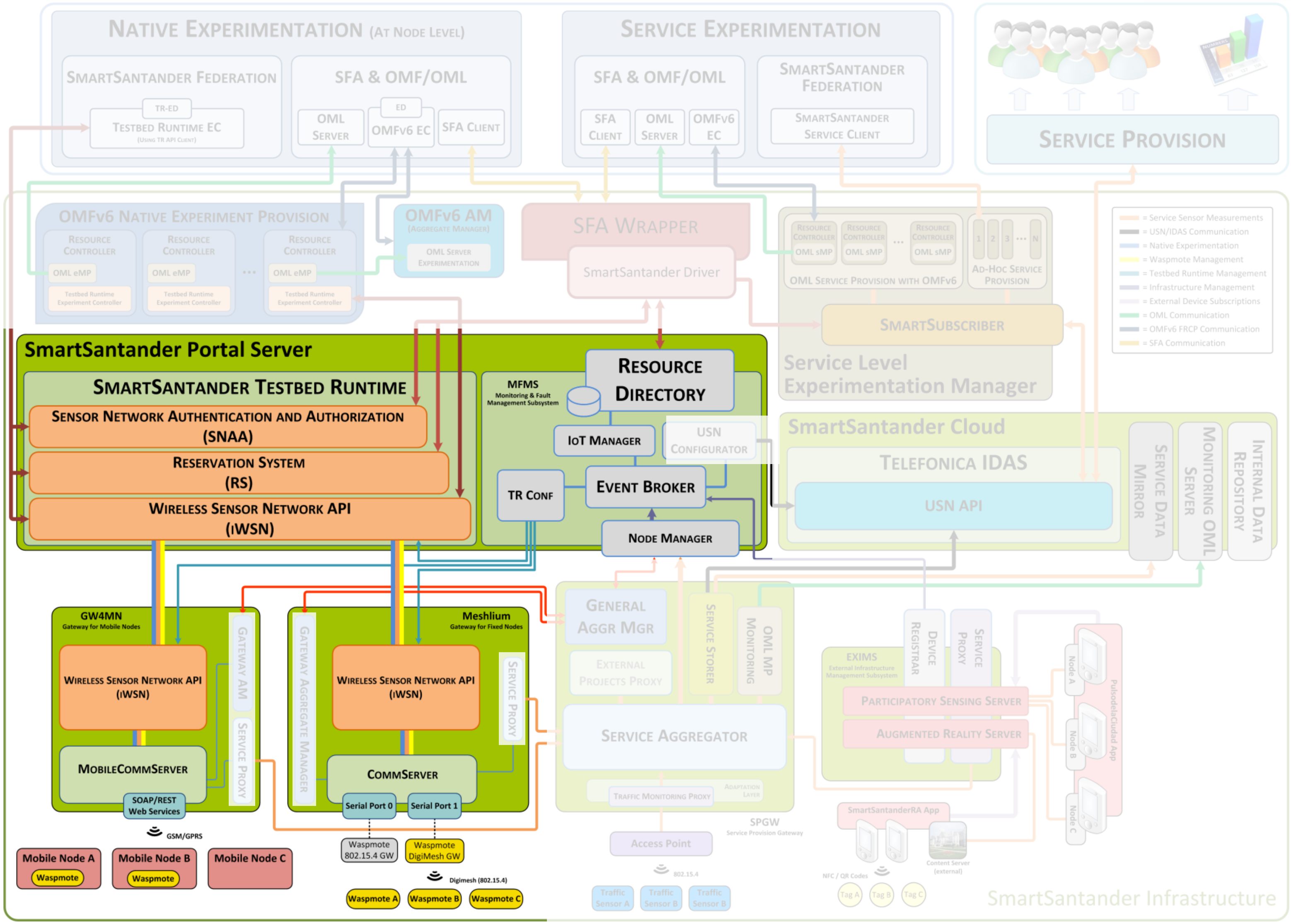
Outline

1. Testbed Architecture
2. The Testbed Runtime Project
3. Testbed Runtime in Action
4. Installation
5. API Basics
6. Configuration
7. Extending TR with Plugins
8. Federation
9. Summary & References

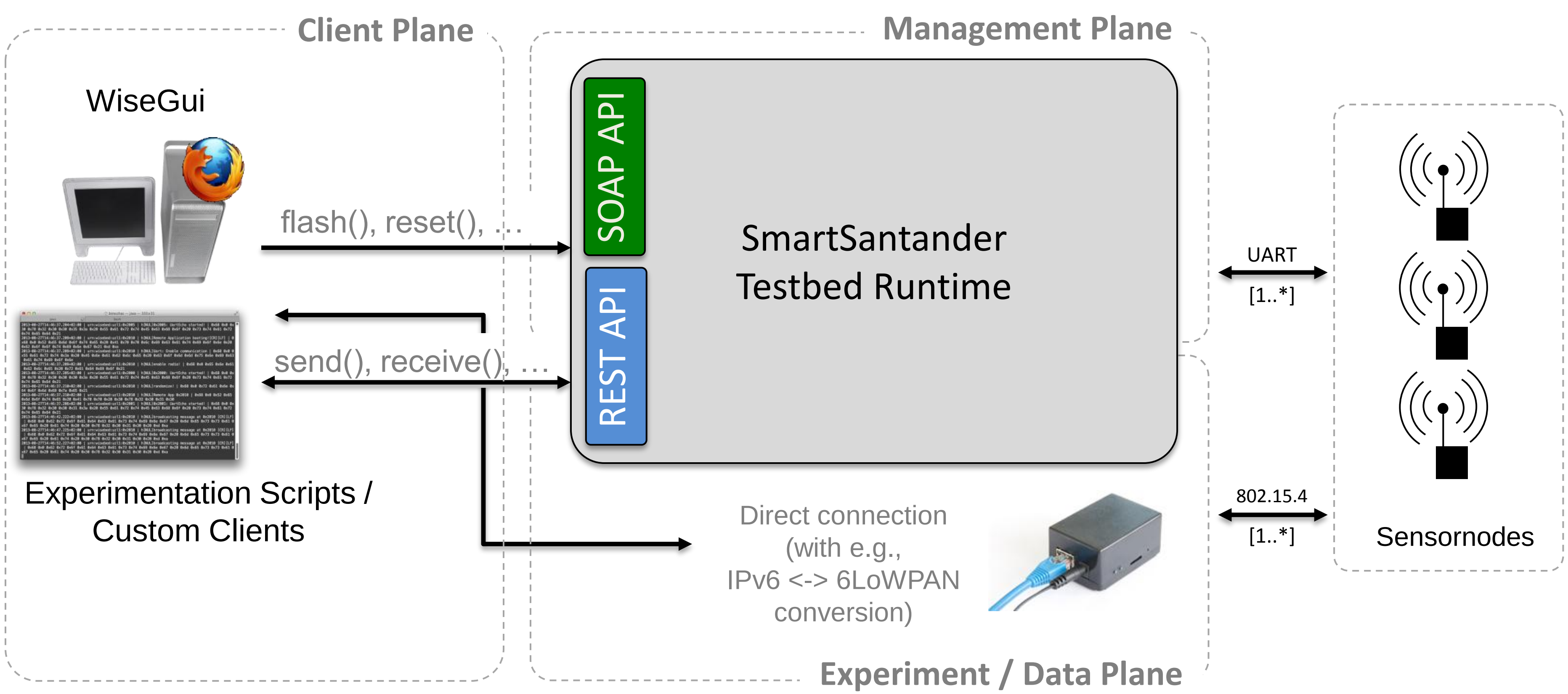
Setting up your own Testbed

1. TESTBED ARCHITECTURE

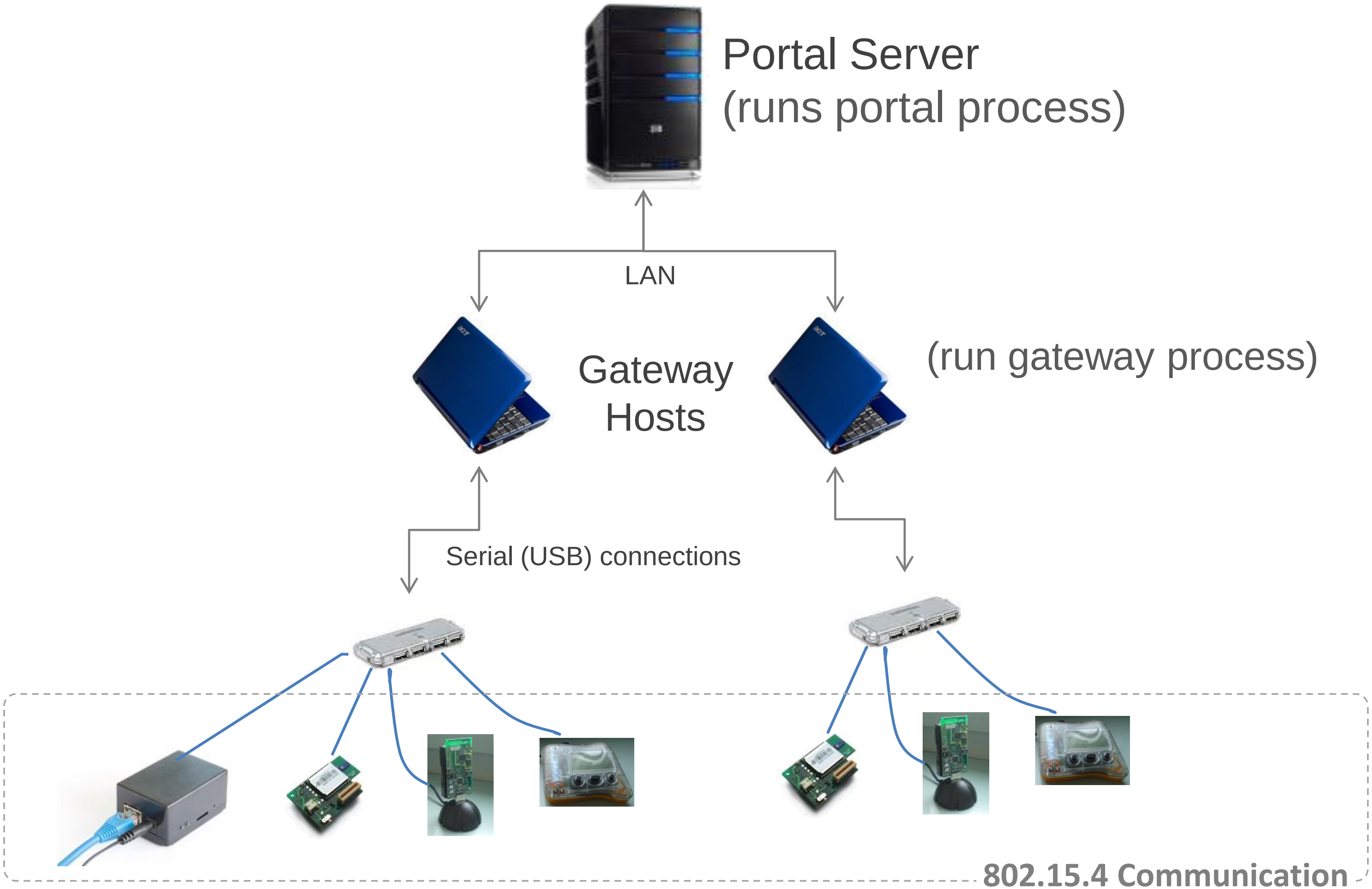
1. Testbed Architecture: SmS Context



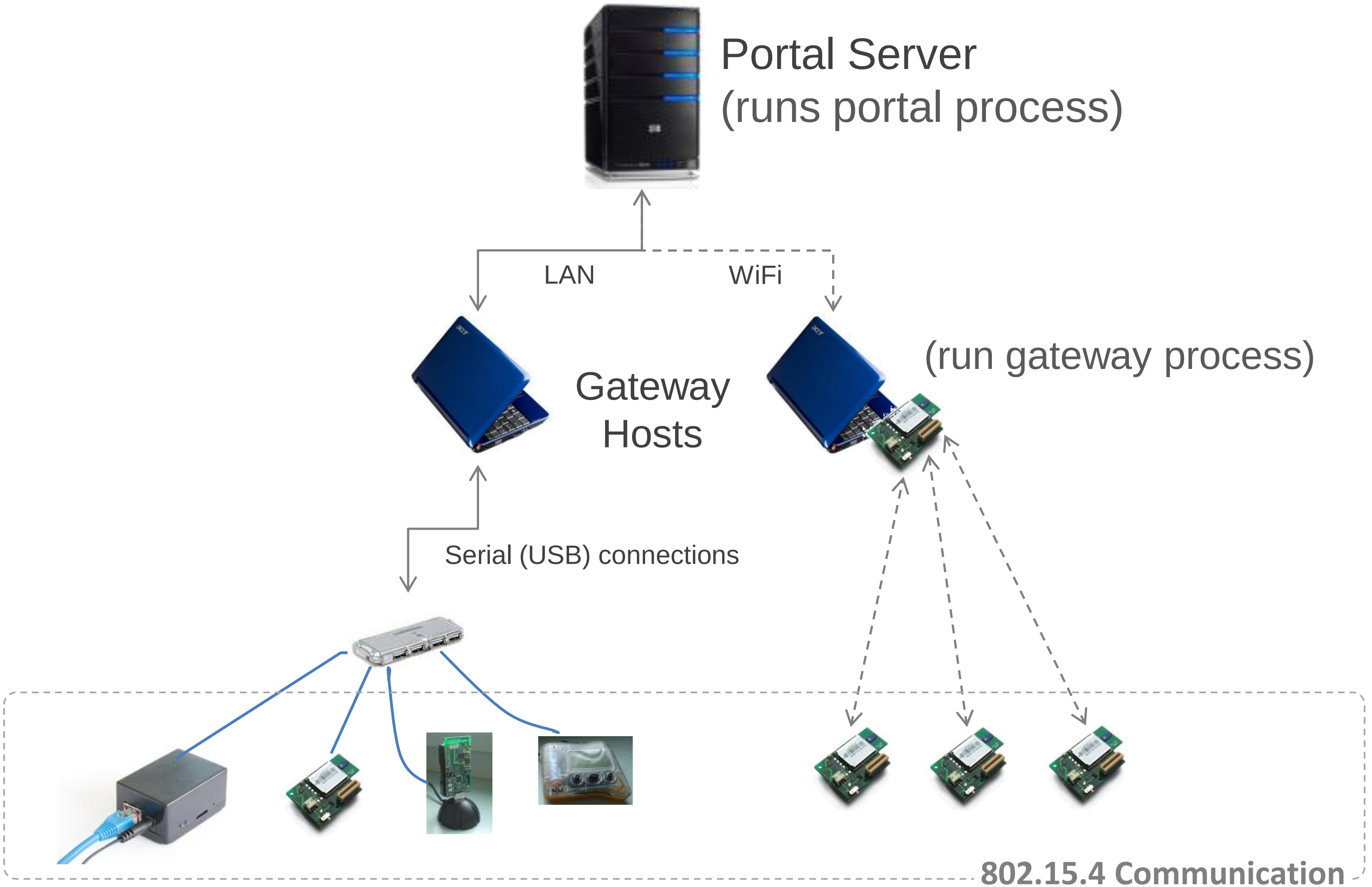
1. Testbed Architecture: Planes



1. Testbed Architecture: Hardware



1. Testbed Architecture: Hardware



Setting up your own Testbed

2. THE TESTBED RUNTIME PROJECT

2. The Testbed Runtime Project

Testbed Runtime

- Reference implementation of WISEBED APIs
- The “backbone” of testbeds
- 20 (bugfix) releases in the last 3 years
- Run at 5/9 testbed sites of WISEBED project
- Basis for node-level experimentation in the SmartSantander project

2. The Testbed Runtime Project

- Source Code
  https://github.com/itm/testbed-runtime/

- Documentation
  https://github.com/itm/testbed-runtime/wiki

- Issue Tracker
  https://github.com/itm/testbed-runtime/

- Mailing Lists
  testbed-runtime-users@wisebed.eu
  wisebed-users@wisebed.eu

2. The Testbed Runtime Project

- Last release: Version 0.8.5, about a year ago
- Today: Version 0.9 beta (soon to be released)

- Version 0.9 is a complete rewrite with
  – lots of improvements,
  – many simplifications (e.g., configuration),
  – higher performance,
  – cleaner design,
  – better extensibility,
  – embedded REST API and WiseGui frontend

Setting up your own Testbed

3. TESTBED RUNTIME IN ACTION

3. Testbed Runtime in Action

- Start portal & gateway process
  - JAR file on USB stick or from https://maven.itm.uni-luebeck.de

```
java -jar tr.iwsn-portal-0.9-server.jar --logLevel INFO --config portal.properties
```

```
java -jar tr.iwsn-gateway-0.9-server.jar --logLevel INFO --config gateway.config
```

- Add devices @ http://localhost:9999/devicedb
- Browse to http://localhost:9999/
- Done ☺

3. Testbed Runtime in Action

- portal.properties

```
urnprefix                                   = urn:local:

portal.overlay.port                         = 9990
portal.configuration.sm_endpoint_uri        = http://localhost:9999/soap/v3/sm
portal.configuration.snaa_endpoint_uri      = http://localhost:9999/soap/v3/snaa
portal.configuration.rs_endpoint_uri        = http://localhost:9999/soap/v3/rs
portal.configuration.wsn_endpoint_uri_base = http://localhost:9999/soap/v3/wsn

snaa.type                                   = DUMMY

rs.type                                     = IN_MEMORY

devicedb.type                               = IN_MEMORY
devicedb.webapp.context_path                = /devicedb
devicedb.rest_api.context_path              = /rest/v1.0/devicedb

wisegui.testbed_name                        = My local testbed
```

3. Testbed Runtime in Action

- gateway.properties

```
urnprefix                   = urn:local:

gateway.portaladdress       = localhost:9990

devicedb.type               = REMOTE
devicedb.remote.uri         = http://localhost:9999/rest/v1.0/devicedb
```

Setting up your own Testbed

4. INSTALLATION – DEBIAN PKG

4. Installation – Debian Pkg

- Testbed Runtime is available as Debian package!
  - Add ITM Debian repository to /etc/apt/sources.list.d/

```
wget -O - http://dev.itm.uni-luebeck.de/debian-repo/testbed-runtime-repo.gpg.key | apt-key add -
cd /etc/apt/sources.list.d
wget http://dev.itm.uni-luebeck.de/debian-repo/testbed-
```

  - Install using apt-get

```
apt-get install tr.iwsn-portal
apt-get install tr.iwsn-gateway
```

4. Installation – Debian Pkg

- Configure /etc/tr.iwsn-(portal|gateway).properties
- List Contents

```
dpkg -L tr.iwsn-portal
  /usr
  /usr/share
  /usr/share/tr.iwsn-portal
  /usr/share/tr.iwsn-portal/tr.iwsn-portal-0.9-SNAPSHOT-server.jar
  /usr/bin
  /usr/bin/tr.iwsn-portal
  /etc
  /etc/init.d
  /etc/init.d/tr.iwsn-portal
  /etc/tr.iwsn-portal.log4j.properties
  /etc/tr.iwsn-portal.properties
  /etc/tr.iwsn-portal.properties.example
  /var
  /var/lib
  /var/lib/tr.iwsn-portal
  /var/log
  /var/log/tr.iwsn-portal
  /usr/share/tr.iwsn-portal/plugins
  /usr/share/tr.iwsn-portal/tr.iwsn-portal.jar
```

4. Installation – Debian Pkg

- Run

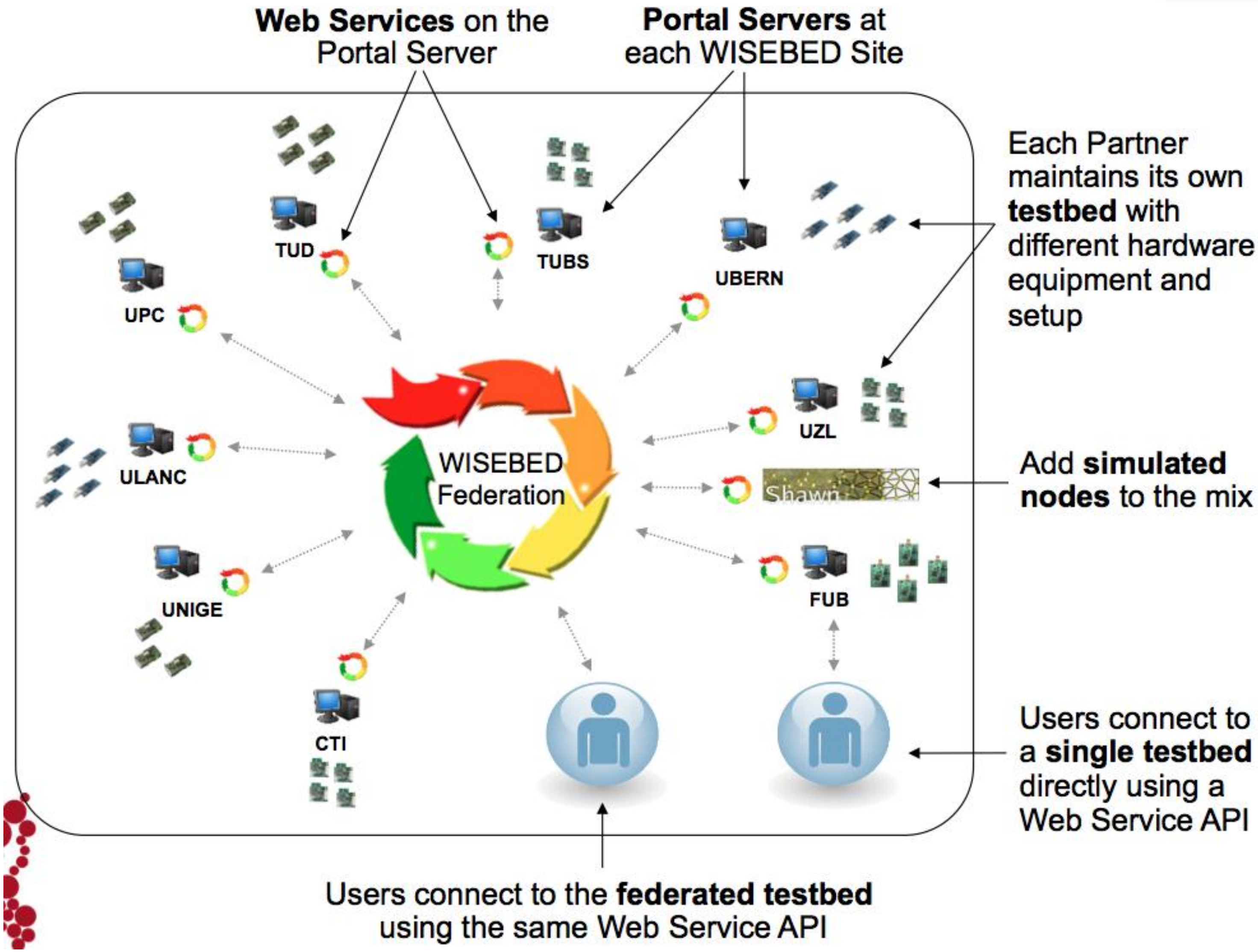
```
/etc/init.d/tr.iwsn-portal start | stop | restart
/etc/init.d/tr.iwsn-gateway start | stop | restart
```

- Done ☺

Setting up your own Testbed

5. API BASICS

5. API Basics - The WISEBED Project



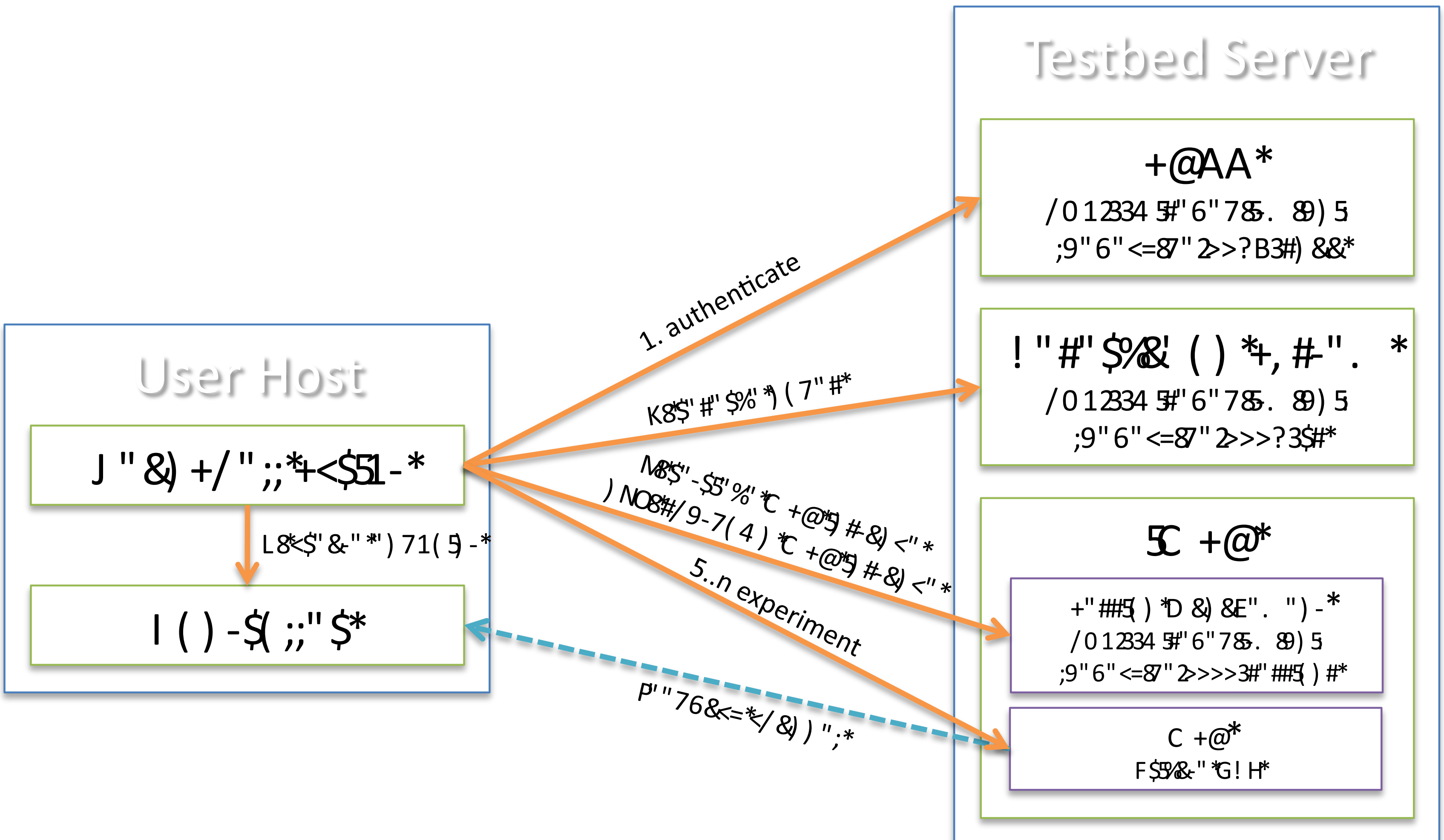
5. API Basics: The WISEBED Approach

- Establish a well-defined, standardized API for sensor node-level experimentation
- Allows to wrap existing testbeds
- Enables client application to rely on API
  - TMON
  - WiseGui
  - Experimentation Scripts
  - Custom Clients
  - …

5. API Basics: SOAP API Endpoints

Name	Abbreviation	Description
Reservation System API	RS	Allows to create, query and delete reservations for a (sub-)set of sensor nodes for a given timespan
Sensor Network Authentication and Authorization API	SNAA	Provides basic authentication and authorization mechanisms
Session Management API	SM	Manages sessions (accessible through WSN API). Each session corresponds to a reservation made at the RS.
Wireless Sensor Network API	WSN	Given access to both management and communication plane functionality
Controller API	---	Runs at client host, “controls” an experiment. Backchannel to retrieve sensor outputs and results of long running operations

5. API Basics: SOAP API - Interactions

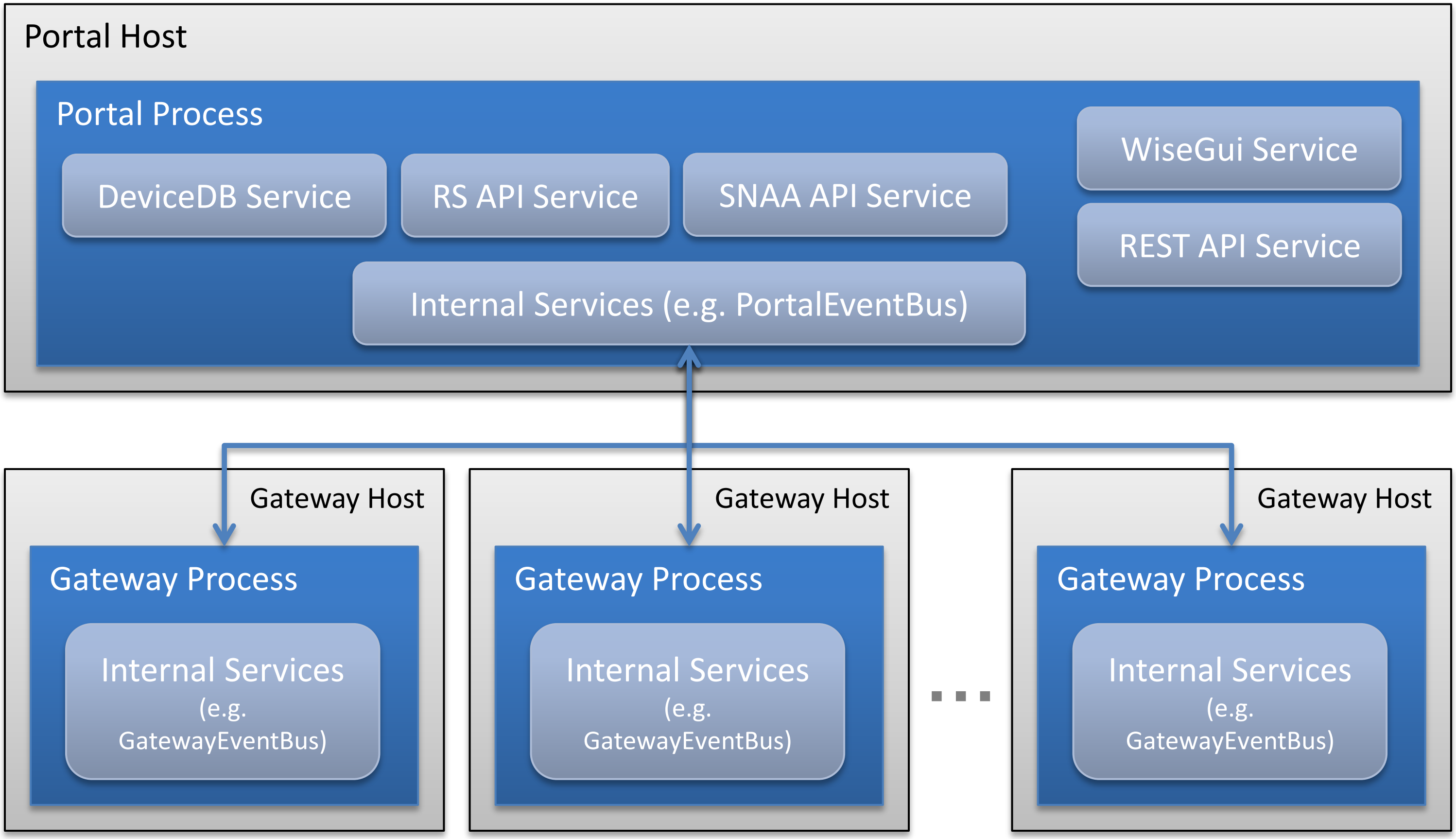


Setting up your own Testbed

6. CONFIGURATION

6. Configuration – Modules

- TR services are realized as individual modules



6. Configuration – Modules

- Every module knows a set of configuration parameters
- Display all configuration of all modules with

```
java -jar tr.iwsn-portal-0.9-server.jar --helpConfig
```

or

```
java -jar tr.iwsn-gateway-0.9-server.jar --helpConfig
```

6. Configuration – SNAA Module

- SNAA module can be configured to use different authentication & authorization backend types

Type	Description
DUMMY	Always returns “true”
JAAS	Uses the Java Authenthication and Authorization Service standard. Comparable to PAM. Provides an API to arbitrary backends like htpasswd, LDAP, …
SHIBBOLETH	Single Sign-on system used in the WISEBED federation (project specific)
SHIRO	Backend with custom database authorization scheme, used in SmartSantander. Based on the Apache Shiro Java security framework.
REMOTE	If SNAA service is used internally and requests should be delegated to a remote SNAA service.

6. Configuration – SNAA Module

- RS module can be configured to use different persistence layers

Type	Description
IN_MEMORY	“Persistence” only in memory. Restart service to have an empty “database”.
GCAL	Uses the Google Calendar service API as persistence layer.
JPA	Uses the Java Persistence API, basically allowing RS to be used with all types of relational databases.
REMOTE	If RS service is used internally and requests should be delegated to a remote RS service.

6. Configuration – SNAA Module

- … and some more module options remain!
- Full configuration parameters reference:
  https://github.com/itm/testbed-runtime/wiki

Setting up your own Testbed

7. EXTENDING TR WITH PLUGINS

7. Extending TR with Plugins

- New in TR Version 0.9: Plugins ☺
- Plugins can be developed for both Portal and Gateway host
- Examples for portal plugins:
  - Plugin that records node outputs in a database & exposes a Web service to retrieve recorded outputs from DB
  - Plugin that automatically flashes unreserved nodes with a default image (e.g., to assess the communication channel topology by measuring link qualities)
  - Plugin that measures usage statistics
- Examples for gateway plugins
  - New drivers for currently unsupported device types

7. Extending TR with Plugins

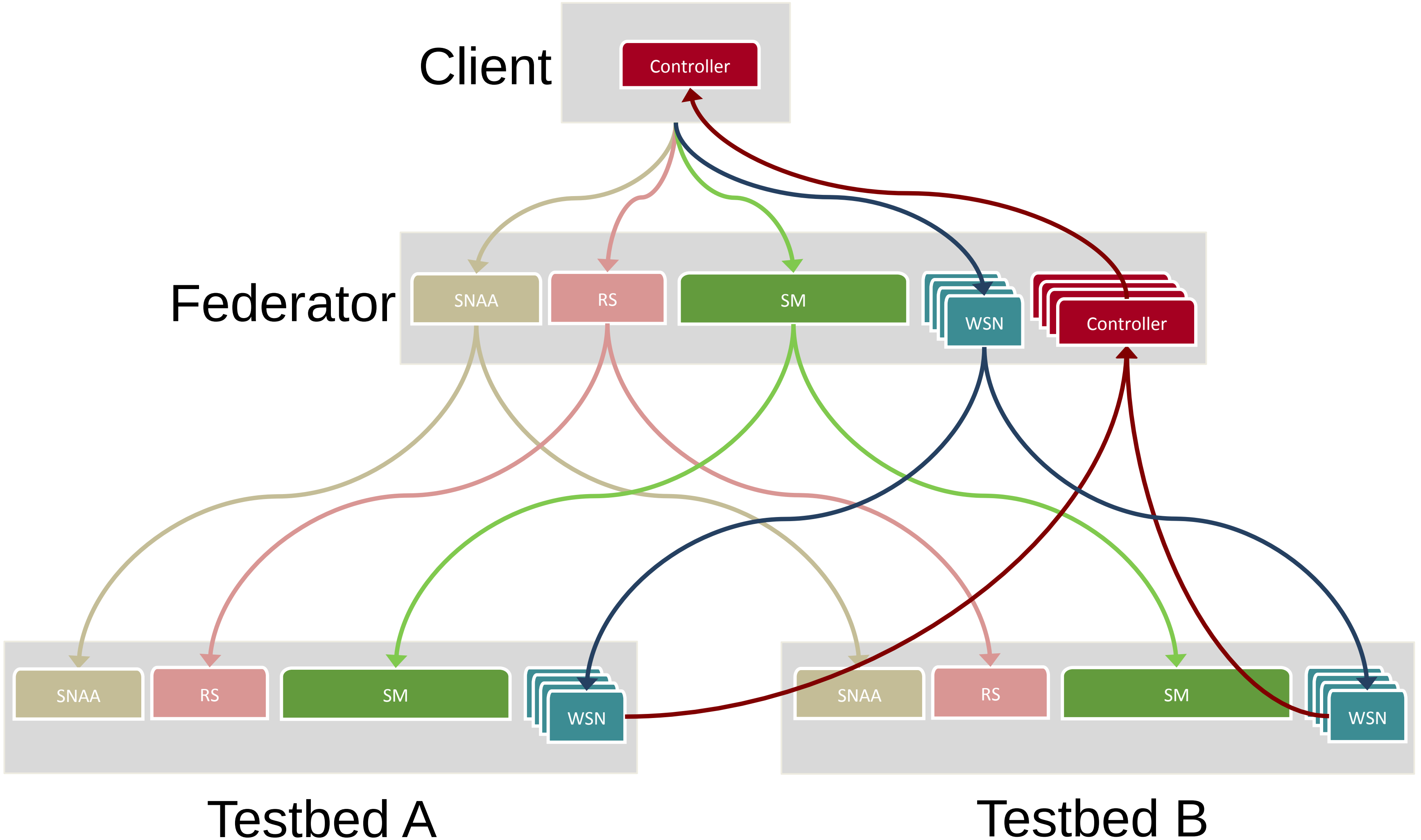
- Plugins…
  - are based on the OSGi platform
  - can be installed and uninstalled at runtime by copying to / deleting a plugin jar from a plugin directory
  - can access all internal events of TR
  - are simple to implement

- Currently available plugins:
  - Mock Device
    (runs on gateway, for deployment debugging)
  - Default Image Plugin (runs on portal)

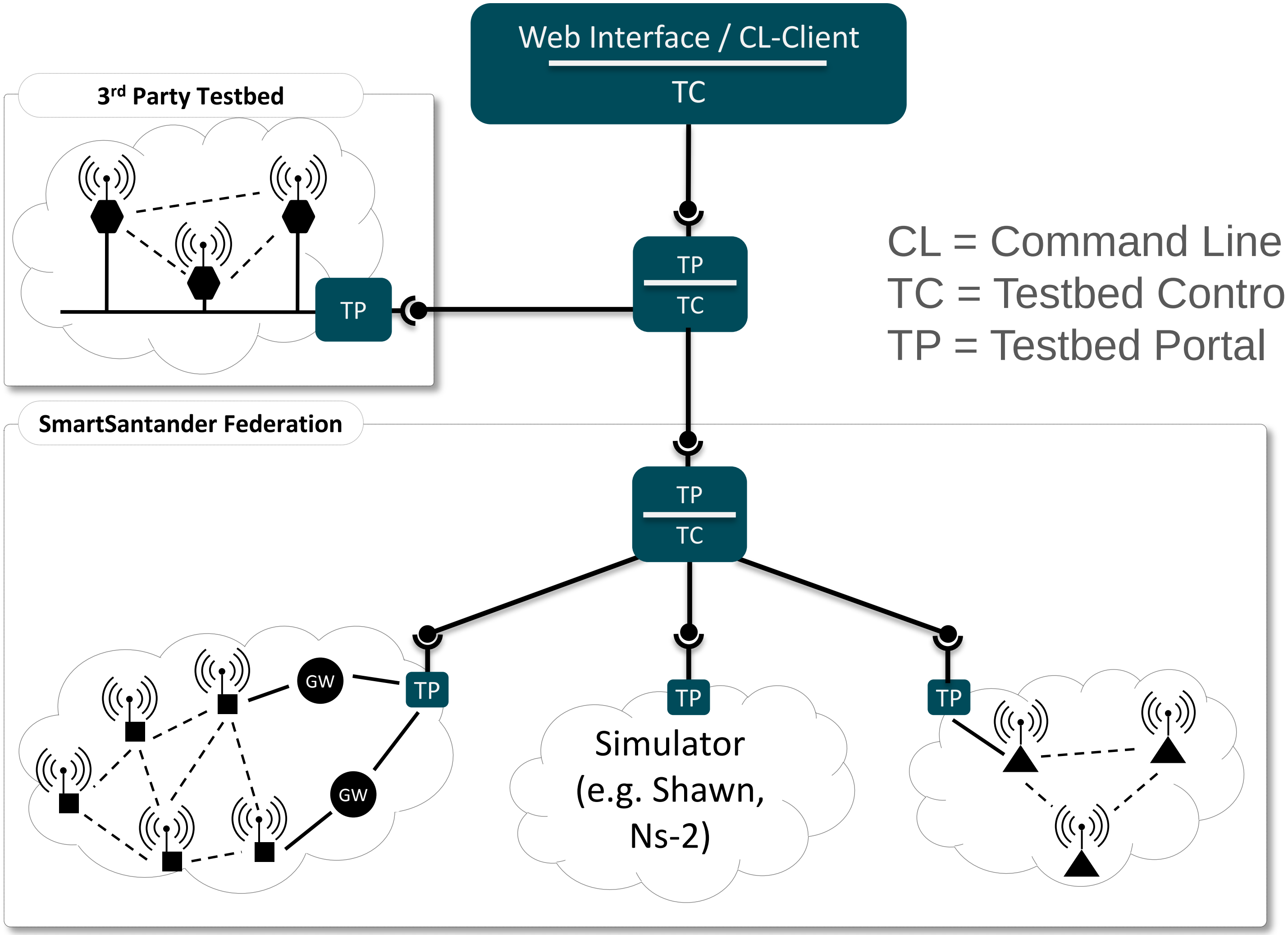
Setting up your own Testbed

8. FEDERATION

8. Federation



8. Federation



CL = Command Line Client
TC = Testbed Controller
TP = Testbed Portal

8. Federation

- Install using apt-get, run using init script

```
apt-get install tr.federator
vi /etc/tr.federator.properties
/etc/init.d/tr.federator start
```

- Download manually from Maven repo (http://maven.itm.uni-luebeck.de), run manually

```
java -jar tr.federator-0.9-SNAPSHOT.jar --logLevel TRACE –
config tr.federator.properties
```

- List configuration options

```
java -jar tr.federator-0.9-SNAPSHOT.jar --helpConfig
```

Setting up your own Testbed

9. RESOURCES

9. Resources

- Testbed Runtime Source Code, Issue Tracker, Documentation
  https://github.com/itm/testbed-runtime
  https://github.com/itm/testbed-runtime/wiki
- Join Mailing Lists
  testbed-runtime-users-subscribe@wisebed.eu
  wisebed-users-subscribe@wisebed.eu
- Contact directly
  bimschas@itm.uni-luebeck.de