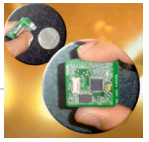
Imperial College
London

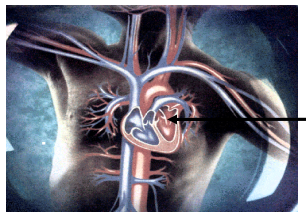
Autonomic Management of Ubiquitous Systems for e-Health

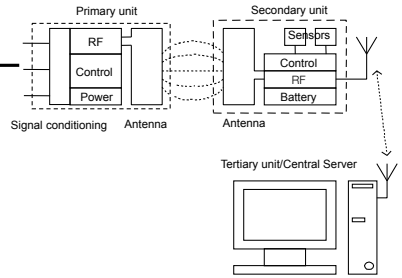
Emil Lupu
Department of Computing
Imperial College London
e.c.lupu@imperial.ac.uk
http://www-dse.doc.ic.ac.uk/~ecl1/

Cardiac Monitoring



UbiMon Body Sensor Node

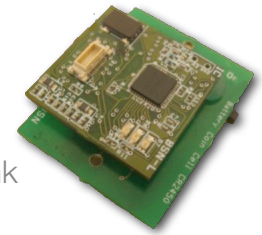


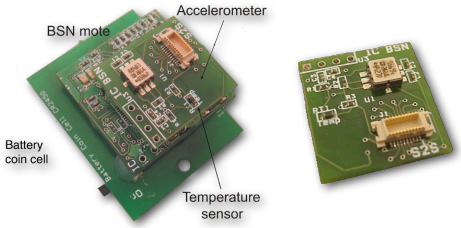


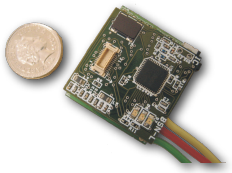
UbiMon
www.ubimon.net

The BSN platform

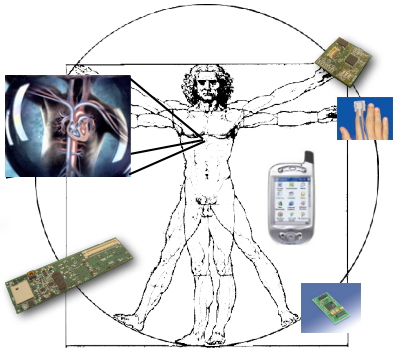
- TinyOS
- Ultra low power 16 bit processor
- 64KB + 256KB Flash memory
- 6 analog channels
- IEEE 802.15.4 (Zigbee) wireless link







Body Area Networks for eHealth



Body Area Networks

- Implanted and wearable sensors: Heart monitoring, blood-pressure, oxygen saturation, etc.
- Need for **continuous adaptation**:
  - sensor failures, new sensors and diagnostic units
  - changes in user activity and context
  - changes in the patient's medical condition
  - interactions with other medical and non medical equipment e.g. nurse visits at home etc.

Policies

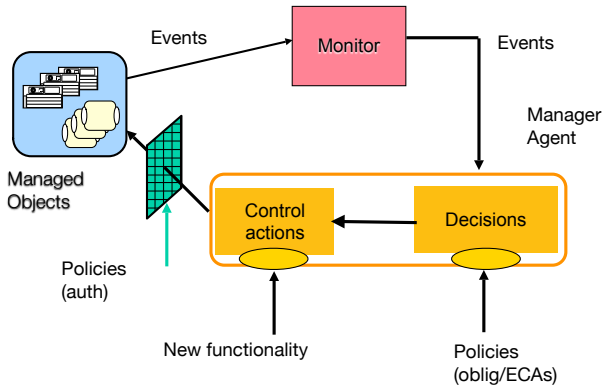
Rules governing choices in the behaviour of systems

- Derive from the need to separate strategy for adaptation from the implementation of functional aspects.
- Can be dynamically changed: loaded, enabled, disabled without interrupting the system.
- Are specified for groups of objects, often before objects are instantiated.

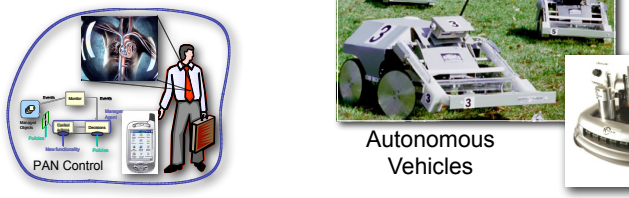
Different Policy Types

- **Obligations** define which operations need to be performed when certain events occur. **Event-Condition-Action** Rules
- **Authorisations** define which operations are permitted and under which circumstances.
- Other policy types: Membership management, Information Filtering, Trust Management, Delegation, Negotiation, etc.

Policy-based closed adaptation loop

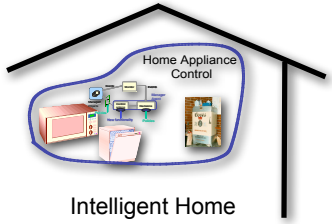


Pervasive Spaces



Personal Area Networks

Autonomous
Vehicles



Intelligent Home
Networks

Pervasive
Environments

A common pattern

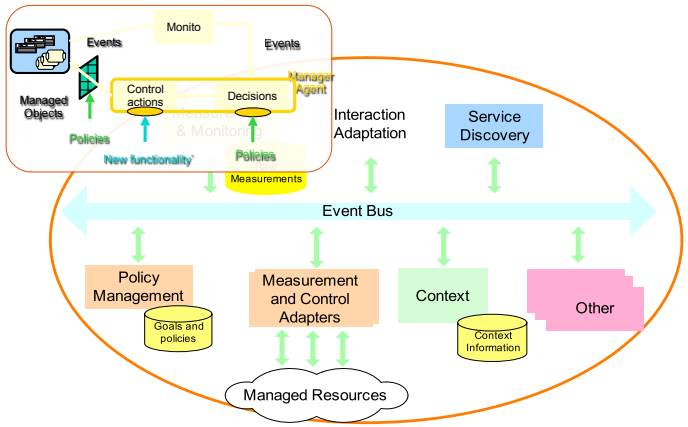
- That can be used at different levels of scale: body area networks, unmanned vehicles, intelligent homes, and large distributed systems and networks.
- That can provide self-management and closed-loop adaptation at the local level.
- That can provide different levels of functionality.
- That is architectural as well as functional.
- Provides low-coupling between the different services.

The Self-Managed Cell (SMC) and its Core Services

What is a Self-Managed Cell?

- A set of hardware and software components forming an administrative domain that is able to function autonomously and thus capable of self-management.
- Management services interact with each other through asynchronous events propagated through a content-based event bus.
- Policies provide local closed-loop adaptation.
- Able to interact with other SMCs and able to compose in larger scales SMCs.

Self-Managed Cell (SMC)

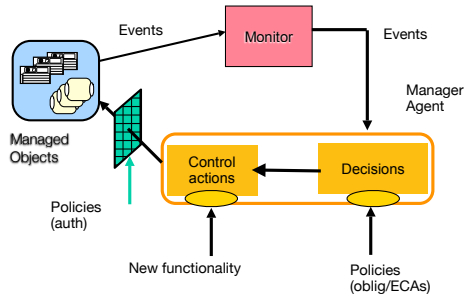


SMC Pattern

- Provides low-coupling between the different services.
- Permits the use of different service implementations when used at different levels of scale.
- Permits to add services to SMCs in order to add functionality:
  - Context service(s) for mobile users and gathering information from the environment.
  - Authentication, Access Control and other security services.
  - Provisioning and Optimisation services for control of resources

SMC Core Services

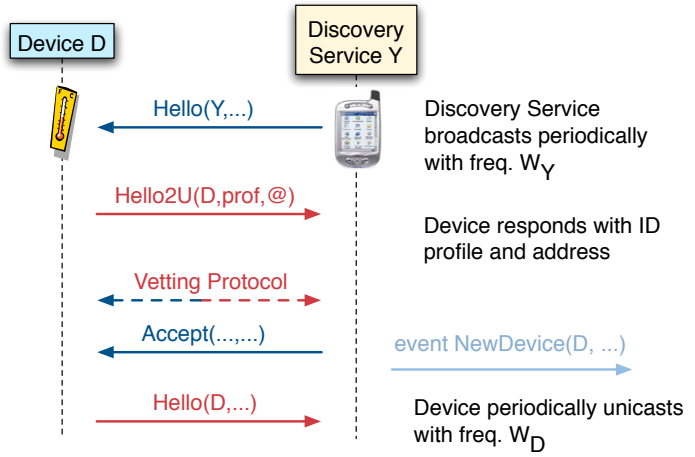
- Discovery Service (including membership management)
- Event Service
- Policy Service



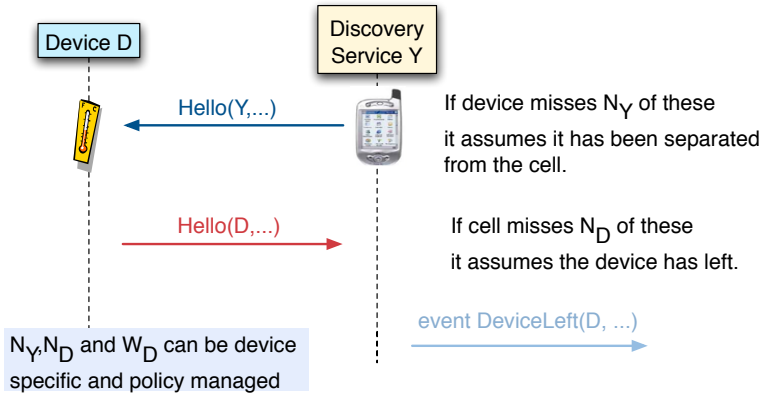
Cell Discovery Service

- **Discovers** new devices and **maintains membership**.
- Queries device for its profile and services;
- **Performs** vetting functions e.g. authentication, **admission control**.
- Listens for new service offers and service removals from the devices
- Generates **events** to signal new/disconnected devices or software components. Interested services can subscribe, receive and react to these events.

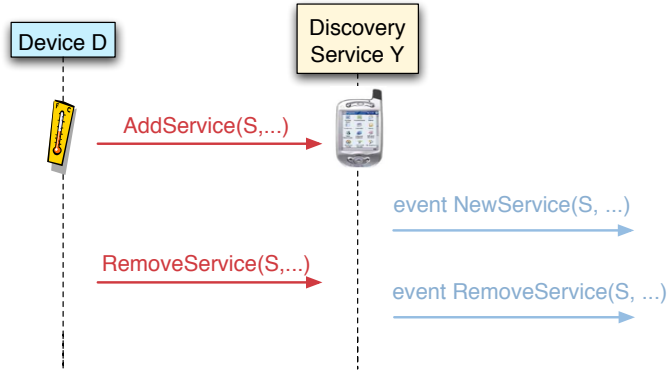
Discovery Service I



Device Discovery - Separation



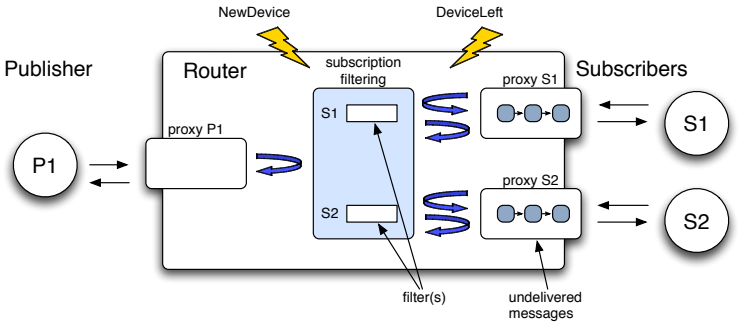
Service/Component Discovery



Cell Event Service

- Publish/Subscribe with content based router.
- At-most-once, reliable event delivery.
- To an individual recipient events are delivered in the same order as received by the router.
- Quenchable publishers to minimise number of messages and power consumption.
- Supports heterogeneous communication.

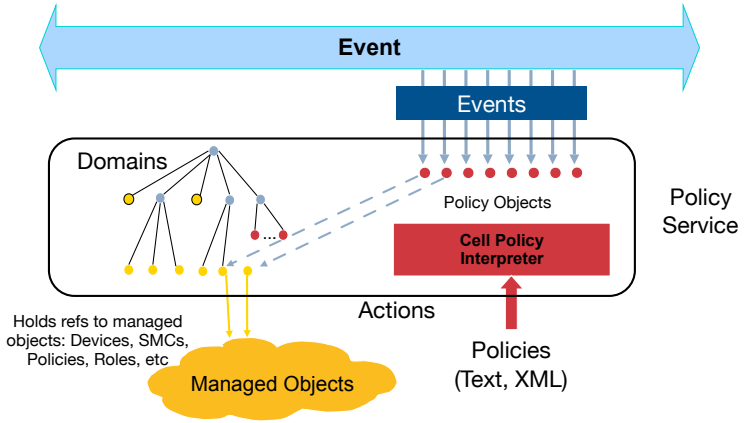
Event Service Architecture



Ponder2

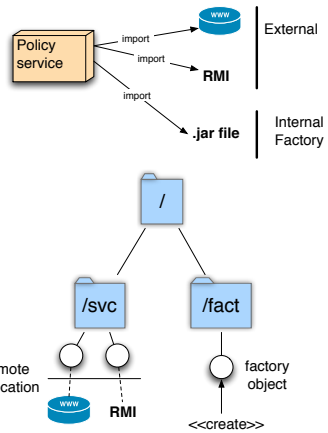
- Supports both obligation policies in the form of Event-Condition-Action rules and authorisation policies. Therefore it requires:
  - **Managed Objects** to represent resources and invoke operations on external services
  - **Domains** to group objects and specify policies in terms of domains of objects.
  - **Events** to trigger policies and interactions with the event bus.
  - **Object invocations** to implement policy actions

Ponder2 Policy Service



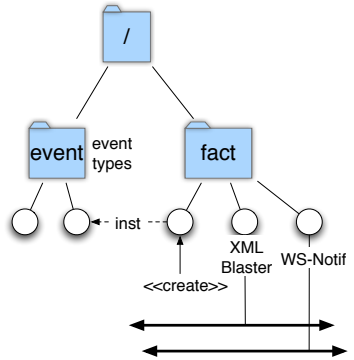
Managed Objects

- A **managed object**
  - Conforms to a set of interface rules.
  - Created through a factory
  - Accept commands
- General purpose object management environment.
- Four pre-defined types of managed objects: **domains**, **policies**, **factories**, **external**

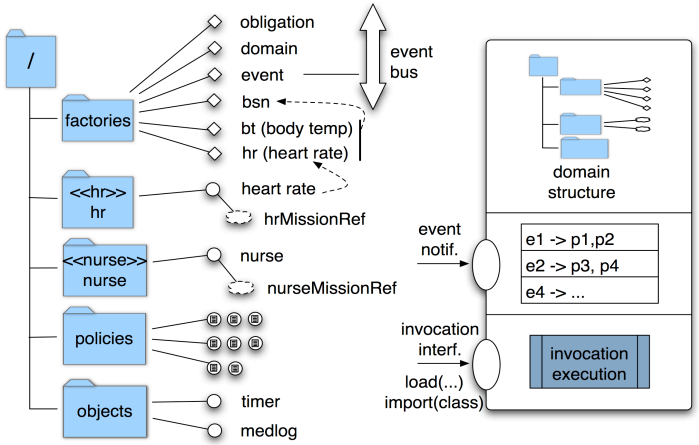


Events in Ponder2

- Event = notification with named attributes.
- Trigger policies.
- Can integrate with one or several external event buses through adapter objects.



Ponder2 Policy Service - II



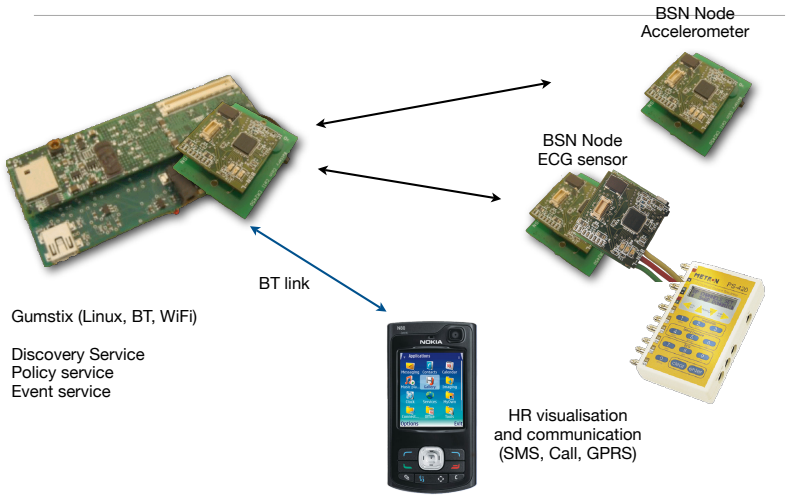
Policies for Different Functional Areas

- **Device and Service Discovery.** How to react to new devices and services and their disappearance.
- **Membership Management**.
- **Context Management.** How to react to changes in location, activities of the user, surrounding environment.
- **Clinical Management.** How to react to changes in the clinical condition.
- **Security Management.**
- **Policy Management.** Enable, disable, unload policies.

SMC Policies

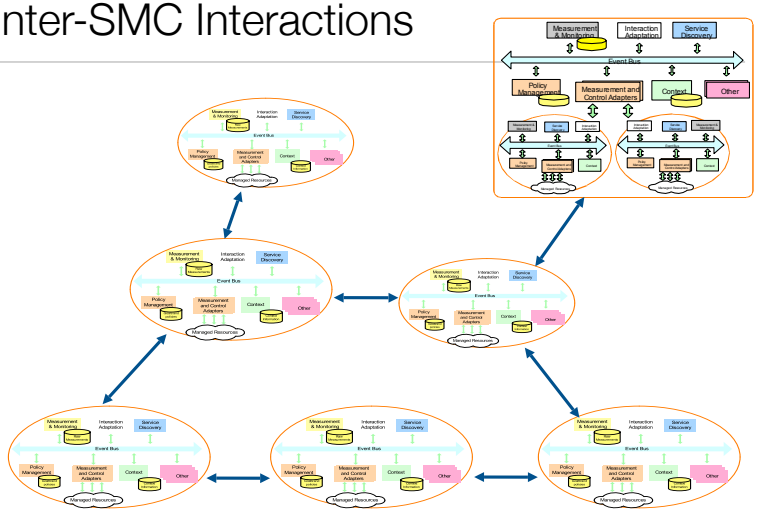
```
on new_component(id, profile, addr) do
  if profile == “heart rate” then
    r = /fact/hr.create(profile, addr); /sensors.add(r)

on hr(level) do
     if level > 100 then /sensors/os.setfreq(10min);
                         /sensors/os.setMinVal(80)

on context(activity) do
  if activity == “running” then
    /policies/normal.disable(); /policies/active.enable()

auth+ /patient → /os.{setfreq, setMinVal, stop, start}
auth+ /patient → /policies.{load, delete, enable, disable}
```

Heart Monitoring Demo

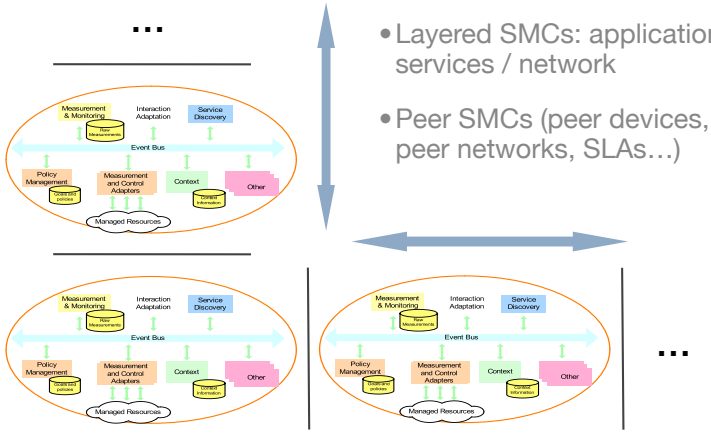


Interactions Between SMCs

Inter-SMC Interactions



Peer-to-Peer Interactions

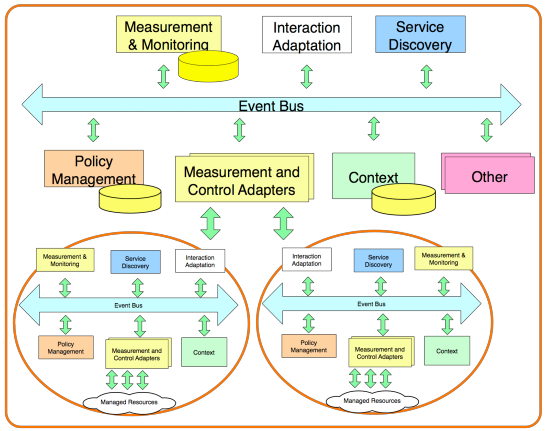


- Layered SMCs: application / services / network
- Peer SMCs (peer devices, peer networks, SLAs…)

SMC Composition

The internal SMCs cease to advertise themselves externally.

The enclosing SMC programs the nested SMCs



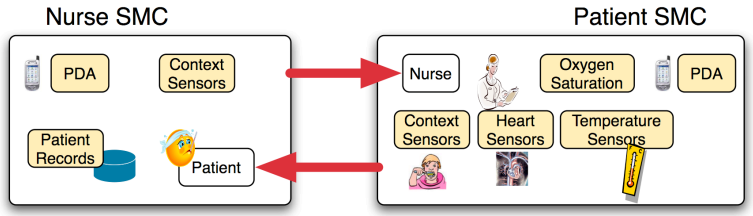
SMC Interactions: Requirements

- Despite apparent differences both peer-to-peer and composition interactions require similar support:
  - **actions:** SMCs need to invoke actions on other SMCs e.g. to access device readings, actions specified as part of policies.
  - **events:** SMCs need to exchange events i.e. both publish and subscribe to events in a remote SMC
  - **policies:** SMCs need to exchange policies e.g. ask a remote SMC to react to events in a particular way

SMC Interactions: Differences

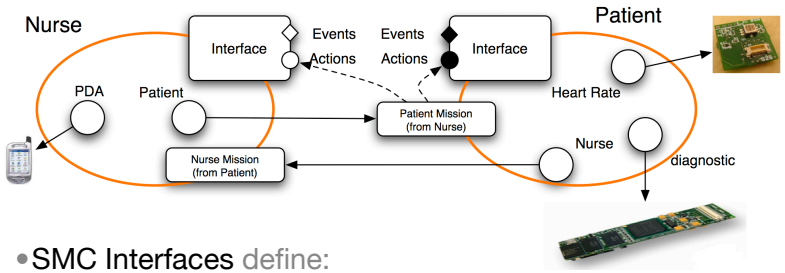
- Authorisations typically permit more actions to a parent SMC then to a peer SMC.
- A SMC comprising other SMCs encapsulates them:
  - The inner SMCs cease to advertise themselves independently.
  - All access to inner SMC is mediated through the outer SMC.
  - A composed SMC will have a single parent
- SMCs do not lose their autonomy.

SMCs discovery



- On SMC discovery, each SMC assigns discovered SMC to pre-defined domains.
- Policies for domain apply to assigned SMC.
- SMC Discovery can also result in policy-exchange and sharing of events and services.

SMC Missions: Policy Exchange

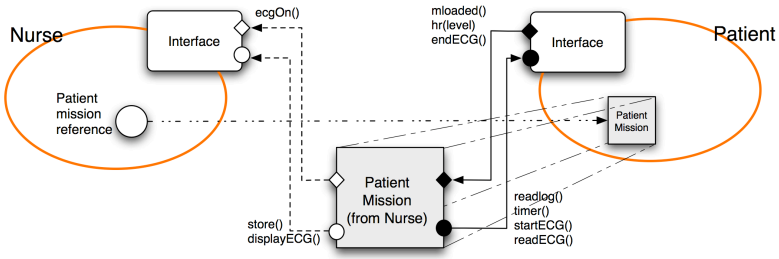


- SMC Interfaces define:
  - **events**: that can be raised by an SMC
  - **notifications**: that an SMC can receive
  - **actions**: that can be invoked on the SMC

Policy Exchange II

```
mission patientT(nurse, patient, ECGlevel, ECGTime) do
    on patient.mloaded() do
      nurse.store(patient.readlog())
    on patient.hr(level) do
      if level > ECGlevel then
          patient.startECG()
          patient.timer(ECGTime, endECG())
          nurse.ecgOn()
    on patient.endECG() do
      nurse.display(patient.readECG())
```

SMC Missions: Policy Exchange

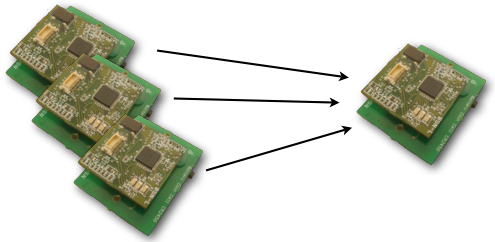



```
auth+ /nurse → /patient.loadMission    // at the Patient
auth+ /patient → /nurse.store           // at the Nurse
auth+ /patient → /nurse.displayECG
on newPatient(p) do
   ref = p.loadMission(/patients.interface, p.interface, 82, 40);  /
    roles[p].add(ref)
```

Measurements and Performance

IEEE 802.15.4

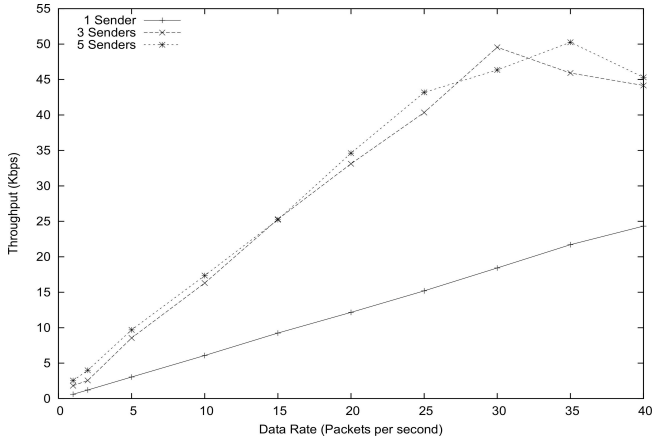
- Claims maximum bandwidth of 250Kbps



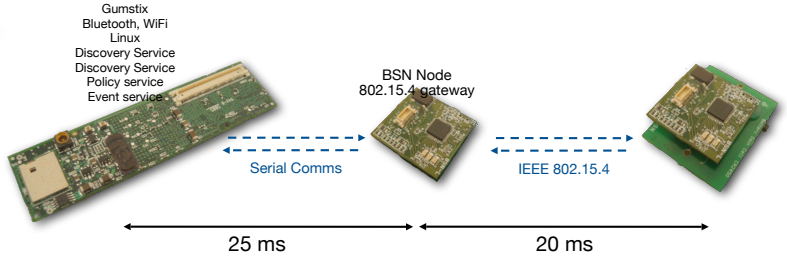
1 hop away
1 packet = 76B
Rate varies:
1-40 packets/s

- Observed max. throughput 50Kbps. At this rate the receiver's packet queue fills up and packets are dropped

IEEE 802.15.4 throughput



End-to-end Delay

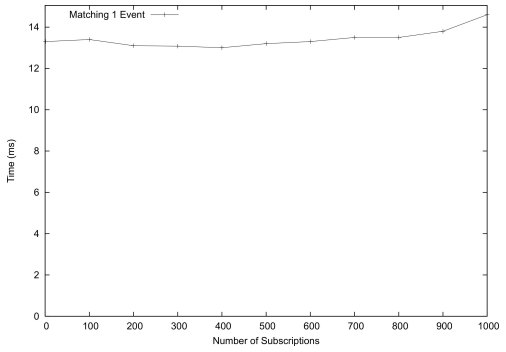


Discovery

- expected 110ms (2 serial + 3 * 802.15.4 packets)
- observed 129ms
- End-to-end: 144ms; includes
  - discovery handshake
  - generation of *new_component* event
  - event proxy and managed object creation

Event Service

- Subscription matching: 13-15ms



Policy Service

- Policy Object: 3.214 kB includes policy type, triggers, actions and constraints
- Simple policy execution (null action): 13.57ms
- Simple policy execution action issued to BSN: 23.88ms
- Simple policy execution + simple condition: 30.05ms
- End-to-end: event published to proxy to policy execution: 46.05 ms

Observations

- Use of XML generates significant overhead in terms of both memory consumption and run-time processing.
- This despite using a small footprint and efficient parser.
- Performance suitable for body-area network for self-management purposes.
- Not always suitable for application data e.g., ECG 200Hz
- Processing and adaptation capability on sensor

Summary

- Common Architectural Pattern applied at different levels of scale.
- Content-based filtering event bus provides flexibility and de-coupling between services.
- Policies for Adaptation and Access Control.
- Composition and P2P interactions across Cells
- Implementation Status
  - Event, discovery and policy service - Gumstix and PDAs
  - Event and discovery clients + basic policy interpreter on BSN nodes.

Ongoing and Future Work

- Ponder2 (next release):
  - concise language for command execution and policy creation; XML will be generated
  - automated generation of Managed Object from annotations on Java source
- Formal Semantics of SMC and interactions behaviour
- Higher level abstractions for SMC interactions
- Applications to unmanned vehicles and cityware environments

Acknowledgements

Imperial College London


Kevin Twidle


Morris Sloman

Sye-Loong Keoh


Alberto Schaeffer

Naranker Dulay

UNIVERSITY of GLASGOW


Joe Sventek


Stephen Strowes


Steven Heeps

References

- Ponder2 Policy Service: http://www.ponder2.net
- E. Lupu, N. Dulay, M. Sloman, J.Sventek, S. Heeps, S. Strowes, K. Twidle, S.-L. Keoh, A. Schaeffer-Filho. AMUSE: Autonomic Management of Ubiquitous e-Health Systems. *Concurrency and Computation: Practice and Experience*, John Wiley and Sons, Inc., 2007 (To Appear).
- Keoh, S.L., Twidle K., Pryce, N., Schaeffer-Filho, A E., Lupu, E., Dulay, N., Sloman, M., Heeps, S., Strowes, S., Sventek, J., and Katsiri, E. Policy-based Management for Body-Sensor Networks. 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007), AAchen, Germany, March 2007.
- Russello, C. Dong, and N. Dulay. Authorisation and Conflict Resolution for Hierarchical Domains. IEEE Workshop on Policies for Distributed Systems and Networks (Policy), Bologna, Italy, June 2007.