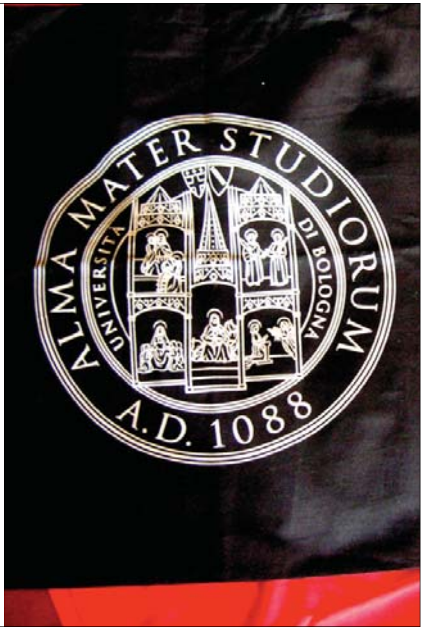
Autonomous Pervasive Systems and the Policy Challenges of a Small World!

Emil Lupu
Imperial College London

University of Bologna

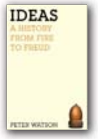
- Oldest University in Europe (certainly the oldest medieval).
- Born out of conflict: the papal-imperial rivalry, restrictions put by the church on learning and in particular on common law.
- Lack of protection of non "citizens" leads to the formation of guilds ("universitas").



Policy at Bologna

- In essence a school of law.
- A university ran by the students. Policy (Bologna University style):
  - *Doctors elected by students.*
  - *Curriculum must be agreed by the students.*
  - *Curriculum must be divided into two-weekly puncta.*
  - *Doctors who start lectures late or finish late must pay a fine.*
  - *Doctors who fail to attract at least 5 students are deemed absent and fined.*
  - *Doctors must pay a deposit before being allowed to leave the city to ensure their return.*

Peter Watson
Ideas: A history from Fire to Freud
Phoenix Publ. 2005



Policies are for Large Systems

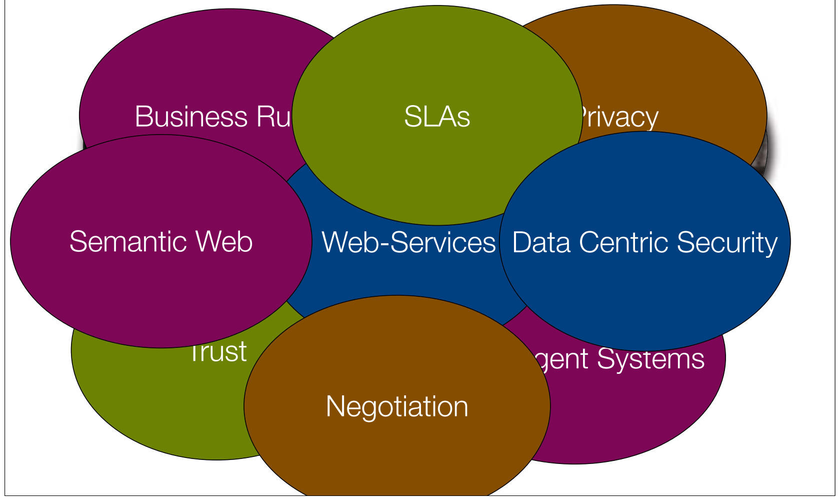
Policies

- Originally introduced to separate the strategy for resource allocation in OSs from the mechanisms controlling the resources.

  R Levin et al. Policy/Mechanism Separation in Hydra. 5th Symp. on Operating Systems Principles (SOSP), November 1975.

- Became popular in large centralised access control systems and subsequently, in the early 90's, for managing large networks and distributed systems.
  - Policies apply to large sets of objects providing uniform configuration.
  - Provide the means to automate adaptation across large systems

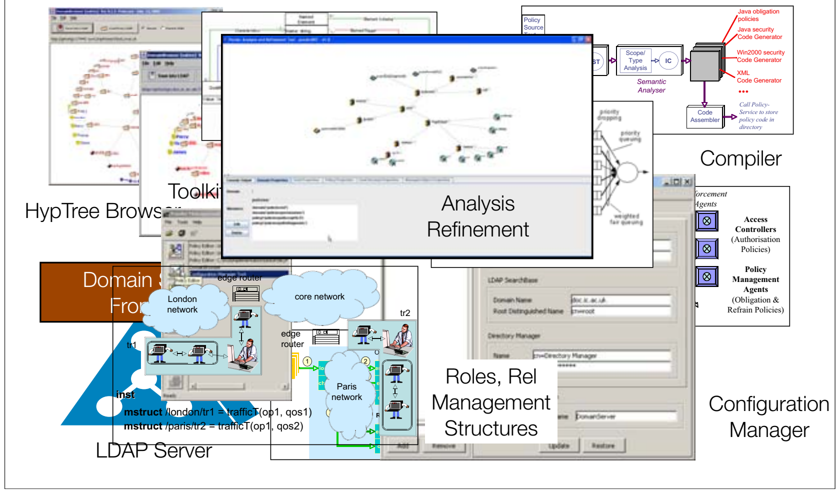
Policy Areas



Policies for Large Systems require Complex Policy Systems

- Build on complex software infrastructure: CIM, LDAP, Storage, Databases, Web-Services (WS-*), Grid-Environments, ...
- Systems are functionally separated. A function realised for the entire system e.g., Authentication, Fault-Diagnostics, Accounting, ...
- Architectures are tightly coupled, making in difficult and laborious to add new elements.
- Computational power is infinite (or almost). Components are always available
- Policies are replacing human actions.

Examples: Ponder



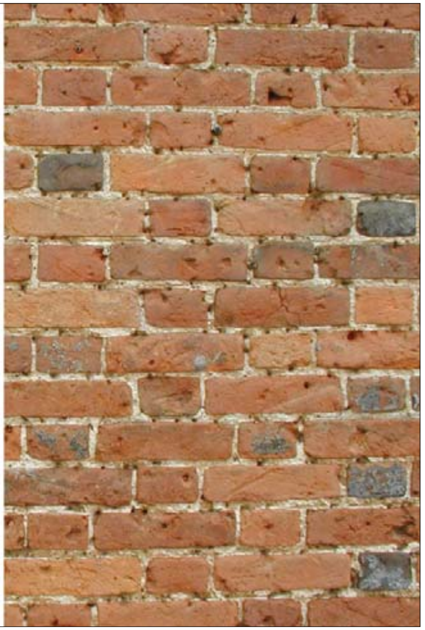
```
inst
mstruct /london/tr1 = trafficT(op1, qos1)
mstruct /paris/tr2 = trafficT(op1, qos2)
```

Lessons

- Development intensive requiring numerous services that depend on many underlying systems and packages. Must be able to rely on commercial policy products ... which aren't there.
- Difficult to maintain, distribute and demonstrate. Numerous queries received about the Ponder toolkit were about LDAP installation and configuration.
- Difficult to integrate with new techniques: planning, context, analysis, security and management ...
- Policies replace human (administrator) led activity. Typically compared with scripting and ad-hoc human-driven solutions. Poor short term ROI.
  Need to provide "advantage": analysis, refinement and validation.
  Need to provide benchmarking and proof of scale up.

Policy Outset

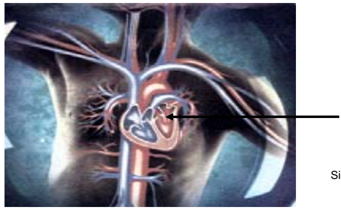
- Policy motivated by arguments of scale
- Industry cannot deliver the products and benchmarks
- Academics cannot deliver convincing demonstrations
  - Restrict to theoretical work.
  - Small proof of concept for individual techniques.

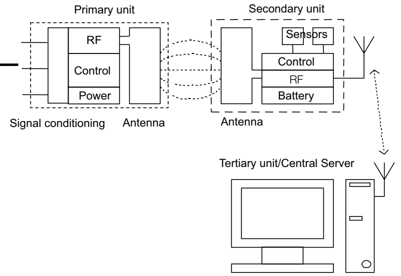


Autonomous Pervasive Systems ... at any scale

Cardiac Monitoring

UbiMon Body Sensor Node

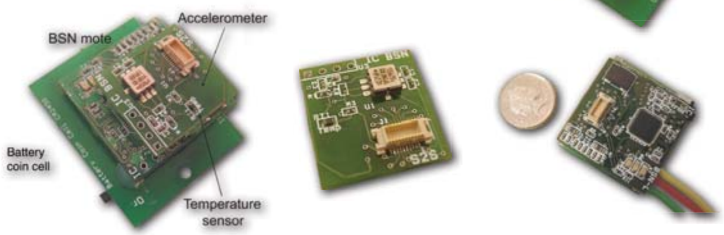




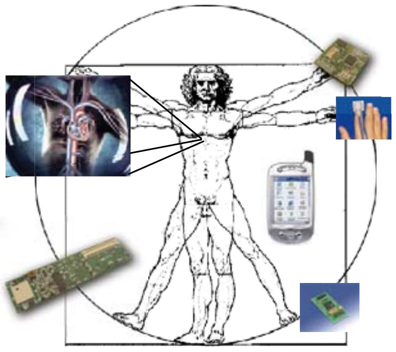
UbiMon
www.ubimon.net

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



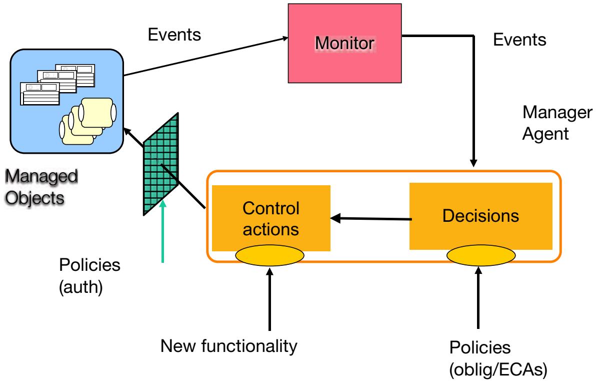
Body Area Networks

- Implanted and wearable sensors: Heart monitoring, blood-pressure, oxygen saturation, etc.
- Continuous monitoring of physiological condition e.g., cardiac arrhythmia.
- Maintenance of chronic conditions: heart deficiencies, diabetes mellitus, chronic anaesthesia
- Incremental drug delivery. Context dependent drug delivery.
- Remote interrogation
- Alert for emergency interventions.

Requirements

- 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 devices in different environments: home, hospital, GP clinic
- Minimal resource (power) consumption
- No administrator interactions
- Low-coupling
- Support for Interactions
  - peer-to-peer interactions between devices
  - composition between subsystems
  - federation between collections of devices
- Decision making: goal-driven, heuristics, utility
- Learning: classification, statistical, declarative

Policy-based closed adaptation loop

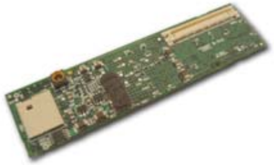


Policies in Healthcare Environments

- 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.
- Policies applied to different functional areas: device and service discovery, device configuration, authentication and authorisation, privacy, collaborations, ...

The Controller: Gumstix

- 200-400MHz (Intel XScale PXA255)
  16 MBFlash
  Bluetooth
- Expansion boards: Wifi, Eth, Cf or MMC, audio, GPS
- Linux 2.6
- GCC, JamVM and other development tools
- 802.15.4 through connected BSN

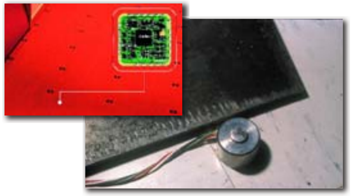


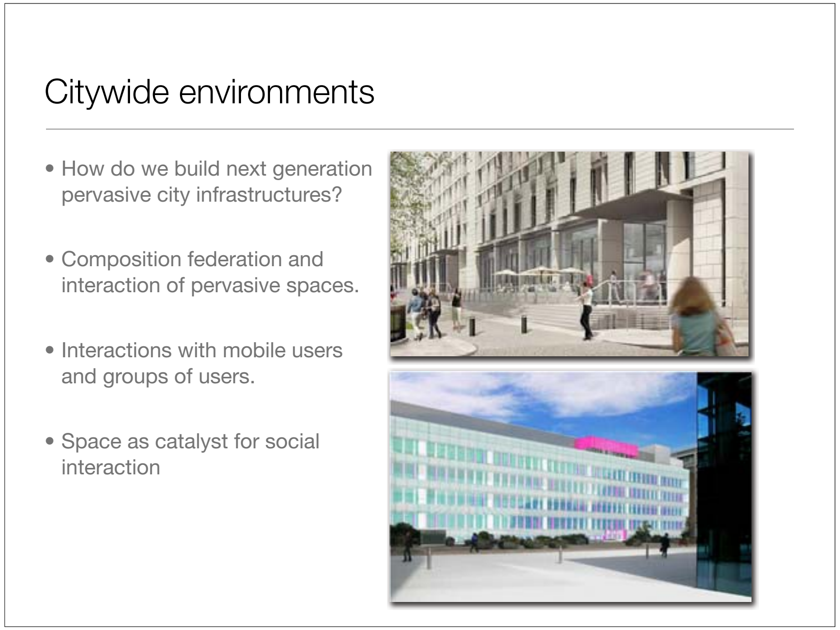
Autonomous Unmanned Vehicles

- Each vehicle is an autonomous collection of managed devices with different functional capabilities
- Must be extensible to different sensors and modules
- Can aggregate and collaborate in fleets of autonomous vehicles
- Must interact with external environment

Building Integration

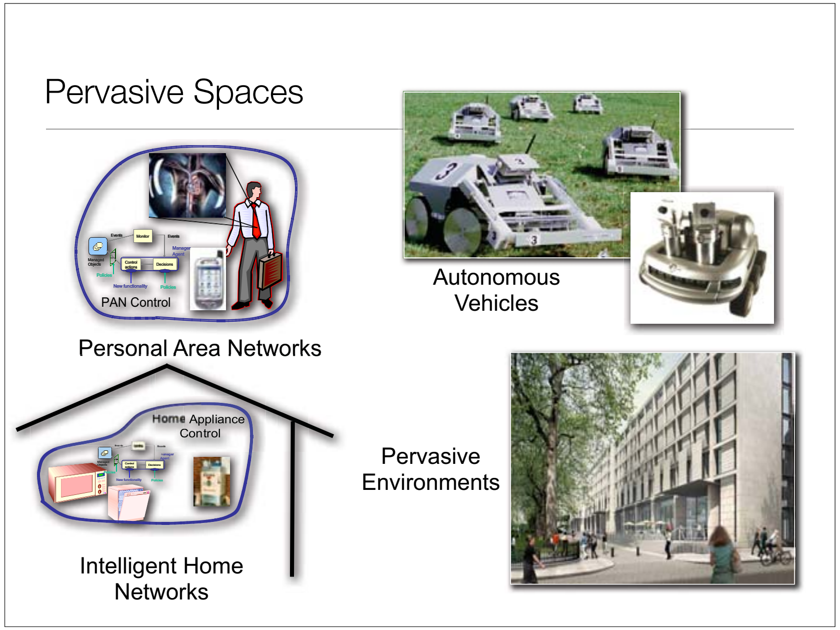
- Instrumentation of in-door environments: multimedia, assisted living for the elderly
- Discovery and autonomy across nested collections of devices
- Interactions with persons (and their personal area networks)
- Composition and federation that follows: physical space, functional space



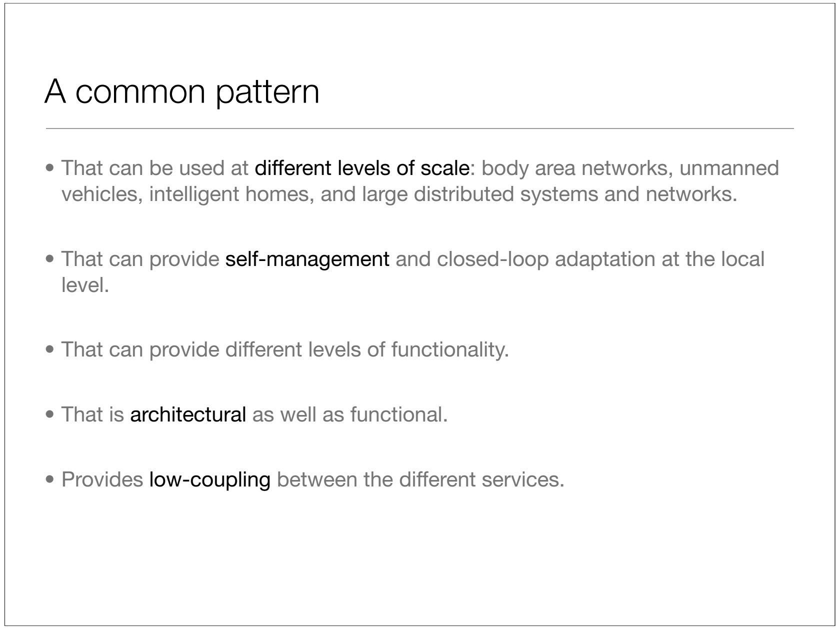


Citywide environments

- How do we build next generation pervasive city infrastructures?
- Composition federation and interaction of pervasive spaces.
- Interactions with mobile users and groups of users.
- Space as catalyst for social interaction

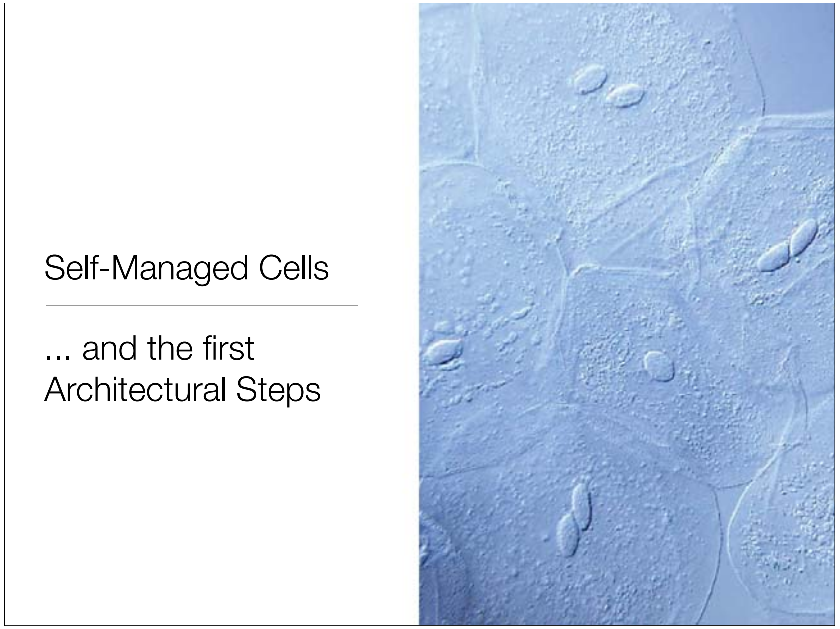


Pervasive Spaces



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.



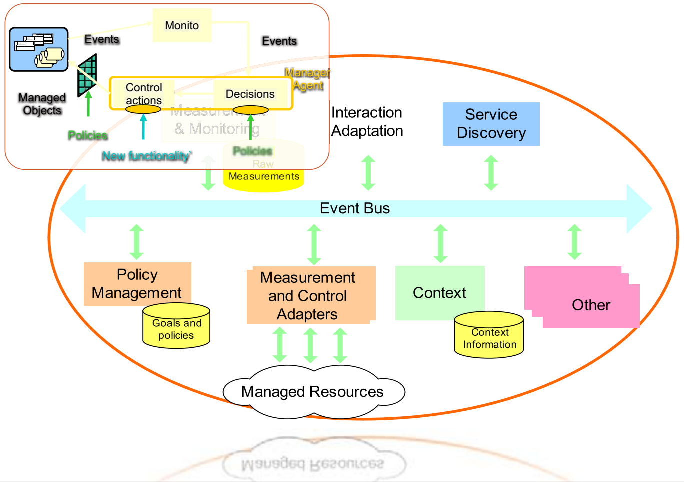
Self-Managed Cells

... and the first Architectural Steps

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)

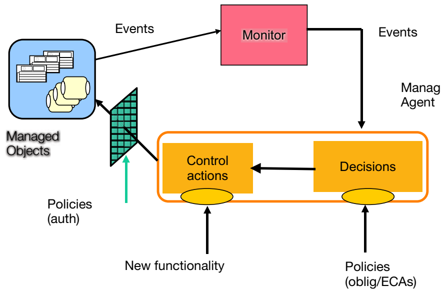


SMC Pattern

- 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



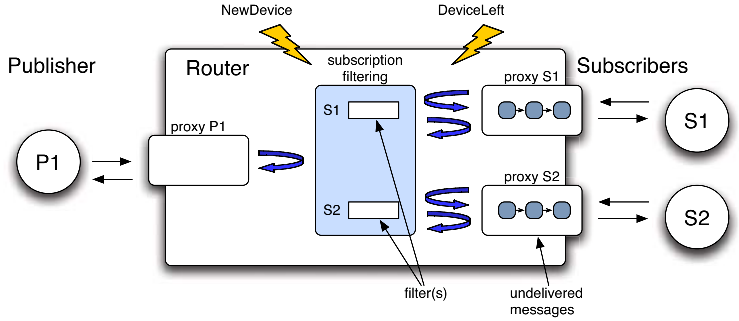
Cell Discovery Service

- **Discovers** new devices and **maintains membership** to detect failures and departures from cell.
- 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.
- Own implementation developed to cater for BSN nodes and policy configurable parameters but other protocols e.g., SDP, SLP, ... could be used in other environments.

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



Policy Service: Ponder^2
... the same, yet very different

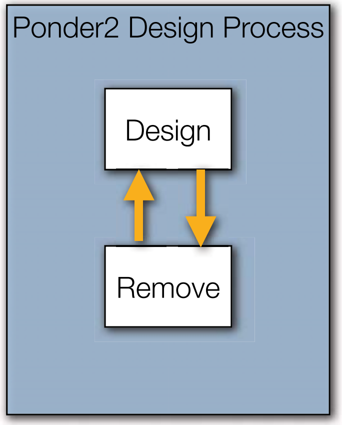
http://ponder2.net

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.

Ponder2 Design Goals

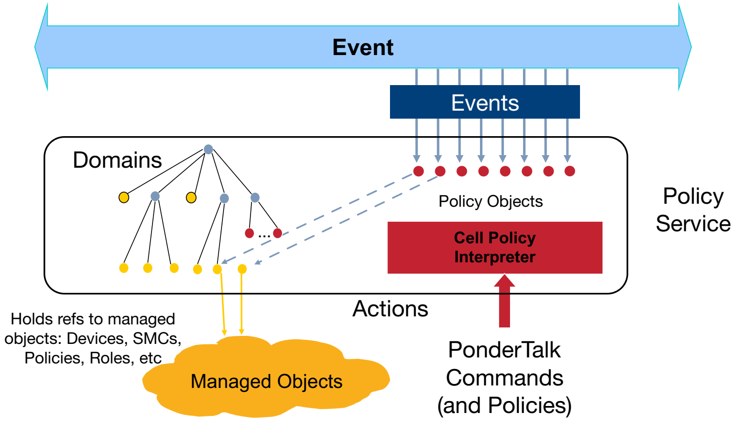
- Permit interaction with a running SMC
  - invoking operations on objects
  - policy creation, activation, etc.
- Only loads what is needed
- Can be extended (dynamically)
- Must run on a Gumstix (and possibly on BSN nodes)



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.
  - Policies of multiple kinds.
  - Object invocations to implement policy actions

Ponder2 Policy Service


Event
Events
Domains
...
Policy Objects
Cell Policy Interpreter
Policy Service
Actions
Holds refs to managed objects: Devices, SMCs, Policies, Roles, etc
Managed Objects
PonderTalk Commands (and Policies)

Ponder2, try again

- The Policy Service requires:
  - Convention for loading and creating Managed Objects
  - Invoking operations on Managed Objects
  - Root domain (that does not know it is a domain)
- That's it!
- Domains, policies, events, ... are themselves managed objects that follow the same conventions.

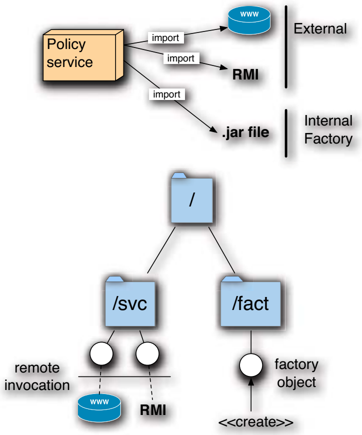
Bootstrapping Ponder2 in PonderTalk

- SMC is just an empty domain - root
- Import domain factory
- Create domains
- Import basic factories
- Read more PonderTalk

```
// Bootstrap code for Ponder2

// Import the Domain code
// and create the default domains
domainFactory := root load: "Domain".
root
   at: "factory" put: domainFactory create;
   at: "policy" put: domainFactory create;
   at: "event" put: domainFactory create.

// Put the domain factory into the factory directory
root/factory at: "domain" put: domainFactory.

// Import event and policy factories
root/factory
   at: "event" put: ( root load: "EventTemplate" );
   at: "oblig" put: ( root load: "ObligationPolicy" ).
```

Managed Objects

- A managed object
  - Conforms to a set of interface rules.
  - Created through a factory
  - Accepts commands
- Several pre-defined types of managed objects: domains, policies, factories, external, events

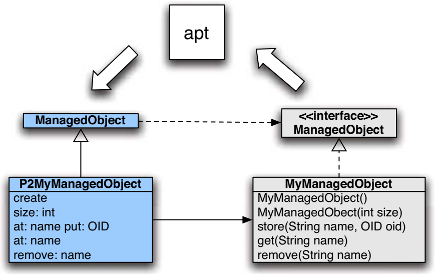


Writing a new Managed Object

- A Managed Object is a Java class
- PonderTalk messages converted to method calls
- Constructors called by factory messages
- Instance methods called by operational messages
- Mapping done by @Ponder2op Java annotation
  - uses apt Annotation Processing Tool instead of javac

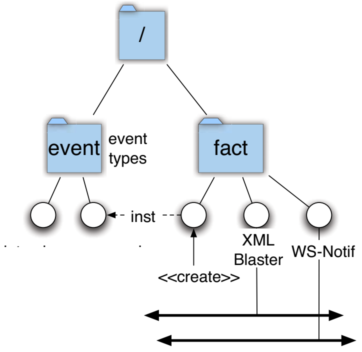
Example: a HashTable Managed Object

```
public class MyManagedObject
         implements P2ManagedObject {

  private Map<String, OID> data;

  @Ponder2op("create")
  MyManagedObject() {
    data = new HashMap<String, OID>();
  }

  @Ponder2op("size:")
  MyManagedObject(int size) {
    data =
       new HashMap<String, OID>(size);
  }

  @Ponder2op("at:put:")
  OID store(String name, OID oid) {
    data.put(name, oid);
    return oid;
  }

   @Ponder2op("at:")
  OID get(String name) {
    return data.get(name);
  }

  @Ponder2op("remove:")
  OID remove(String name) {
    return data.remove(name);
  }

}
```



Events in Ponder2

- Event = notification with named attributes.
- Trigger policies.
- An event factory interfaces with an external event bus.
- Multiple factories can be used.
- Event types (templates) are created by factories.



Example: Discovery of new BSN sensor

- Discovery service issues events when BSN is detected or lost

```
newevent := root/factory/SMCeventbus.

// newBSN event type

root/event
     at: "newBSN"
     put: (newevent create: #("name" "type") ).

// example of raising an event
root/event/newBSN
          create: #("Temp1" "TempMon").
```

Policies

- Created with policy factory
- Dynamically associate events, actions and conditions with a policy
- Can be activated and deactivated
- Are managed objects. Can be moved, deleted, created, activated, deactivated by other policies
- Actions and conditions are blocks

Blocks

- Blocks are objets that group statements.
- Block execution is delayed
- Blocks can take arguments
- Blocks are closures
- Blocks return the result of their last statement

Discovery Policies

- When a new BSN sensor is discovered a policy is used to create the appropriate adaptor managed object
- Adaptor object acts as proxy for the BSN and can receive commands for them e.g. setrate

```
// Create discovery policy
newpolicy := root/factory/oblig.

discBSN := newpolicy create.
discBSN
   event: root/event/newBSN;
   action: [ :name :type |
            root/template/bsnAdaptor
                   create: name
                   setActive: type
           ];
   setActive: true.
```

Blood Pressure Policy

- on bp(value)

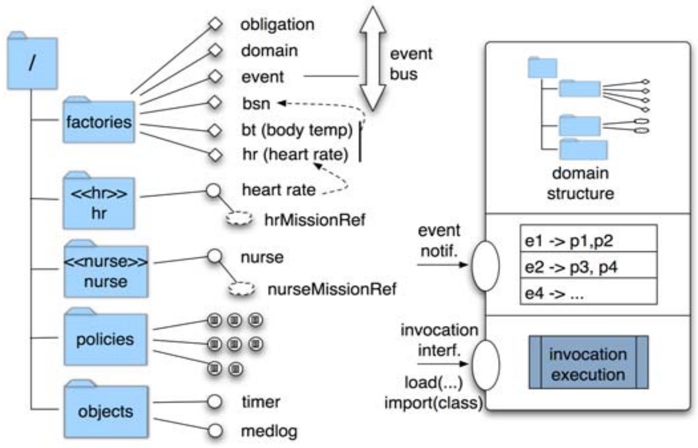
  if (value>150)
   && oldValue<=150

- do
  /bsn/HEART1
   .set(sensorRate=1)

  /alarm.alarm(on)
  /alarm.show

```
// Create blood pressure policy
newpolicy := root/factory/event.
newevent := root/factory/ecapolicy.

bphigh := newpolicy create.
bphigh
  event: (newevent create: #("name", "newVal", "oldVal")
  condition: [ :name :newVal :oldVal |
                 name == "BP1"
                 && ( newVal > 150 )
                 && ( oldVal < 150 ) ];
  action: [
       root/bsn/HEART1  setRate: 0.1.
       root/alarm setAlarm: true; show ];
  setActive: true.

root/policy at: "bphigh" put: bphigh
```

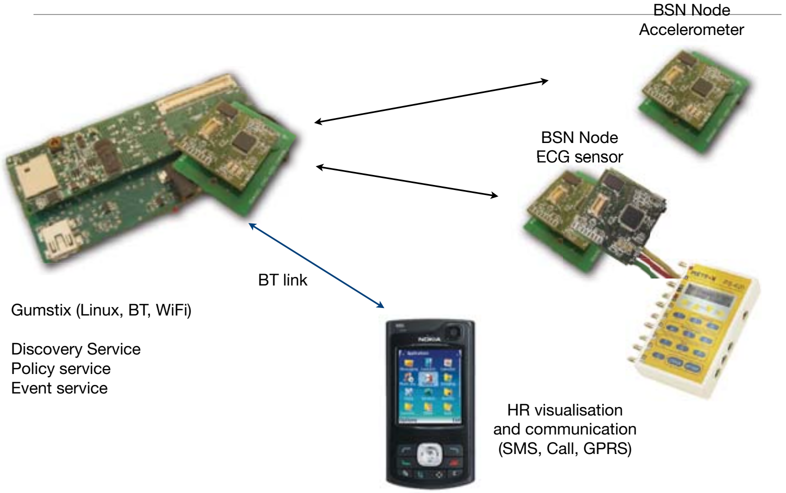
Ponder2 Policy Service - II



Not yet quite a policy language

```
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



BSN Node
Accelerometer

BSN Node
ECG sensor

BT link

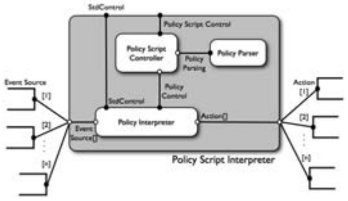
Gumstix (Linux, BT, WiFi)

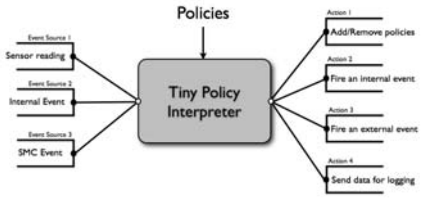
Discovery Service
Policy service
Event service

HR visualisation
and communication
(SMS, Call, GPRS)

How small should an SMC be?

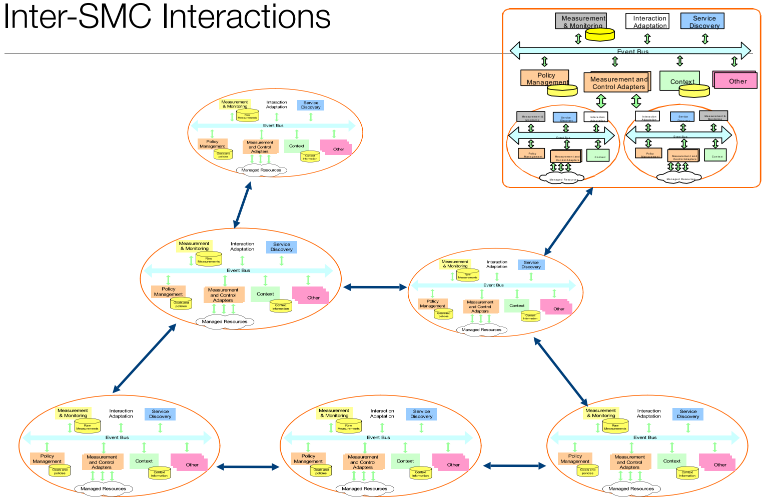
- Is a BSN node an SMC?
- 6 analog sensor channels
- Event based interactions
- Need for policy-based adaptation



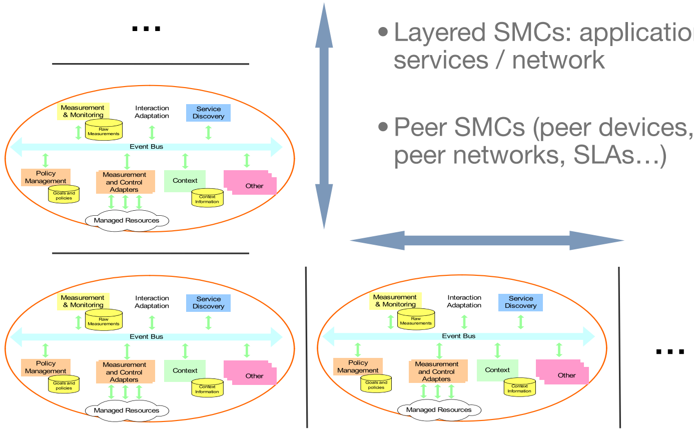


Interactions Between Self-Managed Cells

Inter-SMC Interactions

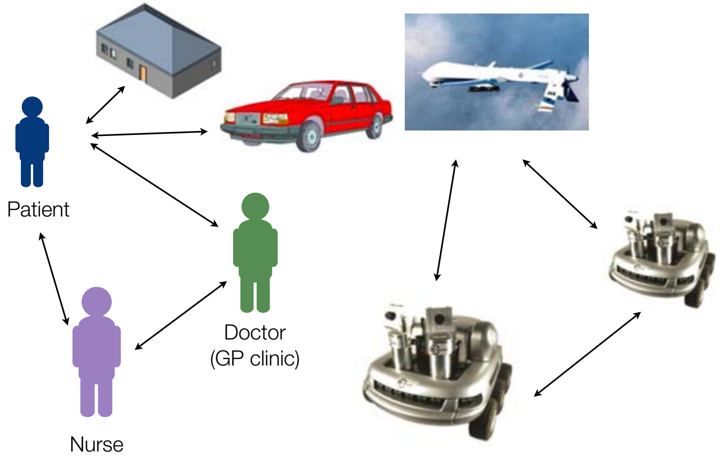


Peer-to-Peer Interactions



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

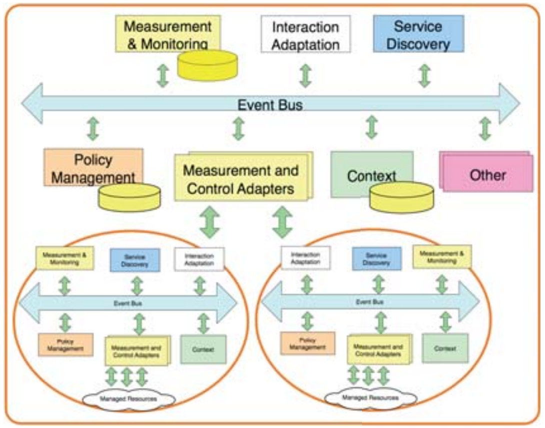
Peer to Peer Interactions



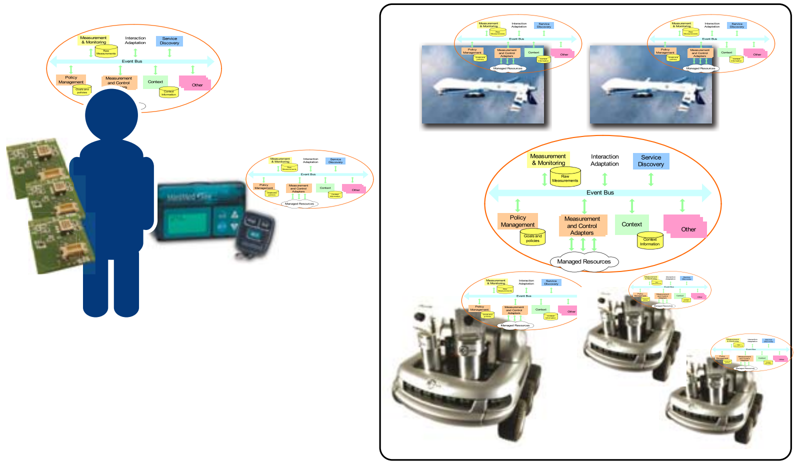
SMC Composition

The internal SMCs cease to advertise themselves externally.

The enclosing SMC programs the nested SMCs



Composition Interactions



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

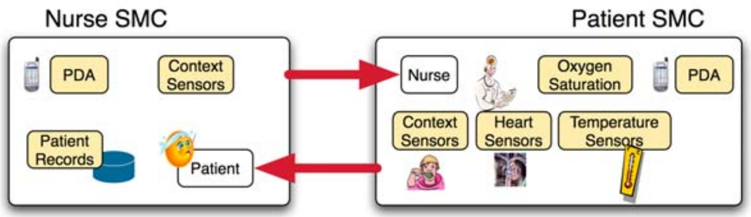
Interactions and Autonomy

- Each SMC must retain autonomy over its resources:
  - It must decide which functions (services) to export
  - It must retain decision on whether to export (bind) any of its internal resources externally
  - It may mediate external interactions to internal managed objects e.g., for filtering and parameter adaptation.
  - It decides which policies to accept (allowing “full access” may jeopardise integrity).
- Applicable in both p2p and composition

Differences: p2p - composition

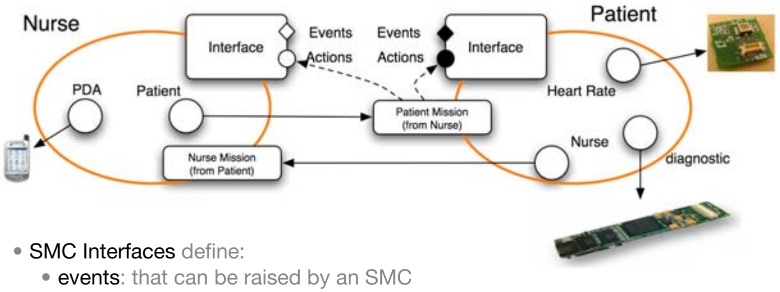
- Once bound as a resource in a composition relationship:
  - The SMC ceases to advertise itself
  - The SMC does not establish other p2p or composition relationships unless directed by the outer SMC
  - “administrative” interfaces are guaranteed to be bound to a single outer SMC.
- Events and services exposed to other SMCs will be different in composition and p2p relationships.
- Policies (i.e., missions) accepted will be different

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

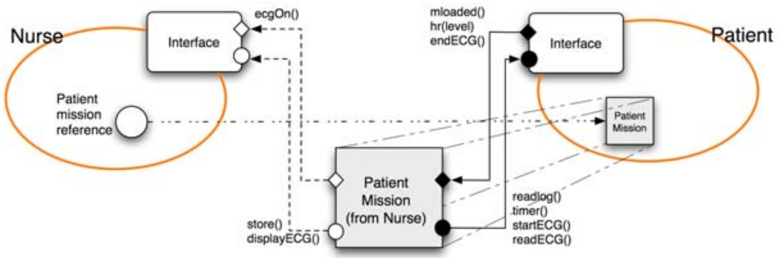


- **SMC Interfaces** define:
  - **events**: that can be raised by an SMC
  - **notifications**: that an SMC can receive
  - **actions**: that can be invoked on the SMC

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

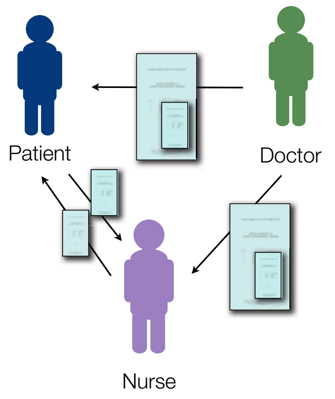


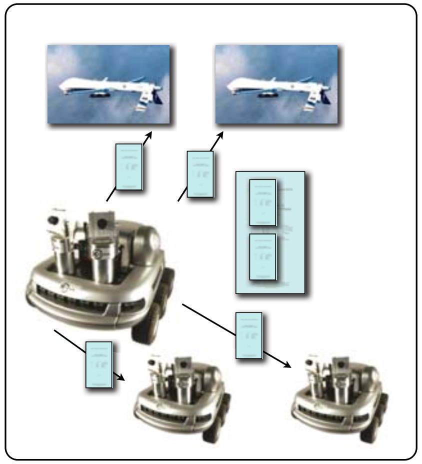
```
auth+ /nurse → /patient.loadMission      // at the Patient
auth+ /patient → /nurse.store             // at the Nurse
auth+ /patient → /nurse.displayECG
on newPatient(p) do
   ref = p.loadMission(/patients.interface, p.interface, 82, 40);  /
    roles[p].add(ref)
```

Interaction Procedure

- Discover SMCs
- Decide what kind of interaction to create (for both SMCs)
- Decide on role assignment
- Exchange Interfaces
- Perform role assignment and creation of managed object
- Decide which missions to instantiate and instantiate them.

Missions in Multi-Party Interactions

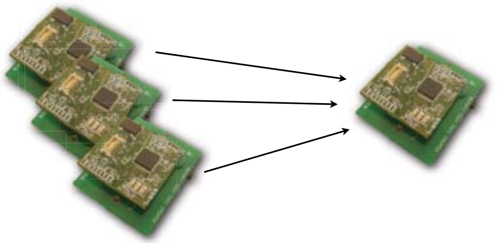




Measurements and Performance

IEEE 802.15.4

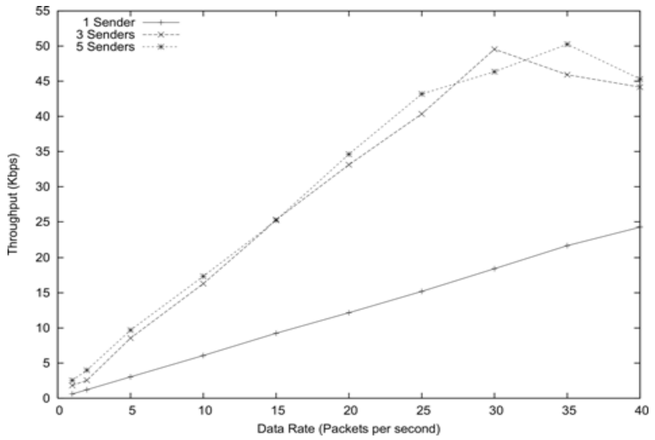
- Claims maximum bandwidth of 250Kbps



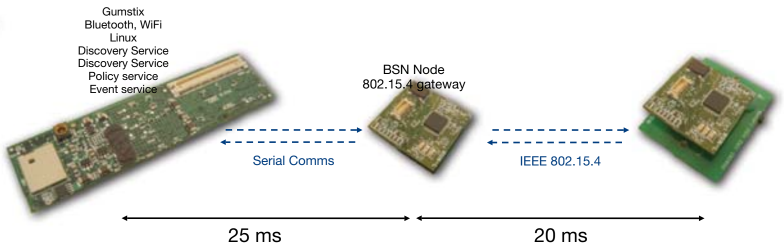
1 hop away
1 packet = 76B
Rate varies:
1-40 packets/s

- 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

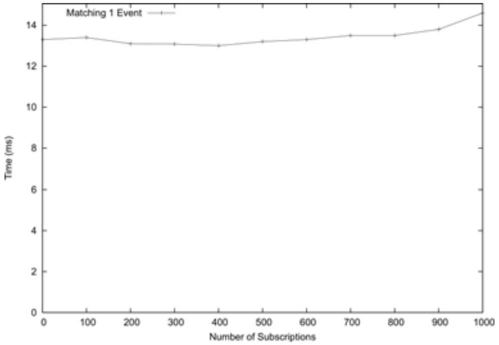


Discovery

- 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



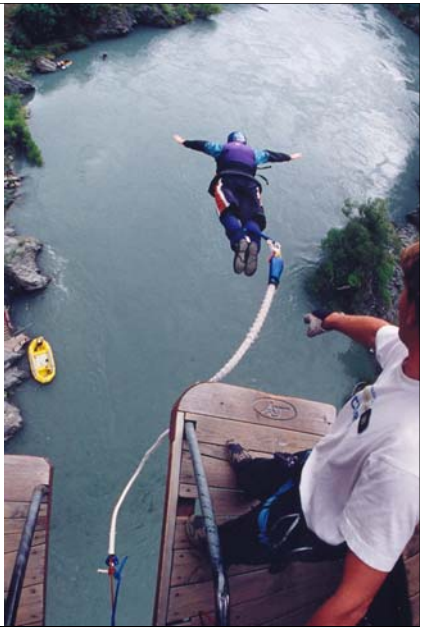
Policy Service

- 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

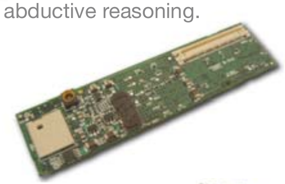
- Use of XML generates significant overhead in terms of both memory consumption and run-time processing.
- This despite using a small footprint and efficient parser.
- Performance suitable for body-area network for self-management purposes.
- Not always suitable for application data e.g., ECG 200Hz
-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

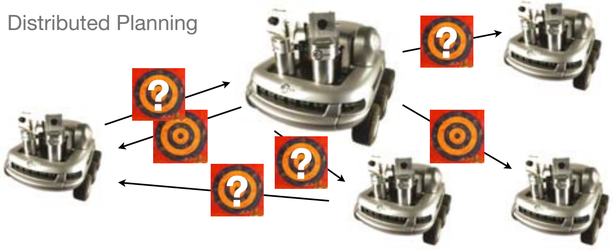
Challenges



Reasoning and Planning

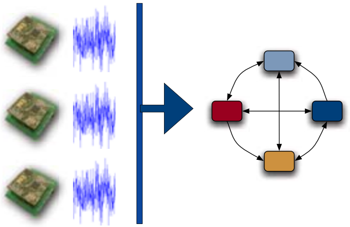
- Analysis and Refinement work to date relies on abductive reasoning.
- Can we do abductive reasoning on a Gumstix?
- Yes with a bit more memory!
- Planning and Distributed Planning

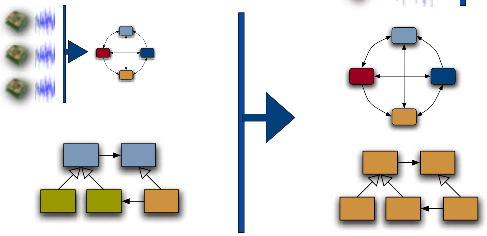




Making Sense of the Surrounding World

- Much work is directed towards abstracting sensor input in higher level “state” information. Accelerometers, Temperature, Sound, ...
- Buy why should be always starting from scratch...

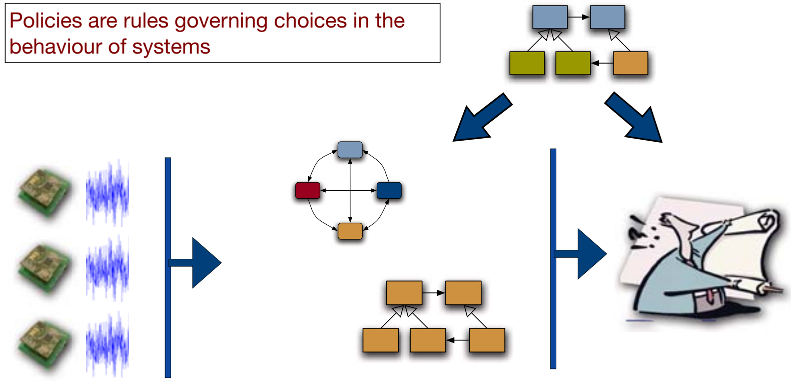




Can we use acquired information to refine Context Models?

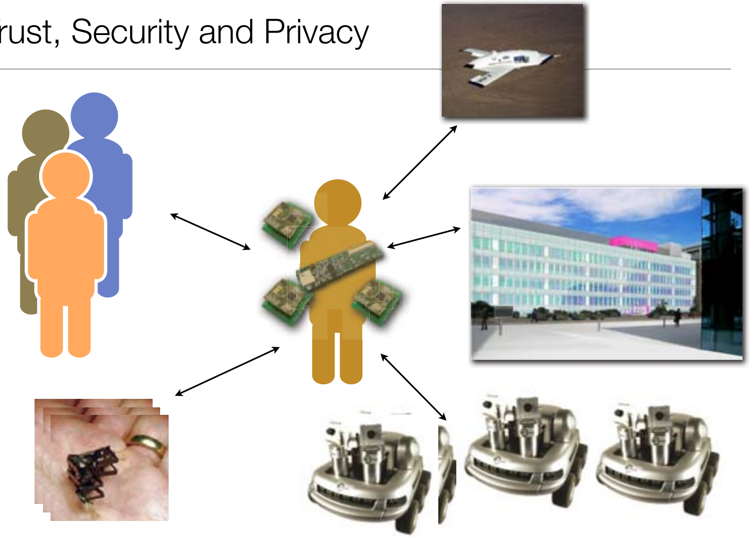
Making Sense of Behaviour

Policies are rules governing choices in the behaviour of systems



Policies are rules learnt from the behaviour of ...

Trust, Security and Privacy



Conclusions

- SMC defines a common architectural pattern that can be applied at different levels of scale.
  - Content-based filtering event bus provides flexibility and de-coupling between services.
  - Ponder2 provides support for general object management and policies
- In contrast to policies in large systems, designs in autonomous pervasive computing strive to be simple. Scale is achieved through extensibility, modularity and composition of autonomous components.
- Realising autonomous pervasive systems requires the integration of multiple techniques from different areas of computing: operating systems, distributed systems, statistical decision methods, AI, DAI, multi-agent systems, knowledge engineering, ...
- ... on a small scale!

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.