Imperial College London

Starfish: Policy Driven Self-Management in Wireless Sensor Networks

THEMIS BOURDENAS, MORRIS SLOMAN

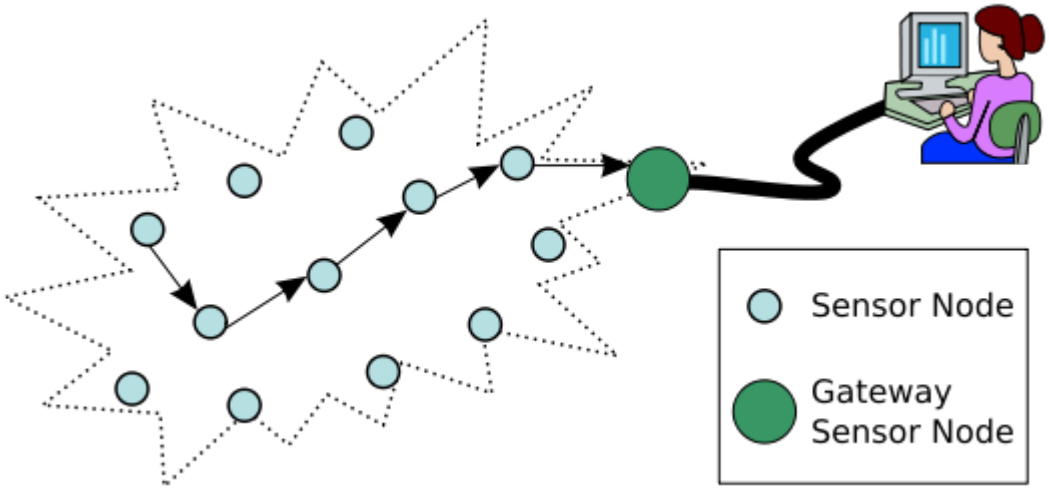
Agenda

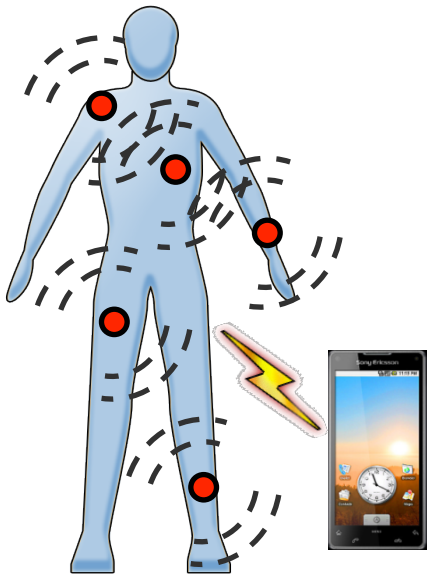
- Wireless Sensor Networks
- Fault Management
- Starfish platform
- Conclusions

Wireless Sensor Networks

- feedback component in pervasive computing

- broad range of application requirements
  - *setup* – small (body) to vast (landscape) networks
  - *accessibility* – easily to impossible to retrieve nodes
  - *power* – usually battery powered
  - *capabilities* – very limited CPU/memory





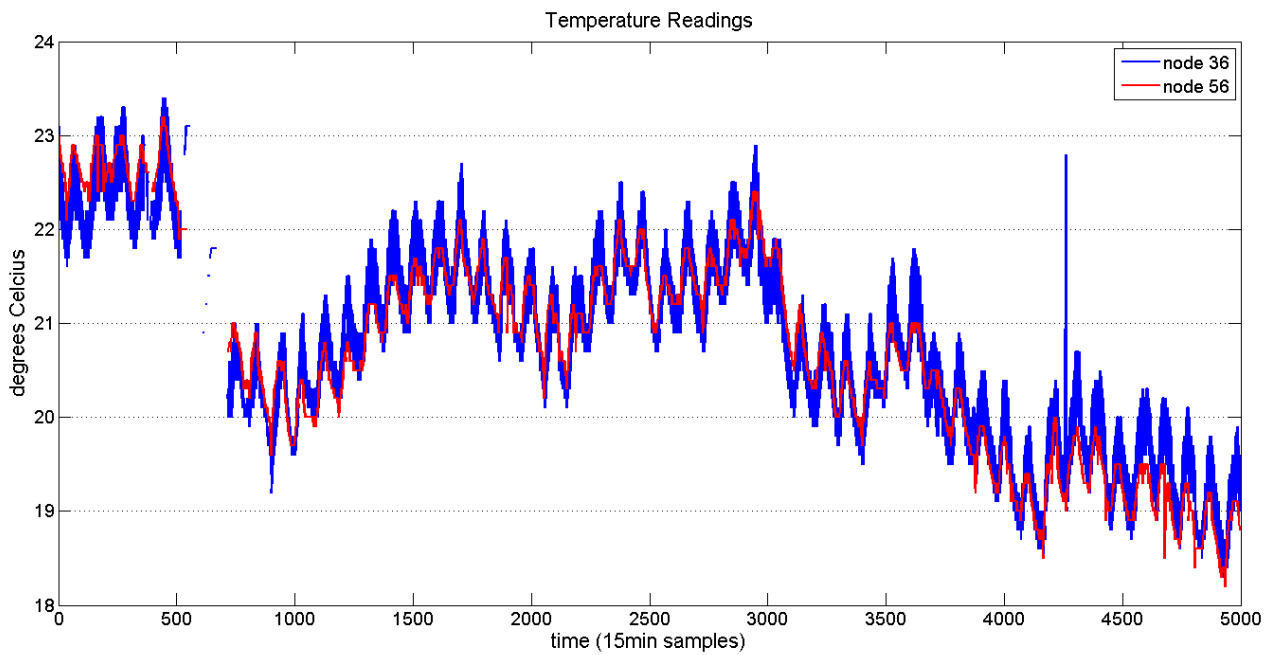
Motivation

- faults in WSNs are not exceptional cases
  - inaccurate sensors
  - conflicting / missing readings
- causes
  - sensitive / inexpensive electronics
  - exposed to physical world / users
  - low battery levels
  - circuitry fouling
- impractical management even for small networks

Imperial College London

Motivation

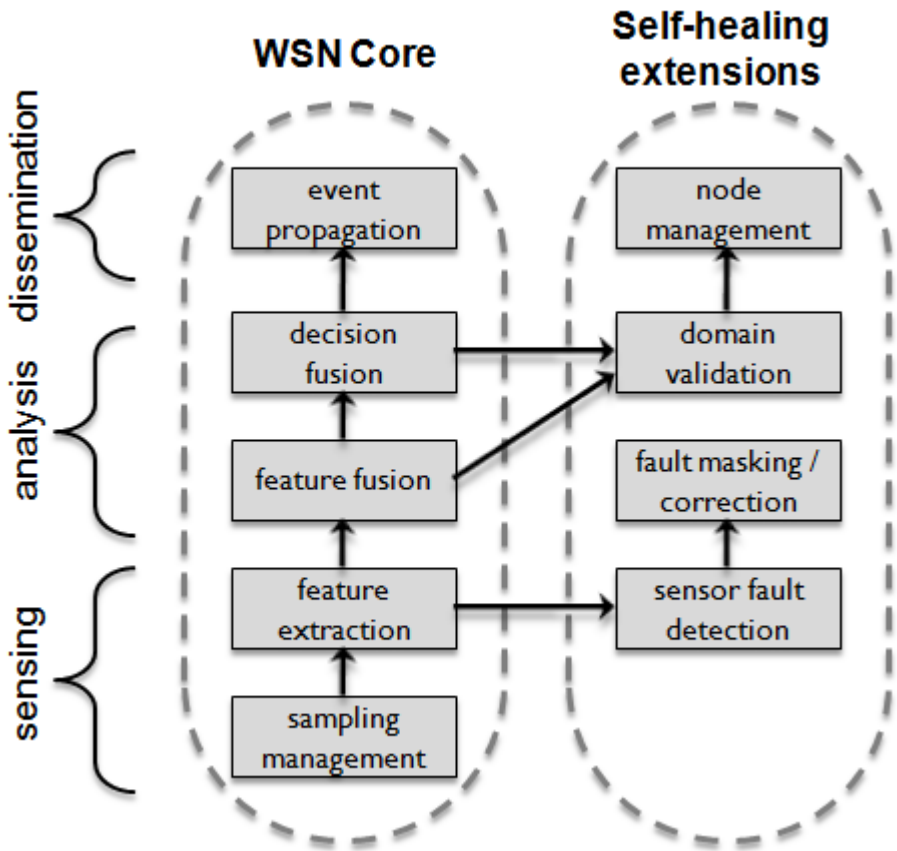
- fault examples
  - spikes, noise, drift, missing data



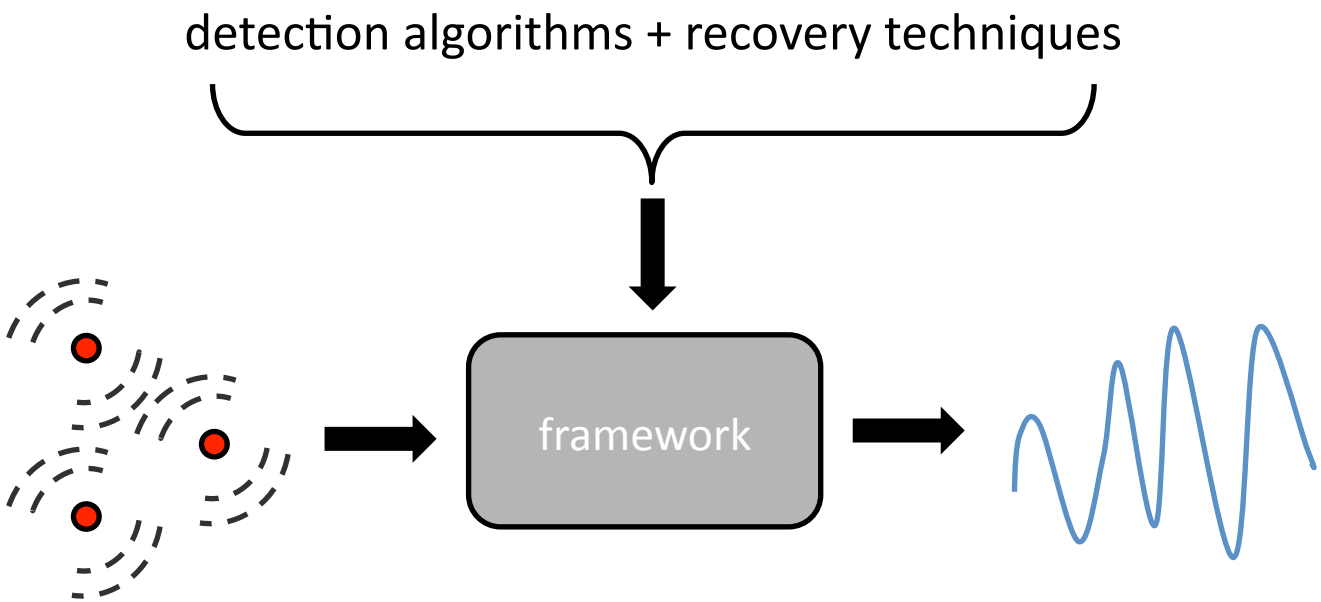
Imperial College London

WSN Architecture

- high level architecture
  - functional roles
  - deployment agnostic
- attaching self-healing services
  - in different levels and scope



Self-healing Framework



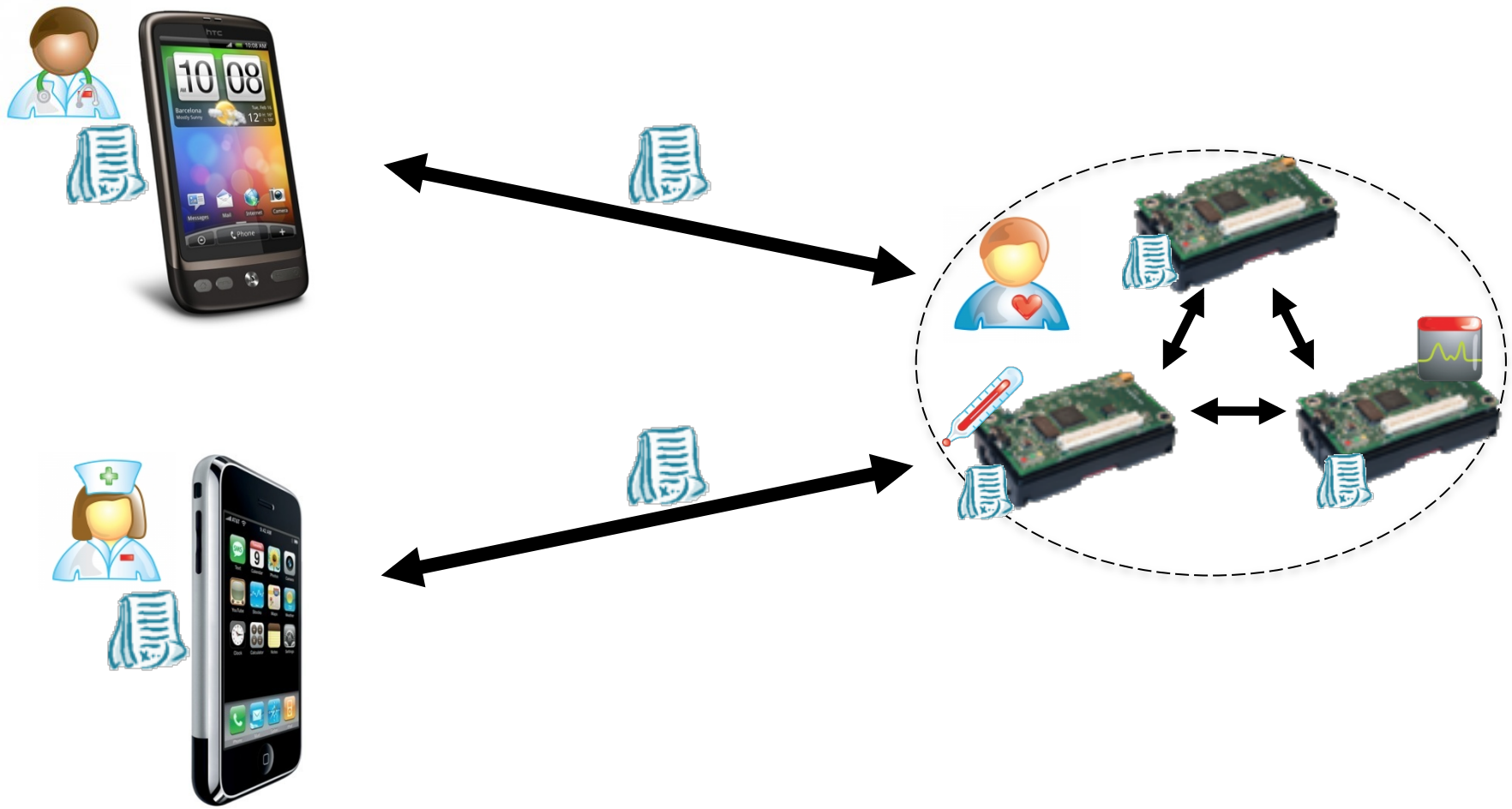
algorithms + framework

Starfish Platform Overview

- Finger2 embedded policy system
  - mote middleware
- Starfish Module Library
  - common programming facilities
- Starfish Editor
  - authoring and management tool

Imperial College London

Roles, Missions and Modules



Finger2 Policies

def ‹*authpolicy*›[+/-]
  subject ‹*role*›
  target ‹*role*›
  if ‹*condition*›
  action ‹*name*›

def ‹*obligpolicy*›
  on ‹*event*›
  if ‹*condition*›
  do ‹*action*›

Imperial College London

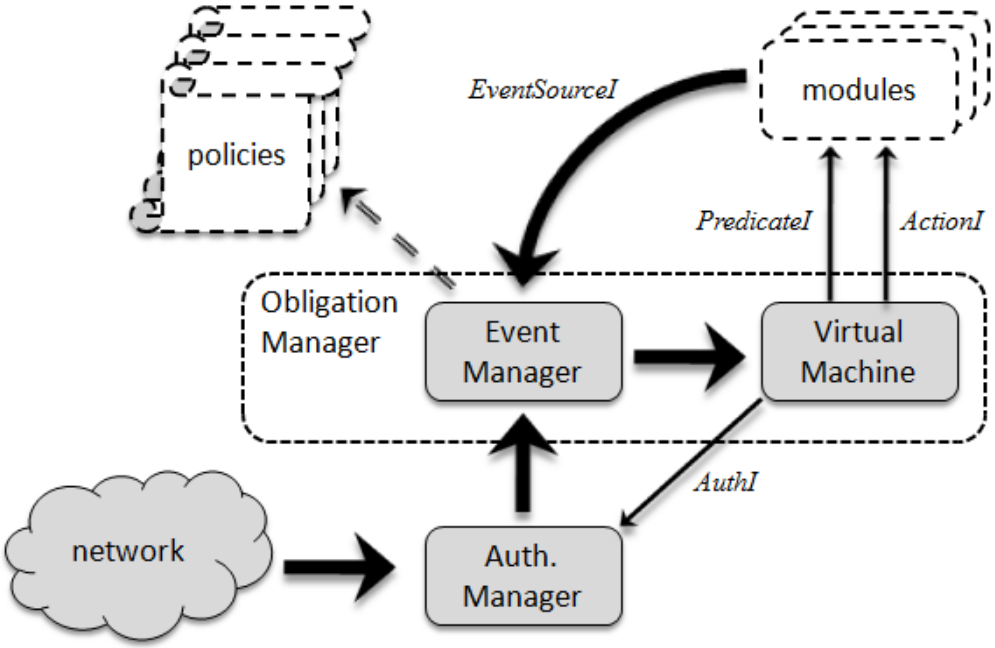
Finger2 Policies

```
def ‹authpolicy›[+/-]
   subject ‹role›
   target ‹role›
   if ‹condition›
   action ‹name›

def ‹obligpolicy›
   on ‹event›
   if ‹condition›
   do ‹action›
```

```
def allow_nurse_policy_install +
   subject nurse
   target patient
   if power.Level() > .2
      and nurse.type is staff_nurse
   action policy.Install

def ECG_request
   on gui.RequestUpdate(patient, type)
   if network.IsAvail(patient) and type is ECG
   do patient.policy.Install(ECG_update)

def ECG_update
   on sensor.Reading(type, value)
   if type is ECG
   do network.Send(nurse, value, timer.Now())
```

Imperial College London

Finger2 Architecture



Starfish Module Library

- **sensing**
  - sync / async reads from sensors
- **buffering**
  - storage on motes
- **feature extraction**
  - statistical functions data preprocessing
- **timers**
  - periodic scheduling, notifications
- **communication**
  - network and serial port communication primitives
- **policies / missions / roles**
  - installation and dynamic management of roles and policies

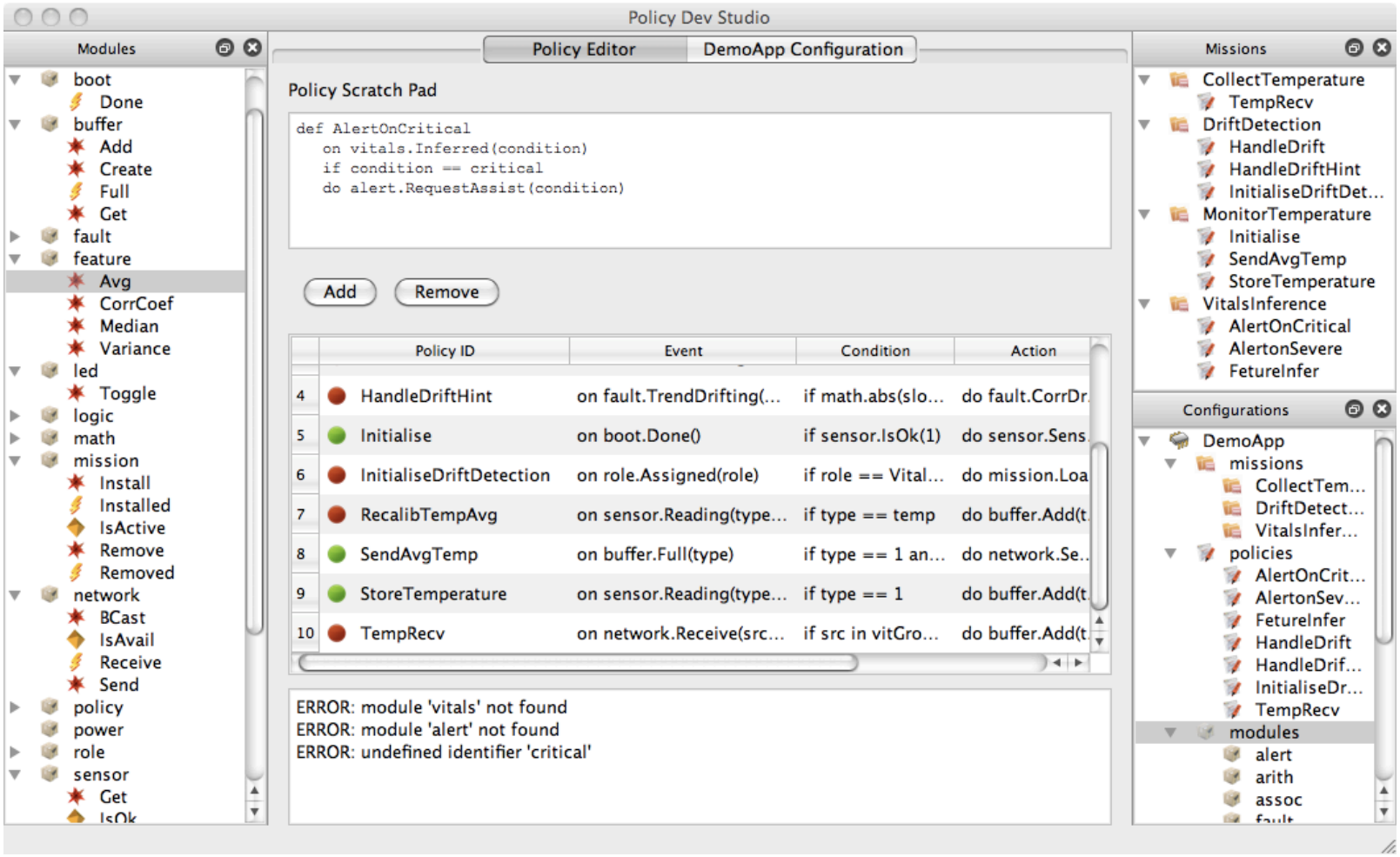
Module Interfaces

```
interface EventSourceI {
    event void evt(event_t eid, argument_t* args);
}

interface PredicateI {
    command int evaluate(argument_t* args);
}

interface ActionI {
    command int perform(
        argument_t* args);
}
```

```
def allow_nurse_policy_install +
    subject nurse
    target patient
    if power.Level() and nurse.type is staff_nurse
    action policy.Install

def ECG_request
    on gui.RequestUpdate(patient, type)
    if network.IsAvail(patient) and type is ECG
    do patient.policy.Install(ECG_update)
```

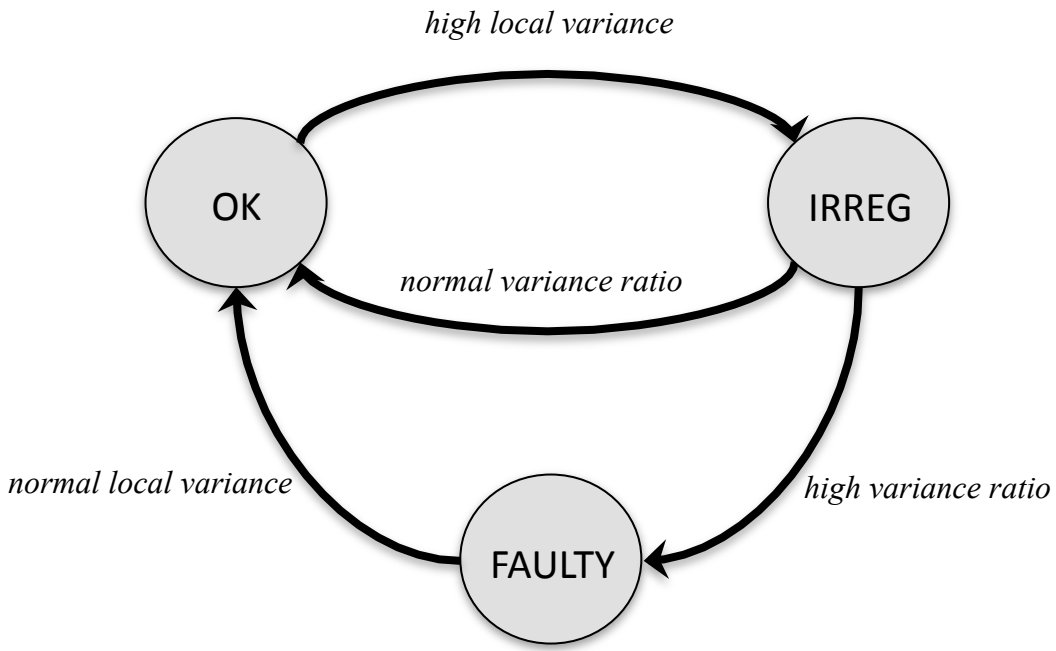
Imperial College London

Policy Authoring Tool



Imperial College London

Fault Detection



Imperial College London

Fault Detection

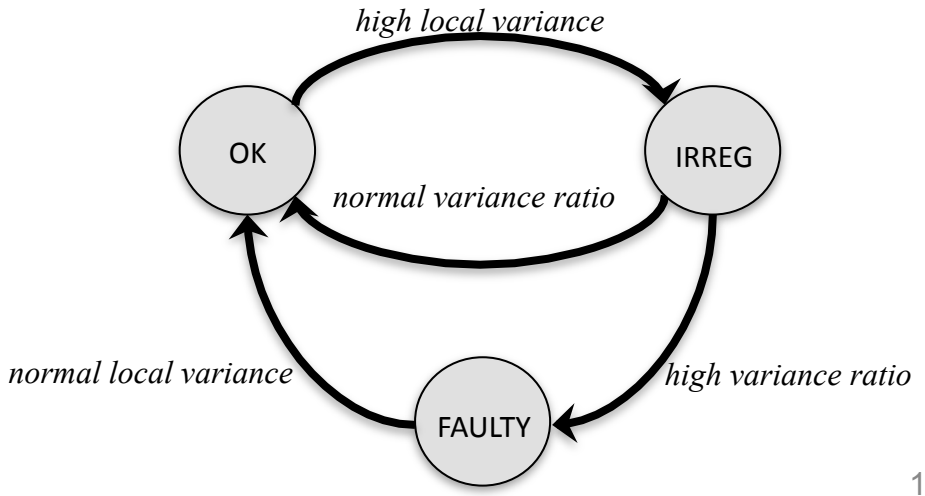
def *CalcVariance*
  on sensor.Reading(*type*, *value*)
  if *type* **in** [TMPR, HUMIDITY]
  do feature.Var (*type*, buffer.Get(*type*))

def *OkIrregTransition*
  on feature.VarRet(*type*, *variance*)
  if *variance* > VAR_TH
  do mission.Swap(Irreg, Ok),
    feature.VarRatio(*type*)

def *IrregFaultyTransition*
  on feature.VarRatioRet(*type, ratio*)
  if *ratio* > VAR_RATIO_TH
  do mission.Swap(Faulty, Irreg)

def *IrregOkTransition*
  on feature.VarRatioRet(*type, ratio*)
  if *ratio* < VAR_RATIO_TH
  do mission.Swap(Ok, Irreg)

def *FaultyOkTransition*
  on feature.VarRet(*type*, *variance*)
  if *variance* < VAR_TH
  do mission.Swap(Ok, Faulty)



Finger2 Overheads

- memory
  - ROM: 12.23KB
  - RAM: 0.72KB
    - policies min 24bytes, expected average 50bytes
  - typical motes: ROM: 48-128KB, RAM: 4-16KB

- processing
  - 69μsec overhead on average for policy execution
    - does not account for application code execution (i.e. predicates and action evaluation)

Future Work

- specialize in self-healing strategies / constructs
  - Bayesian Classifiers for fault detection
  - translate FSM to policies

- integration with ponder2 and Self-Managed Cell
  - building self-healing services on the SMC architecture
  - define interactions between collaborating SMCs
    - low level SMC (cluster of sensor nodes)
    - high level SMC (PDA/smartphone based)

Imperial College London

Questions