OSAS
An Open Service Architecture for Sensors

Johan Lukkien

Richard Verhoeven

Remi Bosman

Framework

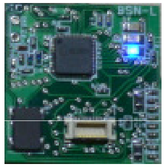
- A framework consists (possibly) of
  - a 'static' part
    - programming model, data model
      - libraries
    - life cycle model
    - methods or tooling for development
  - a 'dynamic' part
    - a run-time system, or platform
      - entirely separate entity or a library
    - a set of services
      - provided by the platform
      - e.g. binding, installation
    - a process model
- A framework has views, e.g.,
  - *logical view* for the framework user (application developer)
    - programming model
    - visible services
  - *development view* for the framework developer, programmer
    - the logical organization of the framework tools and platform
    - the services structure
    - the code
  - *process view* for developer and framework installer
    - the processes in the framework, the connection to the OS, the protocols
  - *deployment view*

Contents

- Introduction to the design problem
  - system operation sketch, goal
  - stakeholders
  - technical environment
  - operational environment
  - constraints
- Analysis
- Design decisions
- Reflection on concepts
- Conclusion

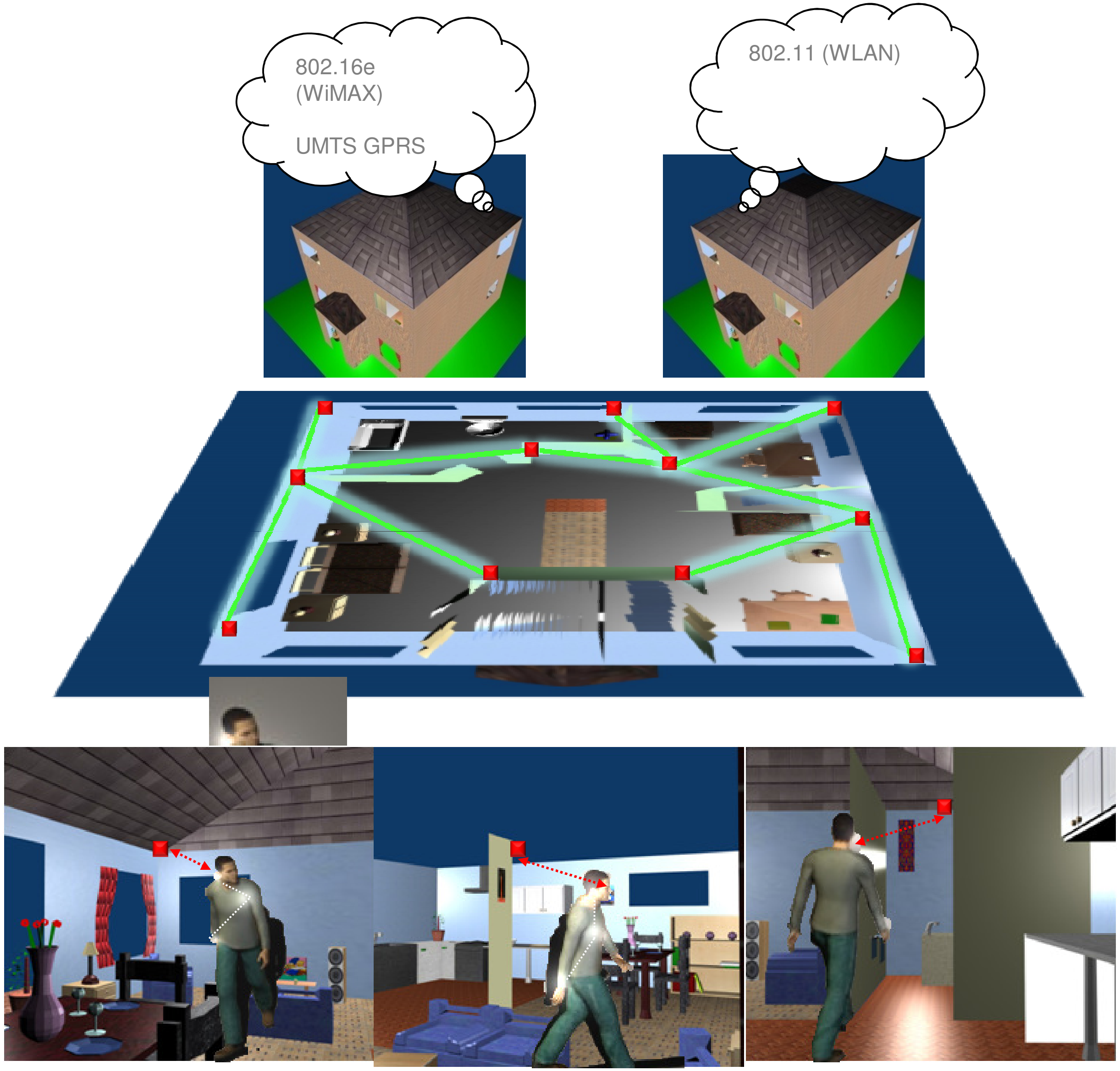
System outline

- The system consists of
  - clusters of wireless sensors
    - sensing, (actuating), computing, communicating
    - cheap, small, mobile, unreliable (communication), low on resources, many
    - mobile (wearing) and ambient deployment
  - infra structure, bridge-ing and backend-machines

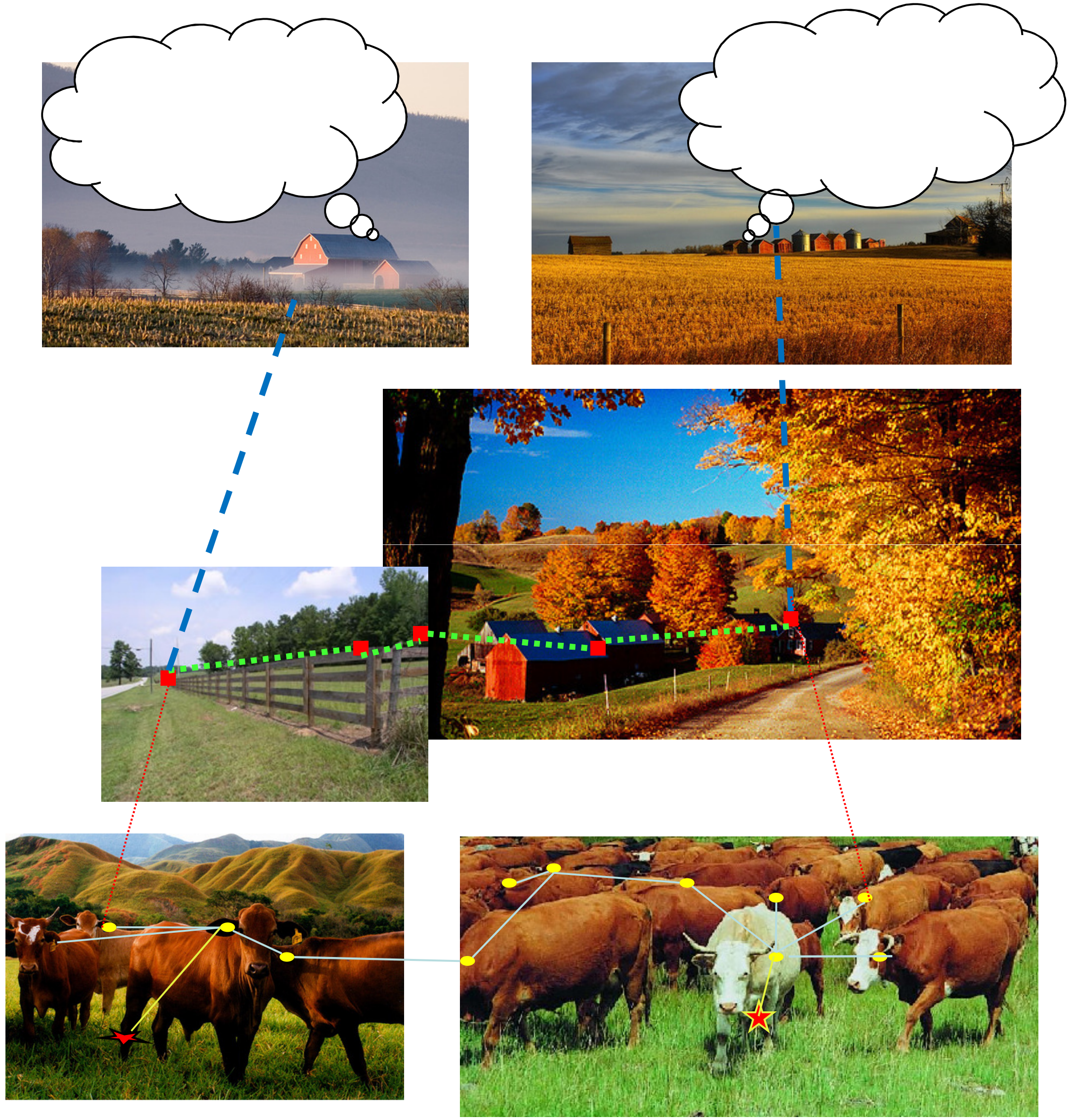


- The goal:
  - develop a programming framework for sensor networks
    - specify sensor behavior, and computations
    - adjust sensor behavior
    - integrate in infrastructure
    - ‘convenient’ deployment (impossible to physically contact each sensor)

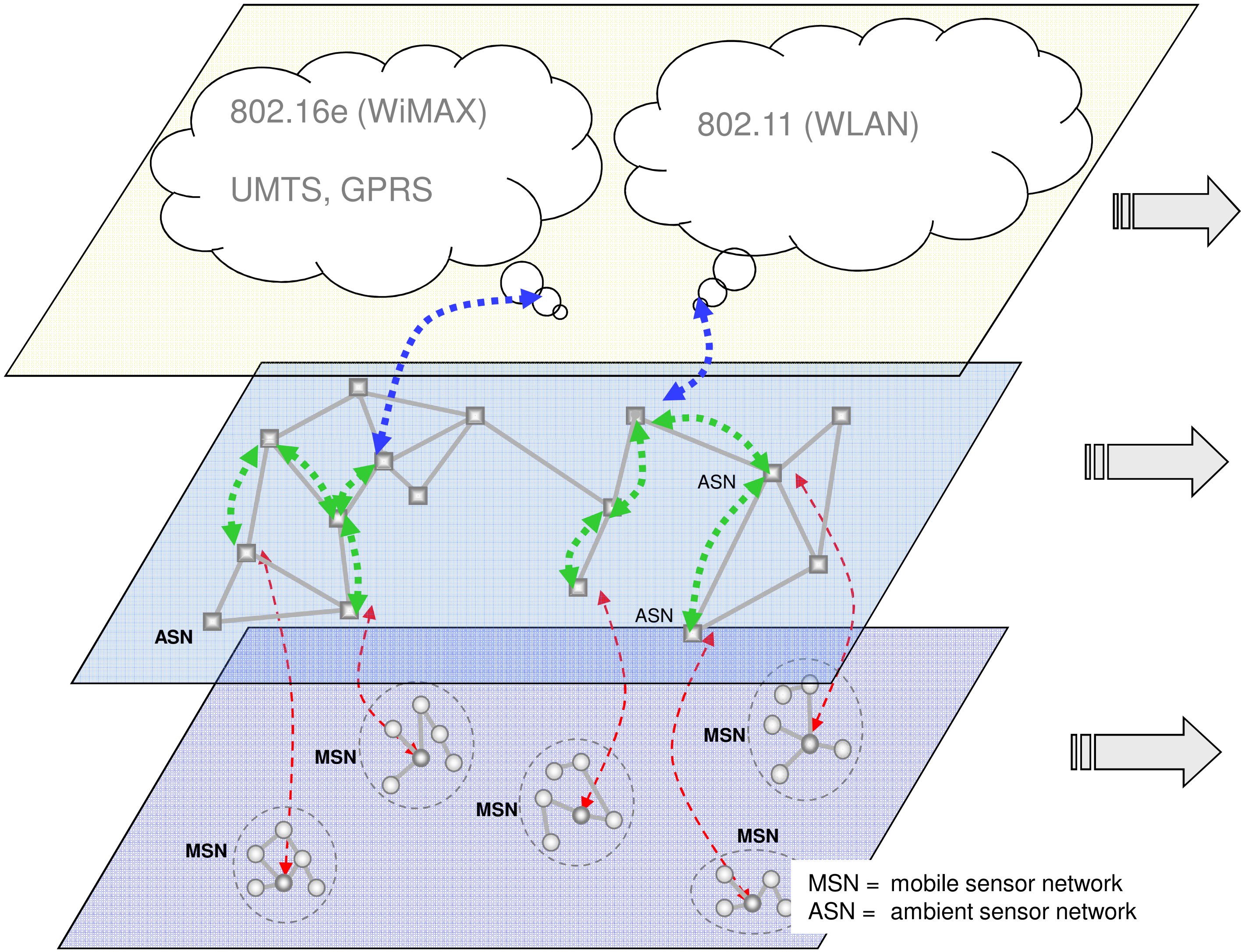
Elderly care: person moving between different locations



Herd control: monitoring animals in the field



General architecture
Physical organization



IP-based (cellular) network with wide-area coverage

WSN infrastructure: static ambient nodes using the same communication technology as lowest layer

Moving clusters of nodes with rich sensing capability but with limited resources

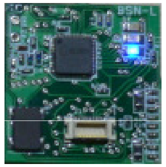
e.g. single person moving around

Taxonomy

Middle layer / Bottom layer	Multi hop (ambient infra structure)	Single hop (no hop) (access points)
Multi hop	Most general case: moving clusters through ambient infra structure	Moving clusters connecting to access points
Single hop	Moving nodes connecting to ambient infra structure	Moving nodes connecting to access points

System outline

- The system consists of
  - clusters of wireless sensors
    - sensing, (actuating), computing, communicating
    - cheap, small, mobile, unreliable (communication), low on resources, many
    - mobile (wearing) and ambient deployment
  - infra structure, bridge-ing and backend-machines

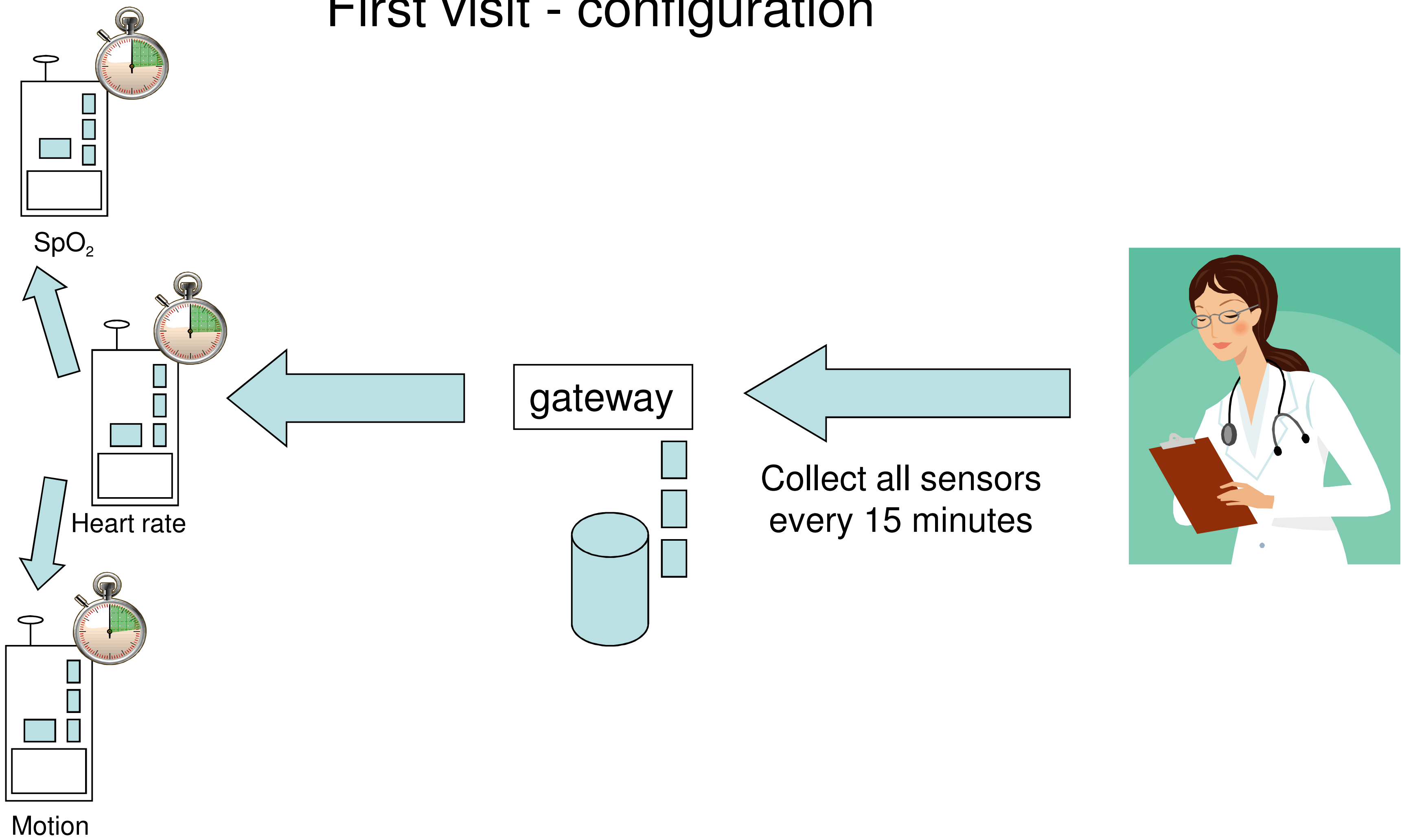


- The goal:
  - develop a programming framework for sensor networks
    - specify sensor behavior, and computations
    - adjust sensor behavior
    - integrate in infrastructure
    - ‘convenient’ deployment (impossible to physically contact each sensor)

Stakeholders

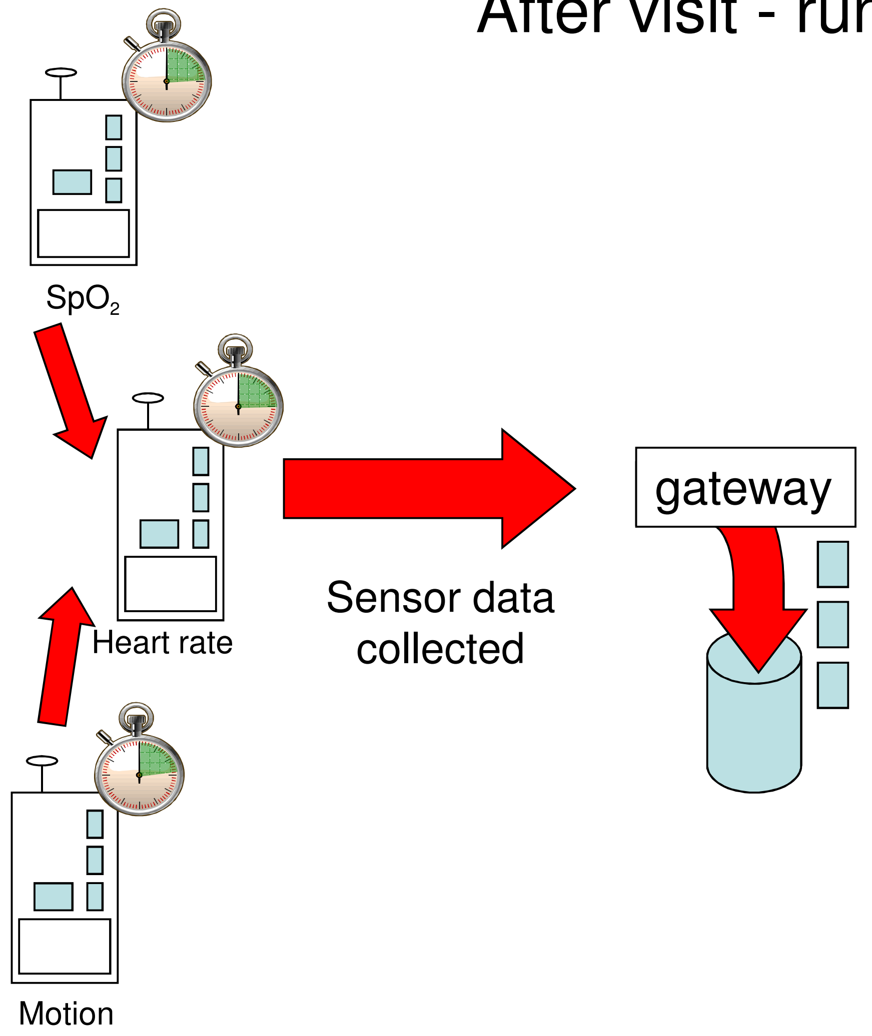
- *Users* (farmers, doctors)
  - buying and deploying sensors
  - configuring sensor systems
  - **note**: the person that *wears* the sensors is not considered as stakeholder here
- *Application builders*
  - programmers, programming sensor systems
  - putting systems together
- *Integrators*
  - integration of new hardware
  - developing and integrating system software

Scenario of running system
First visit - configuration



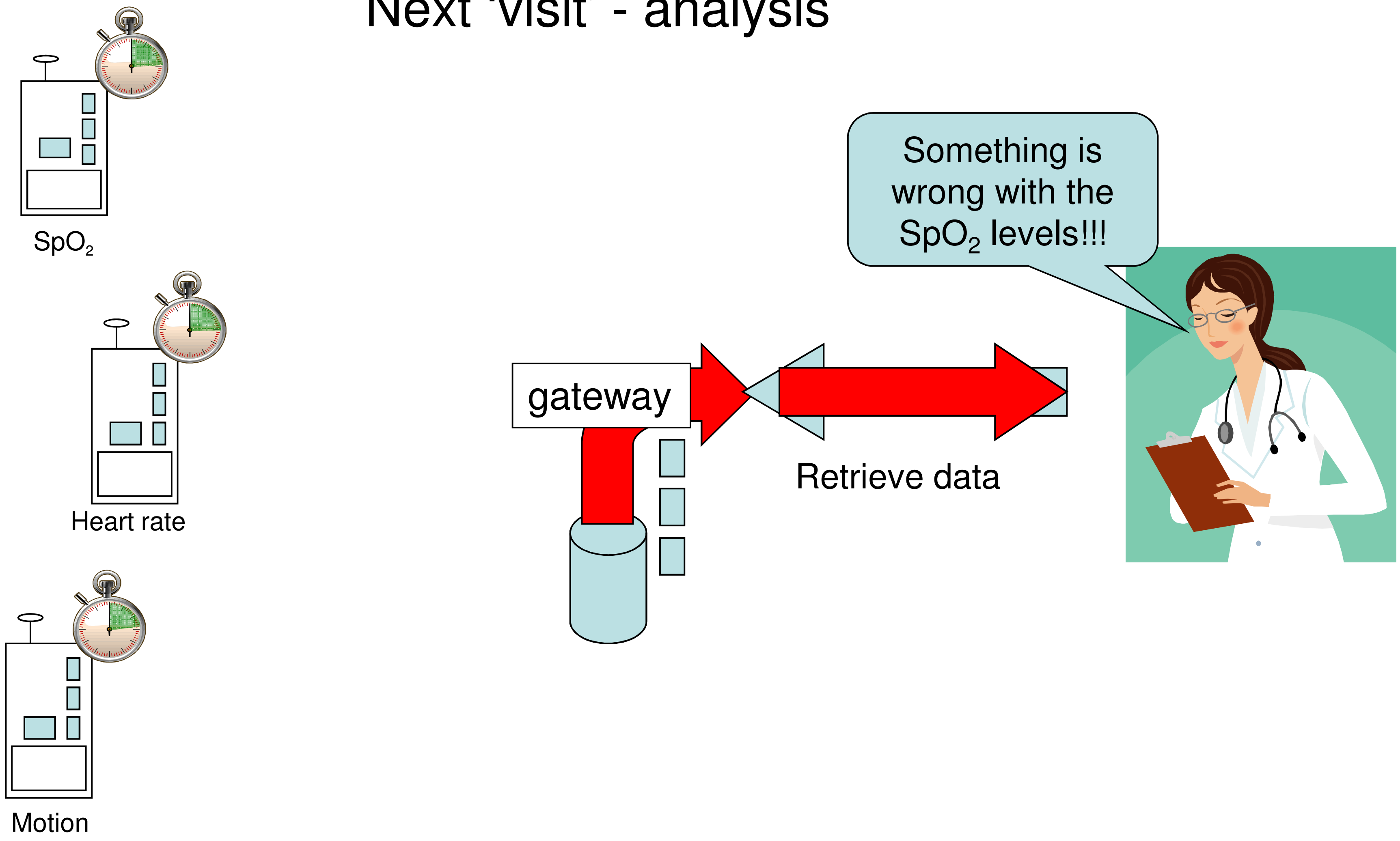
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Running system
After visit - running



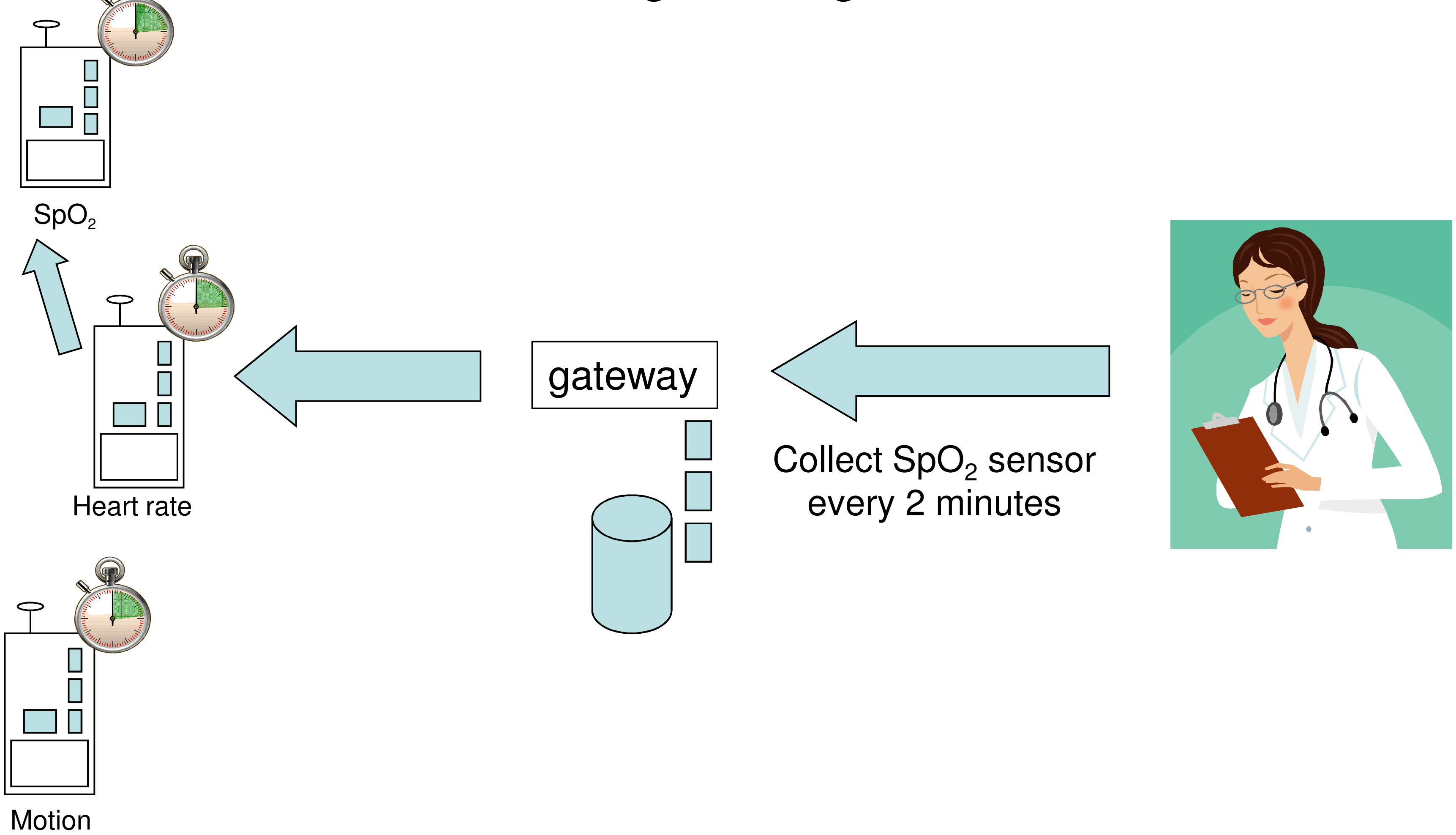
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Running system
Next ‘visit’ - analysis



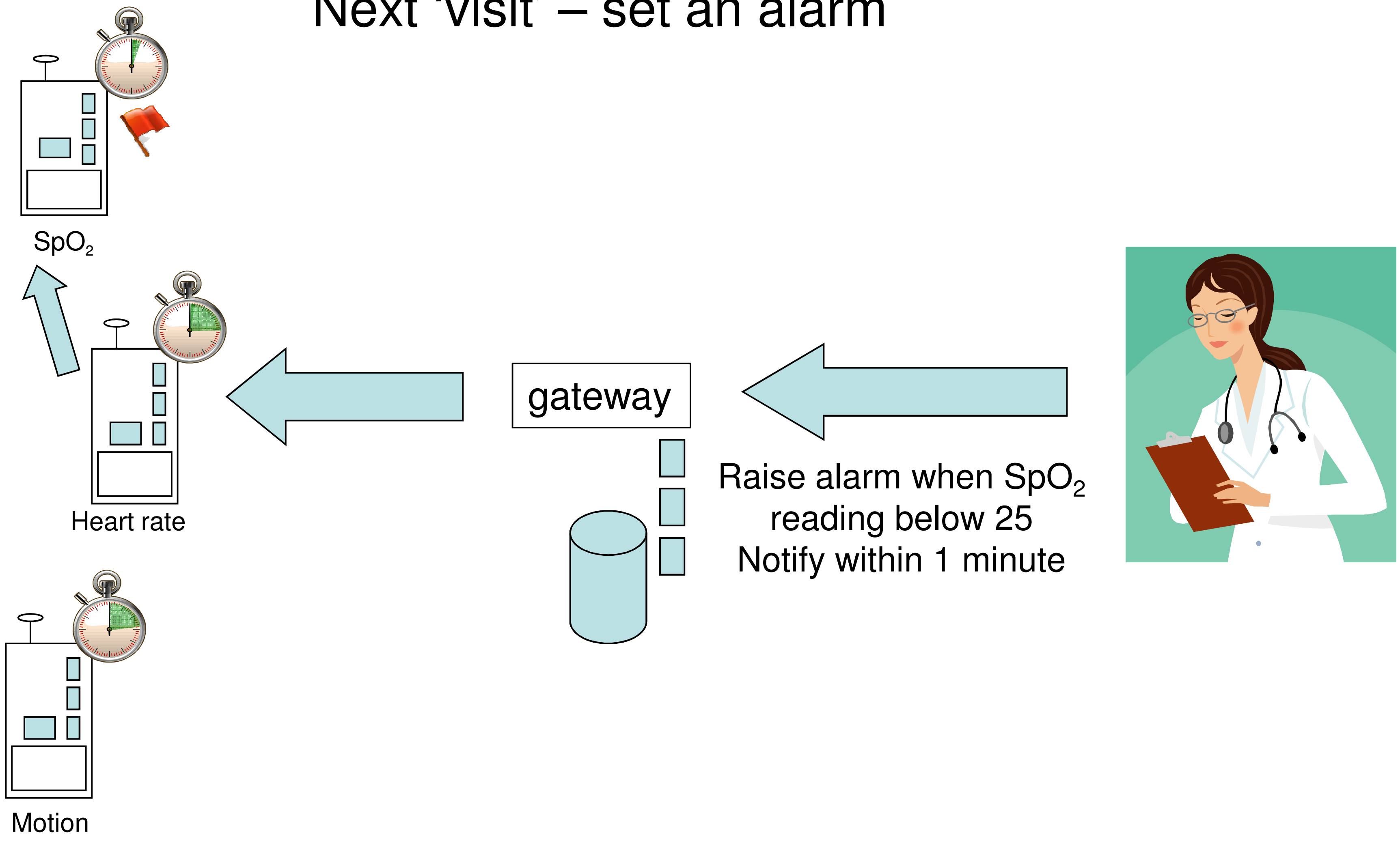
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Running system
Next ‘visit’ – change configuration



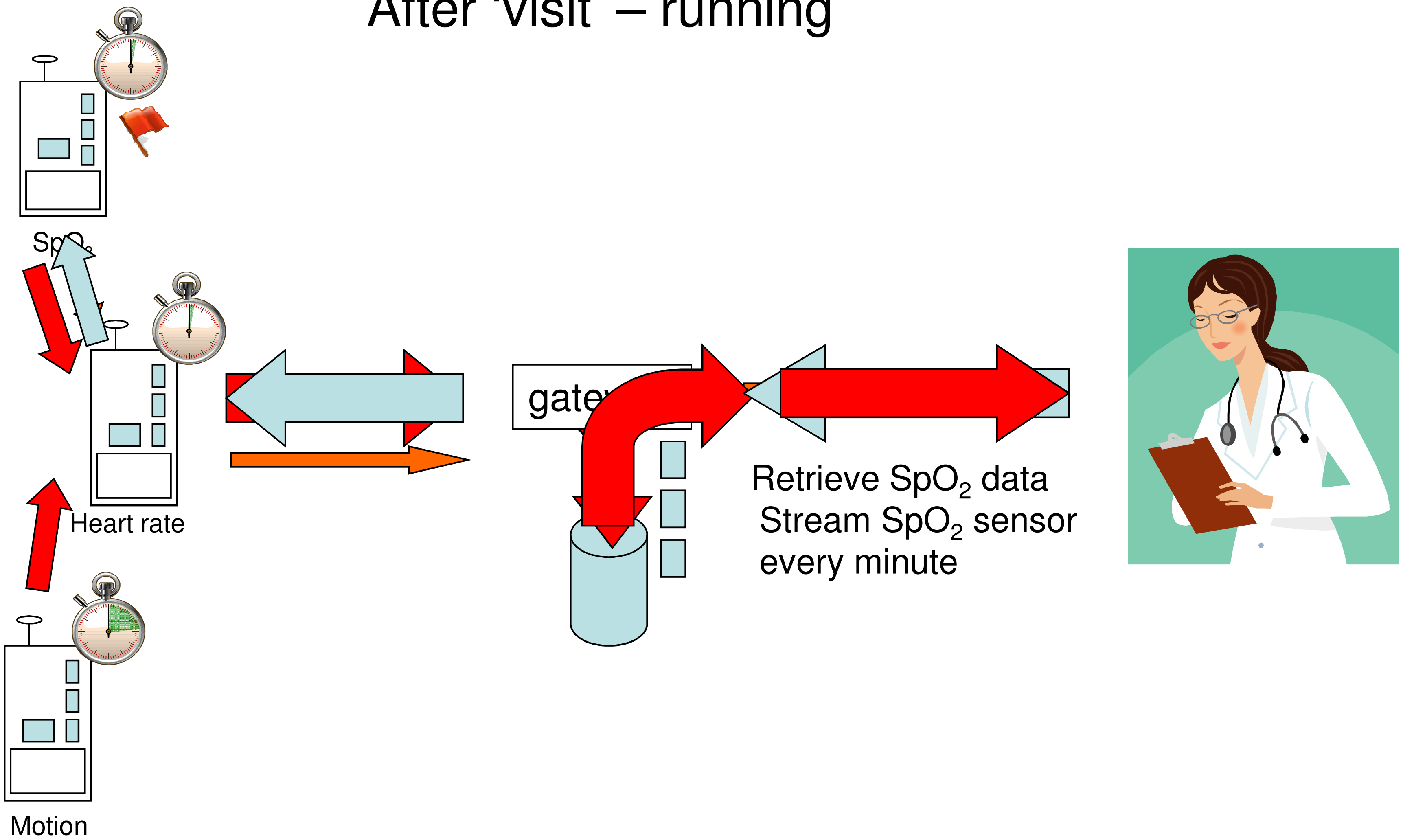
Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Running system
Next 'visit' – set an alarm



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Running system
After 'visit' – running



Building blocks & constraints

- Physically: mentioned devices
  - several kilobytes of flash, 2-10kB of RAM, slow clock
  - small batteries, small range radio
  - error-prone wireless communication

- Software (explained later):
  - on nodes:
    - Operating System, providing *system calls*
      - basic OS functionality, but little protection
      - sensor / actuator access
    - Network stack
    - to-be-developed components, using a language of choice (typically C)

  - in the infra structure:
    - standard platform (e.g. Windows)
    - to-be-developed components, using a language of choice

Extra-functional properties

- Energy constraints
  - operational for months on small batteries
- Long time, no touch
- Programmer productivity
- Portability, limit platform dependence
- (Security, privacy)

Technical environment

- For our design we used:
  - Mantis OS, with a simple link layer protocol in the sensors
  - C as programming language
  - both Linux and Windows platforms in the back-end systems
  - Python, on Linux and Windows
  - A Compiler-Compiler

- **Note**:
  - these are not always given constraints but choices resulting from research
  - it is debatable to what extent such choices influence the architecture. However:

    The concepts that someone works with determine

    the way of thinking about a problem and the choice of solutions

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Use cases

- Use case of a doctor (see previously)
- Programmer:
  - programming model
  - workflow of writing, deploying and debugging programs for entire networks
    - specifying behavior and computations of entire system
  - special issues:
    - very limited feedback possibilities from sensors

Contents

- Introduction to the design problem
- Analysis
- Design decisions
- Reflection on concepts
- Conclusion

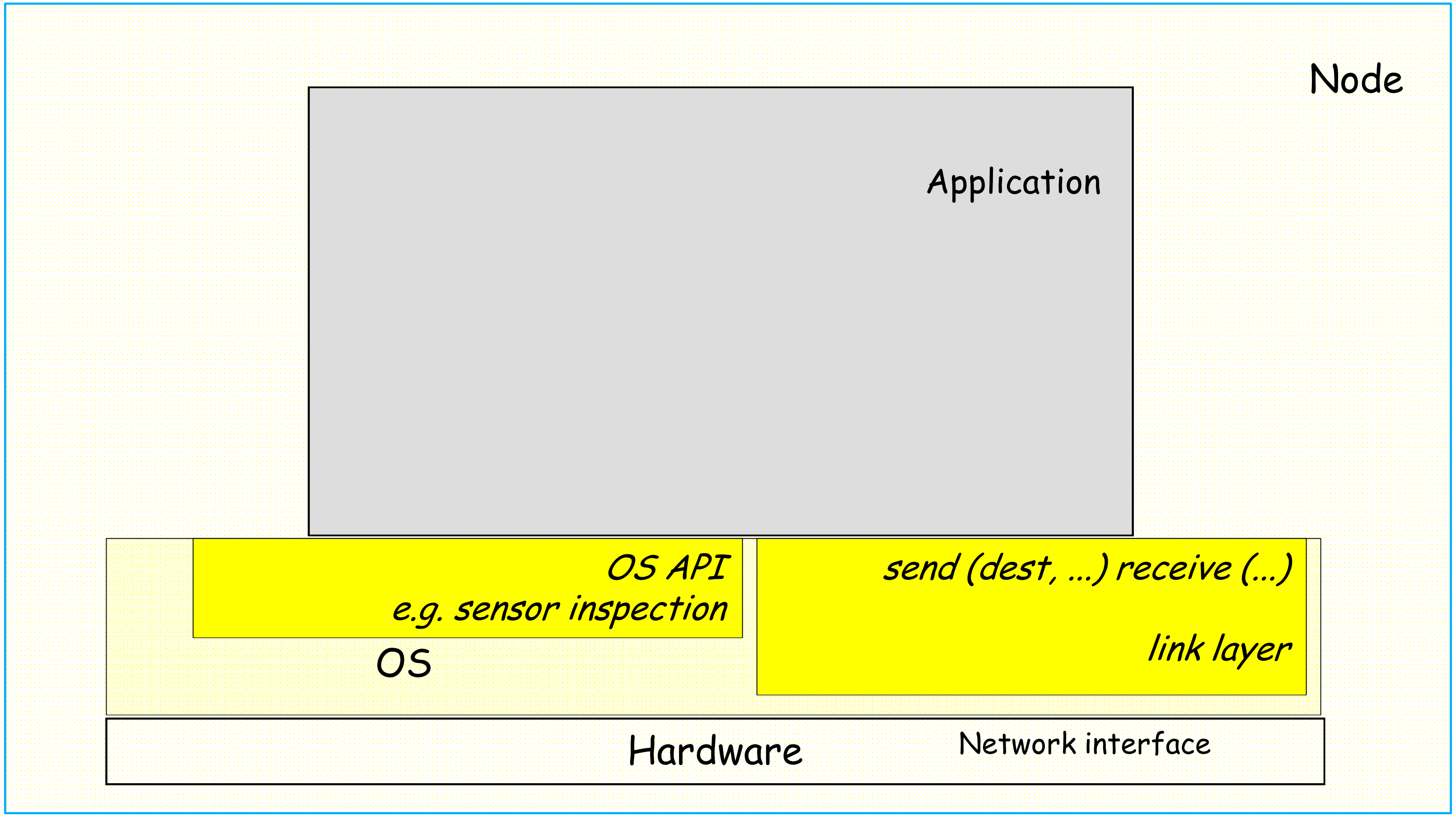
Assumptions about node software environment

- Common assumptions, for systems more powerful than e.g. PDAs
  - POSIX OS, supporting regular OS services
    - process, thread management
    - file, memory and other resource management
    - i/o
  - TCP/IP protocol stack

- Sensor node capacities are so small that
  - no superfluous functionality should be supported
  - in fact, the running of an application should be fully optimized
  - typically this is done by *cross-layer optimization*
    - breaking the conceptual layering, by using information at different layers to obtain global optimization
      - e.g. application-dependent use of the wireless link (app. dependent MAC)
      - e.g. integrating OS functions, applications and communication

- Hence,
  - embedded OS's, if any, with very different programming models
  - no TCP/IP – just a simple link layer communication with neighbors

Model in development view



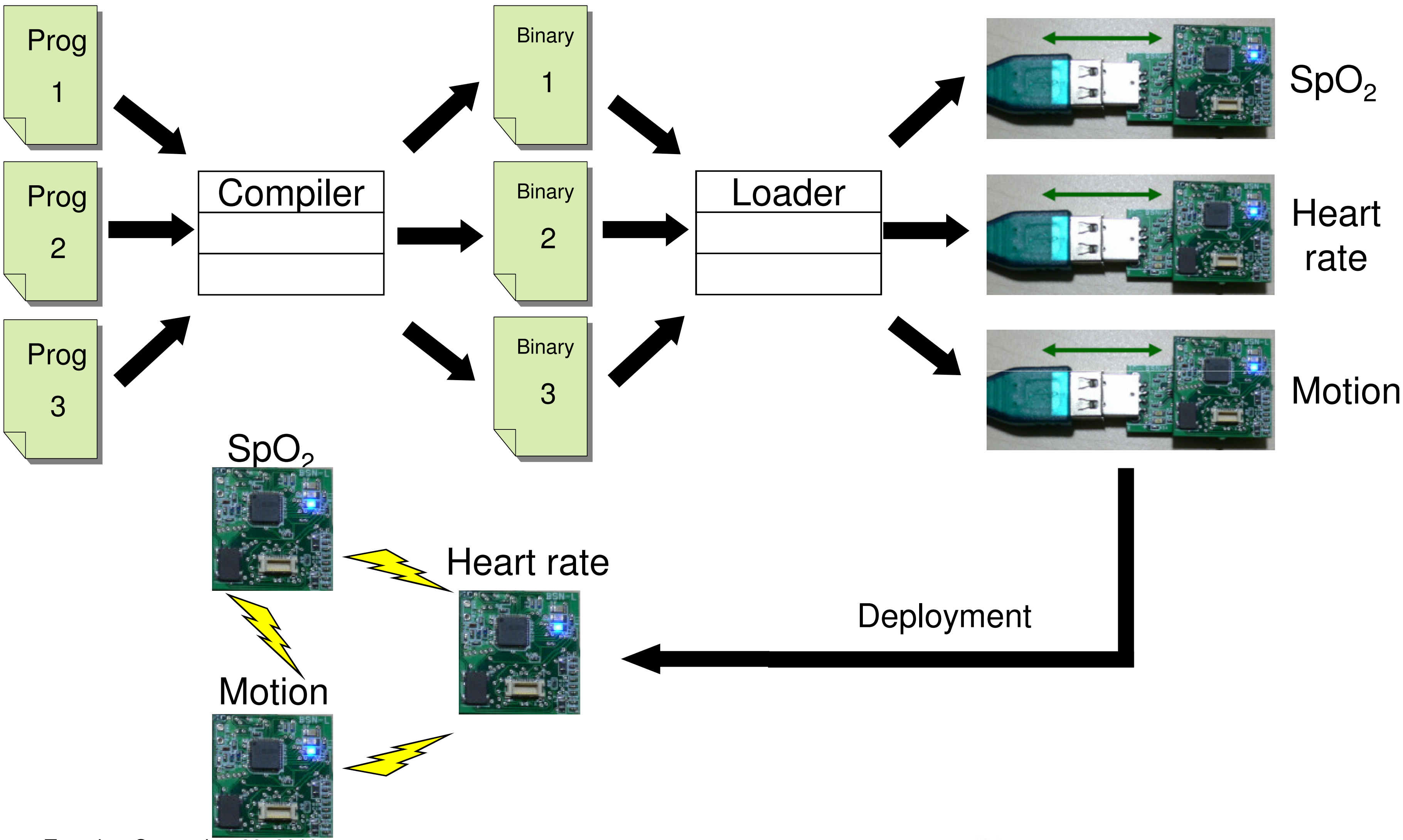
Program life cycle

- Adaptations at runtime range from reconfiguration to reprogramming
  - hence, program deployment upon system setup as well as during operation

- 'Traditional' means of deploying distributed systems:
  - compile, store and run a program for each machine
    - typically by having physical access
    - example: MPI and PVM programs started as a distributed virtual machine
  - client-server:
    - server 'always on'
    - client code started manually, e.g. after downloading
  - configuration (parameters): encoded by the programmer, as part of the program

- We investigate:
  - how to deploy program code
  - what to deploy (partial or full binary, intermediate code)
  - when to add configuration

Traditional development applied to sensors



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

‘Performance’ of deployment procedures

- Deployment procedures have independent choices:
  1. send code through a physical connection (A) or send code over the air (B)
  2. install specific code for each node (A) or send the same code to each node (B)

- Qualities aspects of these choices: performance & reliability
  – performance: communication volume (= energy) and time spent
    - 1A:
      – takes ~minutes per sensor (extract sensor from environment; attach to server; deploy (again) in environment)
      – obtaining the sensor physically may be prohibitive
      – may integrate with normal operation procedures: seeking the right moment for update
    - 1B, 2A (needs reliable communication)
      – the volume sent grows as the number of nodes
      – note that not all nodes are involved; however, nodes close to the source will do more work
      – may integrate with normal operation procedure
    - 1B, 2B (reliable multicast)
      – the code is sent just once (experiments: next slide)

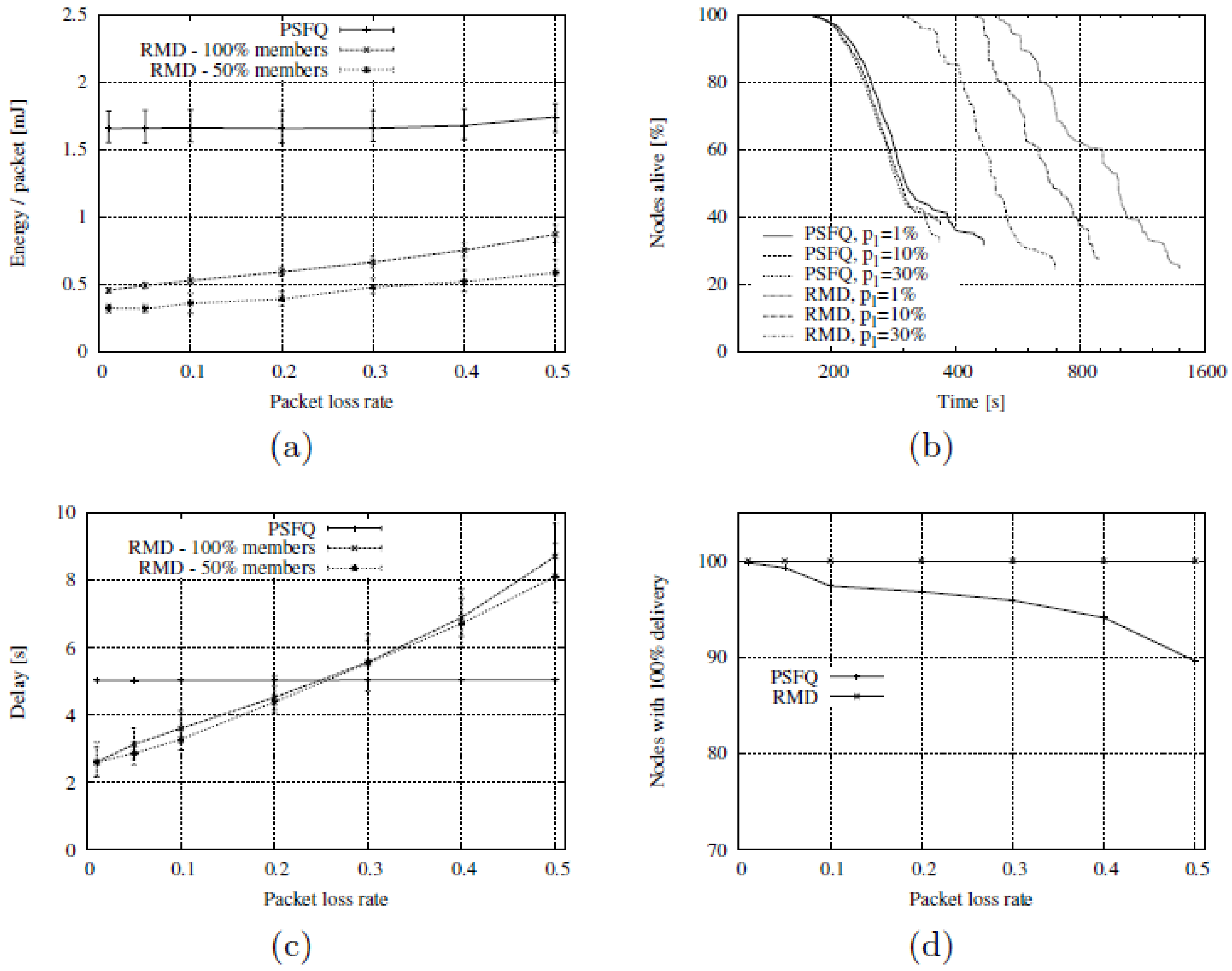
Simulation timing of reliable multicast

100 nodes, randomly in a square (200mx200m) configuration, average delay per packet of code (128b) to be disseminated to a percentage of nodes

Injection rate is 1/s

A full binary is usually around 25kB

The time is determined by the diameter of a spanning tree built for this multicasting (here roughly 7.7). This could become much larger in certain cases.

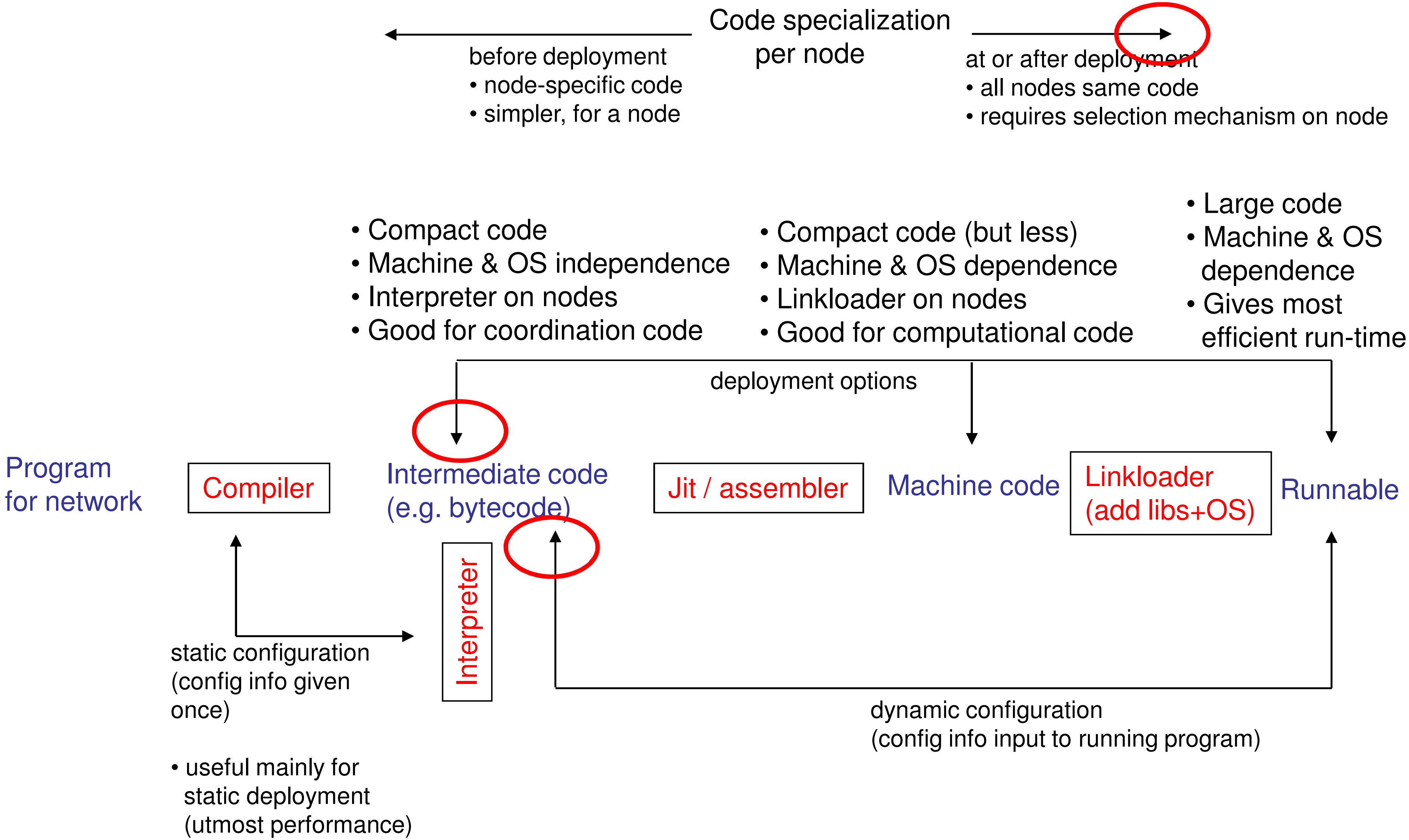


From: *Collaborative Wireless Sensor Networks in Industrial and Business Processes,* Mihai Marin-Perianu, PhD thesis, Twente University, November 2008.

Scalability of deployment procedures

- Usage parameters:
  - number of nodes
  - code size
- Metrics:
  - delay, energy
- Scalability criterion:
  - better than linear dependence
  - small constants (e.g. per node handling penalty)
- In order to keep delay and energy small, our architecture
  - must support loading over the air
  - have a single, small code for all nodes
    - or more precisely: a small number of different code classes
- Note that network diameter is not under our control

Analysis & design choices



Contents

- Introduction to the design problem
- Analysis
- Design decisions
  - decisions
  - results
- Reflection on concepts
- Conclusion

Applied styles

- Architecture style
  - publish & subscribe
  - service oriented
  - virtual machine

- Interaction style
  - event-based
  - asynchronous RPC
  - active messages

Event-based interaction style

- *Event*: change in state of an entity
  - typically related to observations, or time evolution
  - entity: e.g. software component, object, machine
- *Event notification*:
  - asynchronous message, without connection or acknowledge
  - asynchronous invocation, without result (cf. asynchronous RPC)
  - connector: *event bus, event dispatcher*
- *Event-based interaction style:*
  - interaction between entities is through event notifications only
- Motivation:
  - decoupling in *synchronization,* delay *binding*
- Extended in an architectural style by adding architectural elements
  - eventing subsystem, “brokers”

Design decision: services

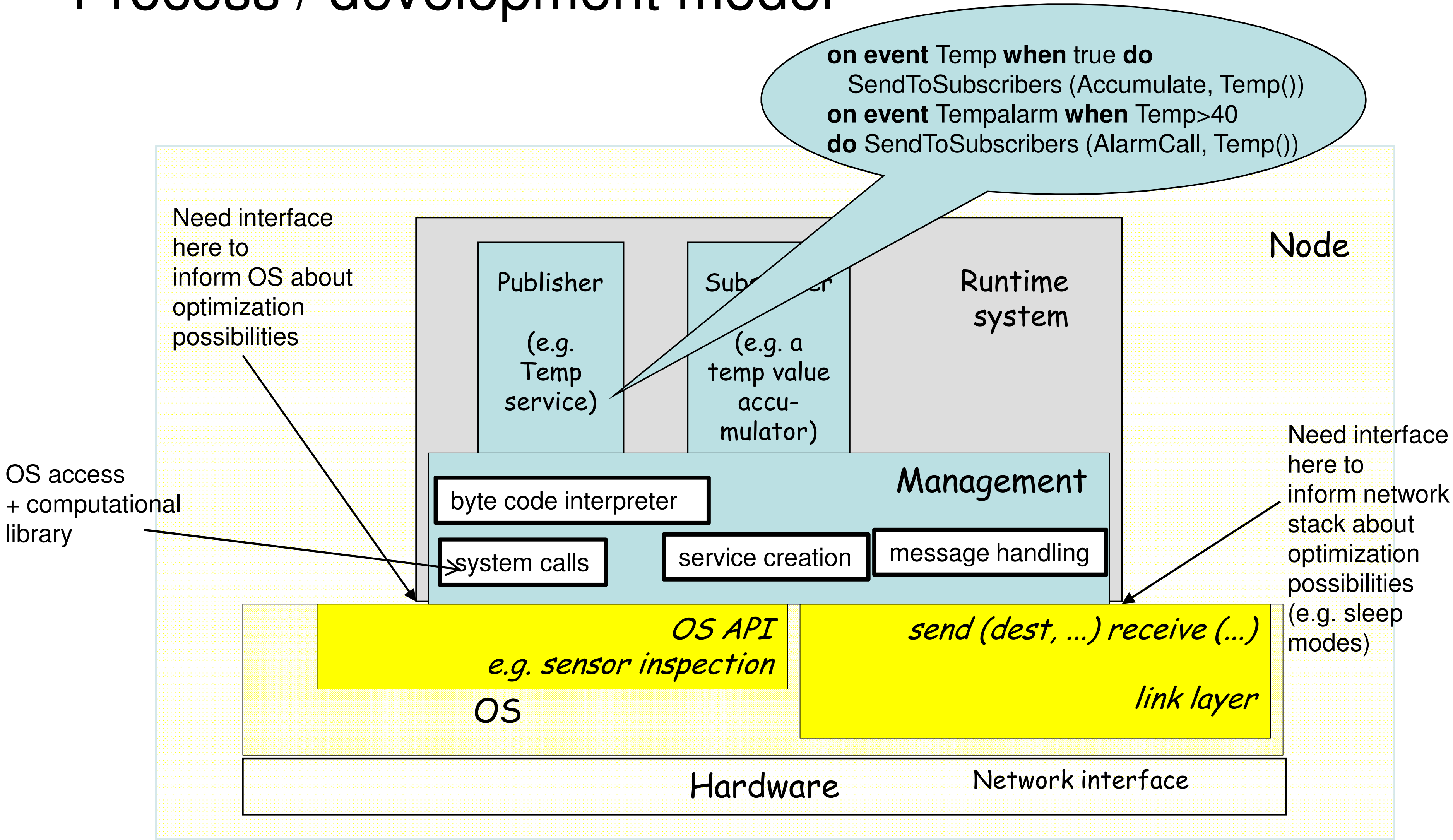
- (after further literature search): Service Oriented Architecture

- *Services* are delivered by nodes through exposed interfaces on the network, comprising
  - events: sampled conditions
    - with a given frequency the condition is checked; when *true,* the event 'occurs' (an event body is executed, resulting in notifications)
  - actions: procedure call (without result)

- *Subscriptions*: fill in event *parameters*; connect events and actions
  - event generates a notification: an asynchronous remote procedure call
  - *publisher*: event generator service
  - *subscriber*: service of which an action is (possibly) triggered
  - the sampling rate, and other parameterization *is determined by the subscriber*

- A *system service* on each node comprises
  - installing/managing new services
  - establishing subscriptions

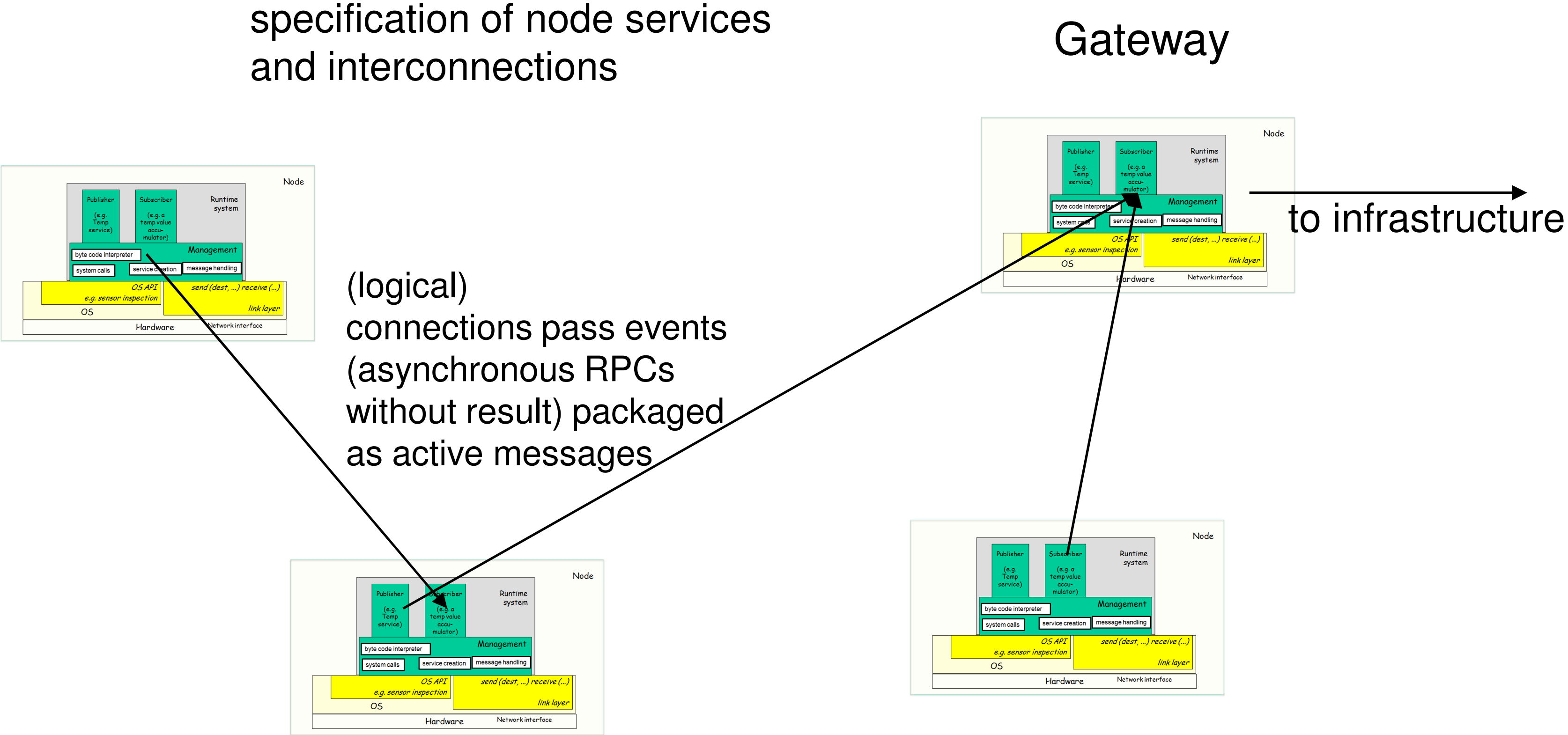
Process / development model



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

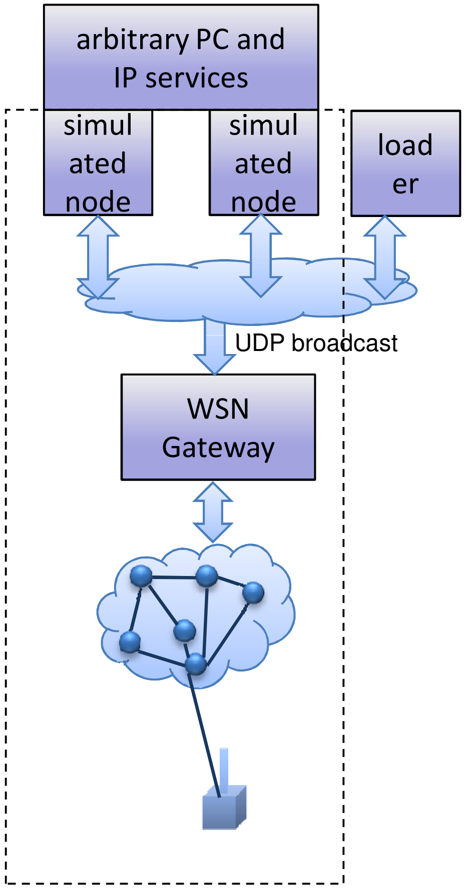
Application builders view on WSN (deployment view)

Program: *event based;*
specification of node services
and interconnections



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

A (logical) model in the development view



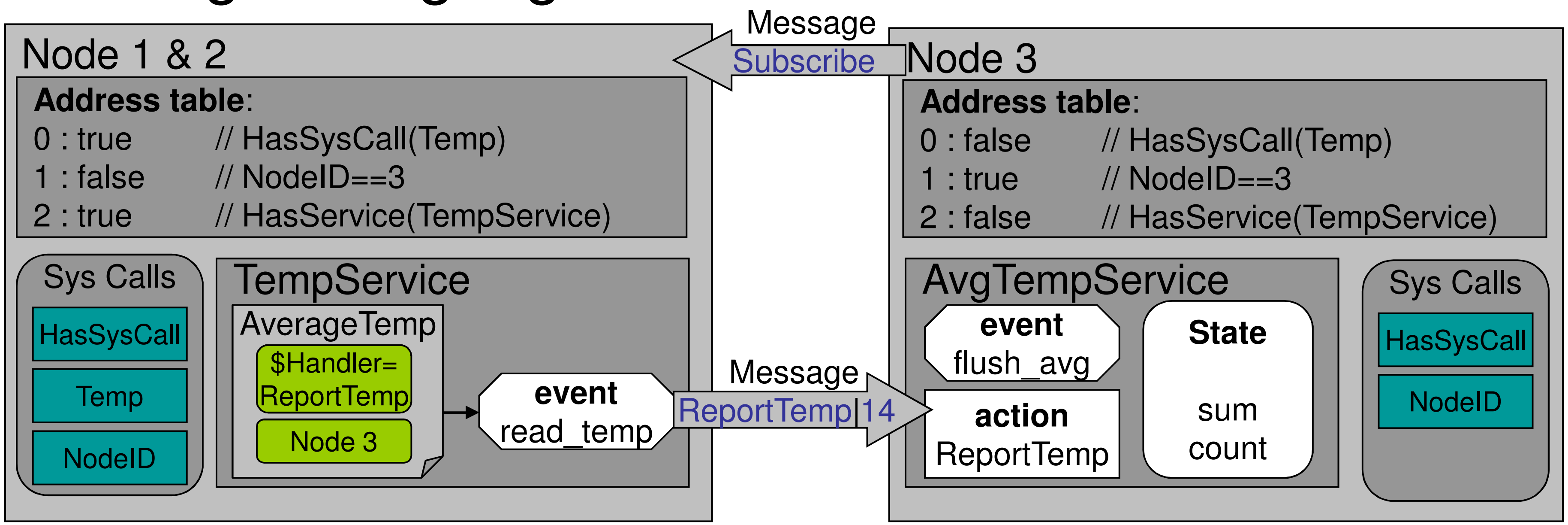
- The mobile node clusters at the bottom
- Gateway bridges node domain with UDP
  - can connect multiple WSNs in this way
- Simulated nodes interpret the exact same messages
- Simulated nodes
  - have access to arbitrary PC and Internet services
  - may provide services to access the WSN
- Broadcast traffic injected in UDP appears in the node domain as well
  - in this way a loader can upload code
- Rather flat: two layers

Design decision: addressing

- A message address is a boolean function that evaluates to *true* on a destination
  - parameters to this function may be provided
    - within the message
    - within the node
- Special cases are dealt with in the system service:
  - a (regular) destination address
  - a broadcast address
- Example:
  - 'node contains a temperature sensor'
  - 'local temperature is larger than 25'
    - 25 as parameter in message
- We call this *content based addressing*

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Design: Language & semantics




```
service TempService($Handler)
   on event read_temperature when True do
      SendToSubscribers($Handler, Temp())
```

```
service AvgTempService($Handler)
   define
      sum := 0
      count := 0
   on event flush_avg when 2 <= count do
        SendToSubscribers($Handler, sum / count);
        count := 0; sum := 0
   action ReportTemp(temp) do
        sum := sum + temp;
        count := count + 1
```

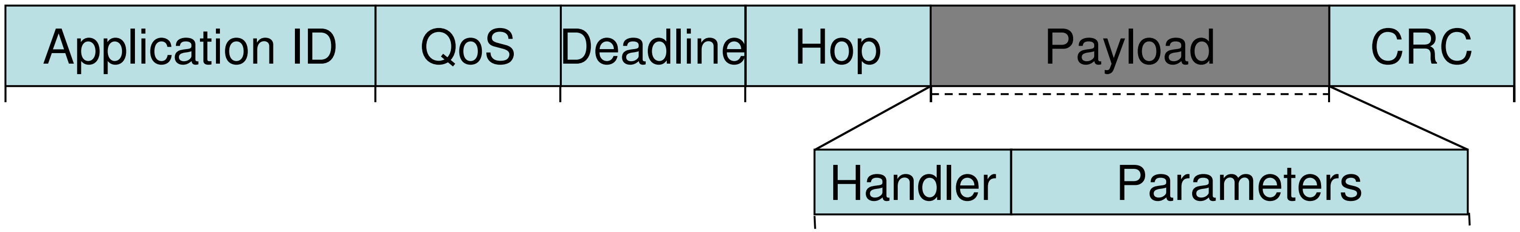
```
subscription AverageTemp
to TempService($Handler=ReportTemp)
with (period=30s, deadline=1m,
      send="High", exec="Normal")
```

```
for [Network|*|HasSysCall(Temp)]
    install TempService
```

```
for [Network|*|NodeID()==3]
    install AvgTempService
    install AverageTemp on
      [Network|*|HasService(TempService)]
```

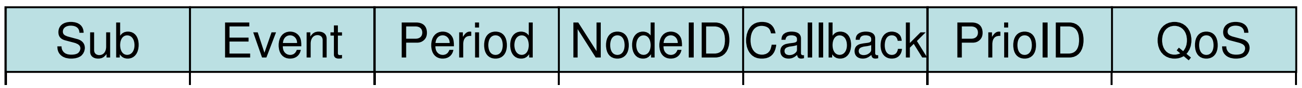
Design decision: messages

- A message fits in a single packet; the format corresponds closely to the asynchronous procedure call (aka *active messages*)
  - the payload is a *handler* and parameters to this handler
    - *handler:* a user-defined service action or system service action



- Upon receipt of a message, a node executes the handler if it understands it

- The following are examples of messages that encode a call to a specific handler
  - subscriptions
  - flooding
  - code upload

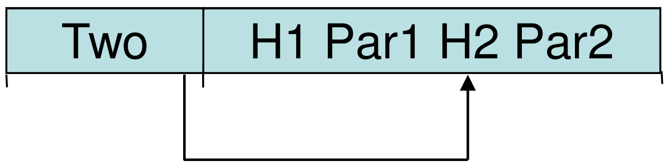


Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Composition

- Parameters to a handler can be:
  - (remainder of) message
  - (handler, parameter) pairs

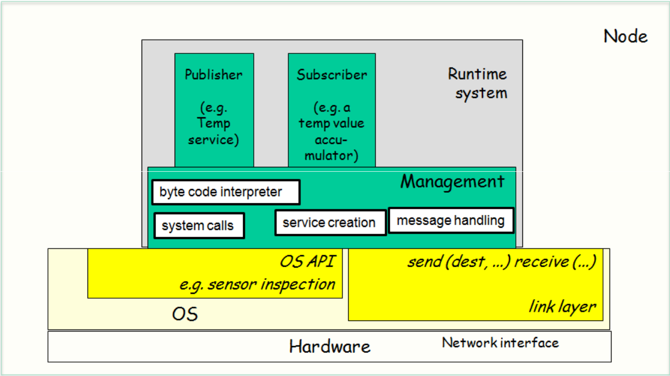
- This allows
  - conditional execution
  - forwarding
  - having more than one handler
  - adapting a message while handling and forwarding



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

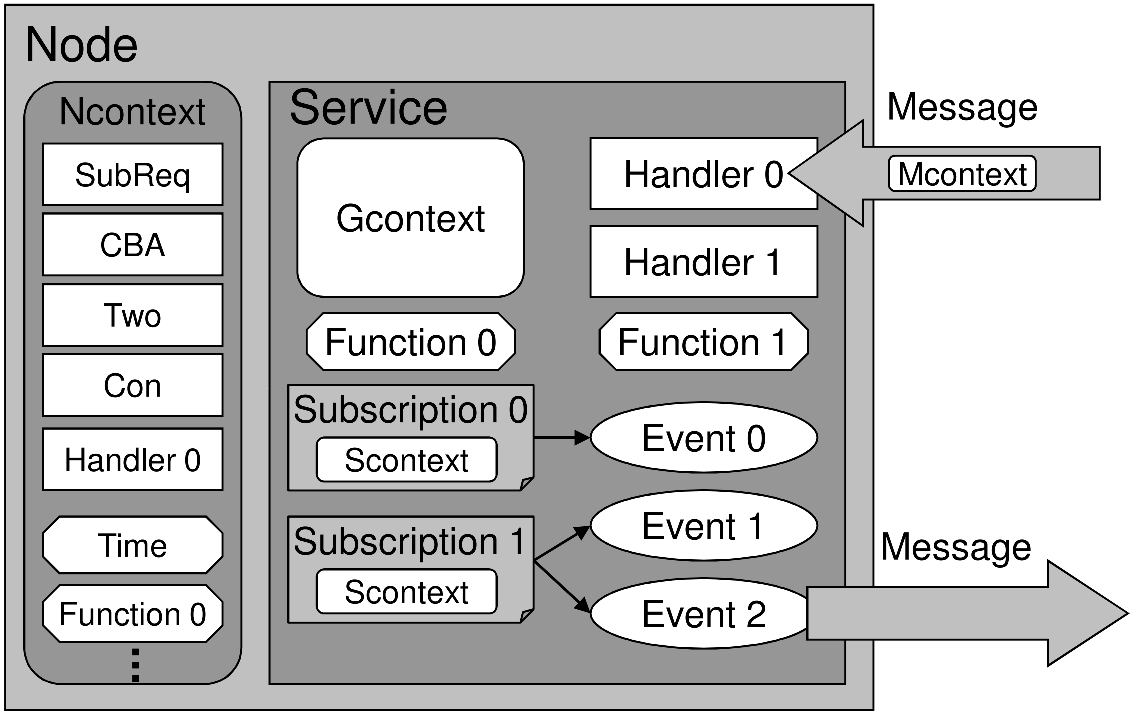
Design decision: virtual machine (byte code)

- The compiler translates a program into byte code to be executed by a virtual machine on nodes
- Fairly standard stack machine with respect to computations
  - Focus on relating *system calls* to *messaging*



- *Special*: memory organization with respect to context
  - the code of an event is executed in the context of each subscription
    - parameters are found with the subscription
  - the code of a (message) handler is executed in the context of the received message
    - that can provide parameters to this handler
  - global variables of the service are shared by handlers and actions
  - global variables in the node are shared by the services

Contexts



- Ncontext
  - Standard handlers
  - System calls
- Gcontext
  - Shared state
- Scontext
  - Timing, params
  - Subscribers
- Mcontext
  - Handler args

- The VM has instructions to refer to each context
  - e.g. PUSHG 1

Example: code load

- Configuration handler
  - Built-in handler

CON	Config instructions

- Install content-based addresses
- Install new services
  - (initialized) variables
  - events
  - actions (handlers)
- Bytes
  - VM code
  - Native code
  - ....

Config instructions

DEFA	AddrID	total	length	offset	bytes

INIT	length	bytes

SERV	serviceID	CRC

STATE	total size

DEFG	varID	size

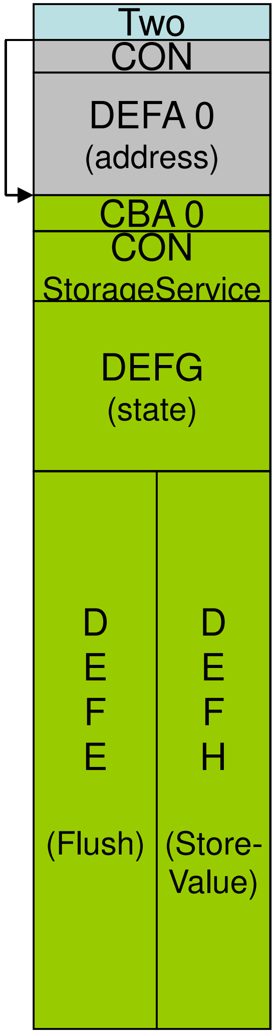
DEFGI	varID	total	length	offset	bytes

DEFE	eventID	total	length	offset	bytes

DEFH	handlerID	total	length	offset	bytes

DEFF	params	functionID	total	length	offset	bytes

Configuration example



TWO	9				# Two Handler						
CON					# Configuration Handler						
DEFA	0	3	3	0	# CBA address 0						
CALL[12]					# NodeType						
PUSHC[2]					# "StorageNode"						
EQ					# NodeType() == "StorageNode"						
CBA	0	0			# Content Based Address Handler						
CON					# Configuration Handler						
SERV	1	3			# StorageService						
STATE	14										
DEFG	0	10			# storageArray0						
DEFG	1	0			# p0						
DEFGI	2	0	10		# size						
DEFGI	3	0	12		# lastError						
DEFE	0	19	19	0	# event flush0	DEFH	18	17	17	0	# StoreValue0
PUSHV[2]					# $EventFlag	PUSHG[1]					# p0
PUSHC[2]						PUSHG[2]					# size
BAND						EQ					
RJF	14				# if cond: action	RJF	4				# then:
PUSHG[1]					# p0	PUSHC[13]					# "Overflow"
PUSHC[0]						STOREG[3]					# lastError
MORE						JUMP	17				# else:
RJF	9				# then:	PUSHA[0]					# value
PUSHV[0]					# $Handler	PUSHG[0]					# storageArray0
PUSHC[2]						PUSHG[1]					# p0
PUSHG[1]					# p0	STORES					
PUSHG[0]					# storageArray0	PUSHG[1]					# p0
SPLICE	10					PUSHC[1]					
NTFY						ADD					
PUSHC[0]						STOREG[1]					# p0
STOREG[1]					# p0						

- **Two** handler, calls two other handlers
  - Install address (CON)
  - Evaluate Content-Based Address
    - with CON as parameter
- When address holds:
  - install State
  - install Event:
    - Flush
  - install Handler:
    - StoreValue0
- Notice: all in a single (~80 byte) message

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Contents

- Introduction to the design problem
- Analysis
- Design decisions
  - decisions
  - results
- Reflection on concepts
- Conclusion

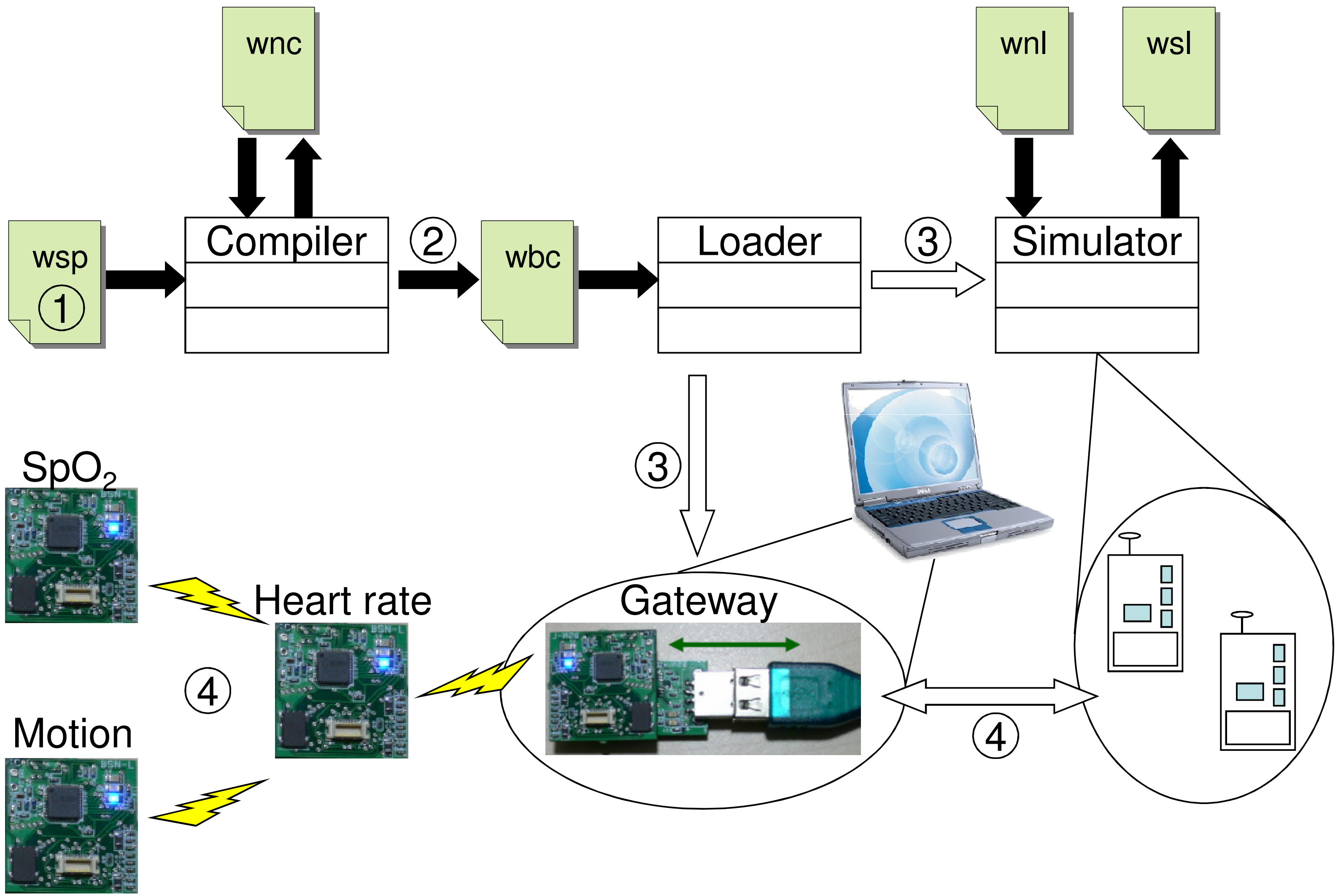
What is OSAS?

- A programming system for networked devices
  - with special emphasis on low-resource devices

- Definition of
  - *language*
    - *with the service and subscription concept*
    - *and* content-based *addressing*
      - *use a predicate to refer to a (set of) nodes*
  - *virtual machine (byte code)*
    - *has access to a set of system calls (Library and OS-provided functions)*
  - *message format*

- Four components
  - *Compiler*
  - *Loader*
  - *Runtime system*
  - *Simulator*
    - *transparent; interprets the exact same message format and can be part of the network*

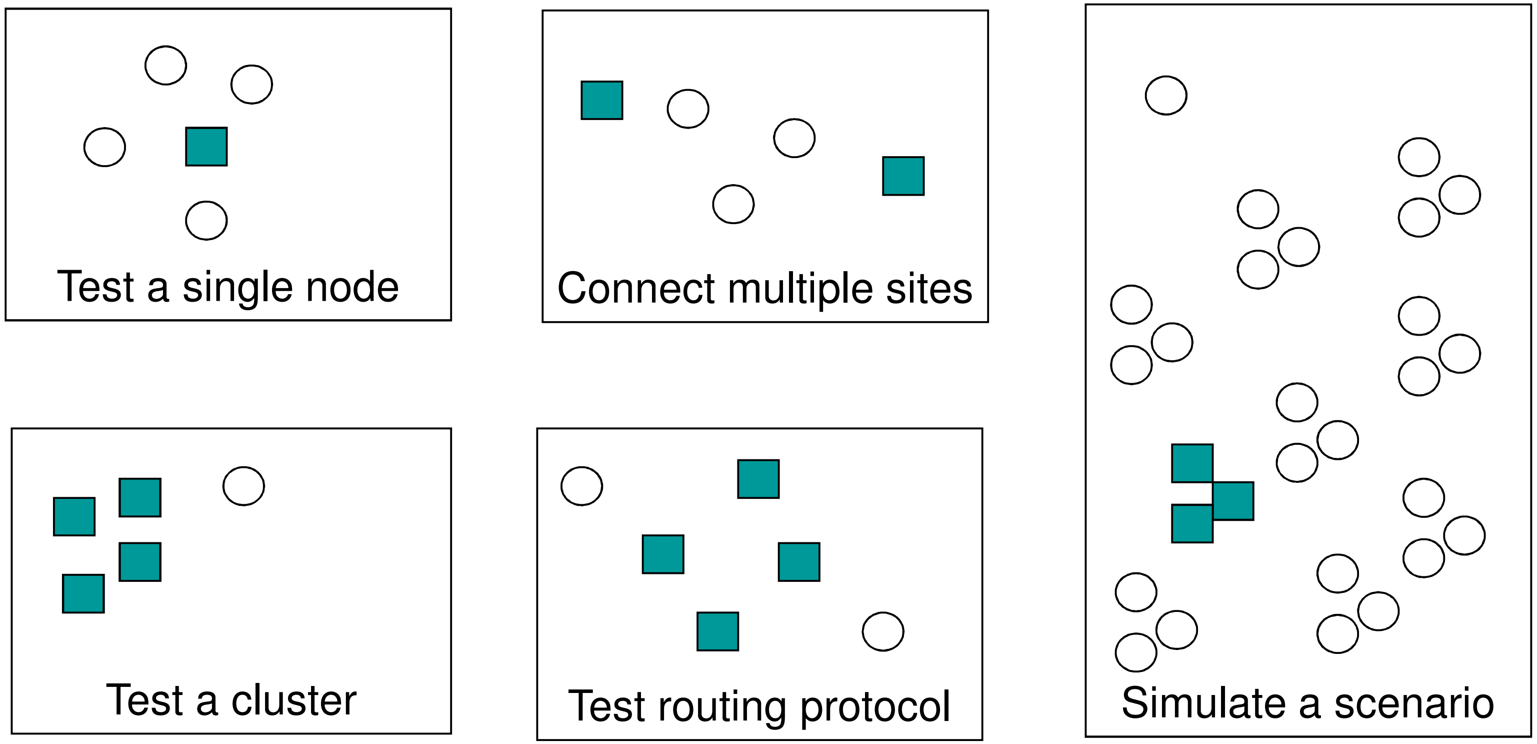
Development cycle & toolchain (logical view for programmer)



Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Virtual or real

Simulated nodes act as if they are part of a network



Contents

- Introduction to the design problem
- Analysis
- Design decisions
- Reflection on concepts
- Conclusion

Which role play the *concepts?*

- *Processes*
  - client-server relations based on *third party* binding
  - each sensor acts as separate process
- *Communication*
  - event: asynchronous remote procedure call without result
  - link (neighborhood communication) as basis
    - layer 2, or overlay
- *Naming and addressing*
  - content-based addressing
    - implemented on top of flooding
  - neighborhood addressing, can be used to build routing
- *Reliability*

Conclusion

- The goal:
  - develop a programming framework for sensor networks
    - specify sensor behavior, and computations
      - This is derived from a single program for the entire network
      - The compiler uses global knowledge for optimization
        - » no discovery or interpretation of functionality needed
      - *Computations*: delegated to pre-installed libraries (weak point)
      - *Coordination*: events, subscriptions
    - adjust sensor behavior
      - by changing subscriptions
    - integrate in infrastructure
      - simulated nodes are part of the network but support more powerful OS calls
    - 'convenient' deployment (impossible to physically contact each sensor)
      - deployment through the air using content-based addressing

Memory resources

Memory Footprint

TinyOS version for ICL nodes (values in bytes)

Static

OS image	24084 ROM,	1191 RAM
interpreter	1664 ROM,	20 RAM
system calls	1530 ROM,	146 RAM

Dynamic

bytecode evaluation stack	160
content based address	4 + length(bytecode)
service	14 + #global variables * 2
event generator	4 + length(bytecode)
action	6 + length(bytecode)
subscription	16 + #parameters * 2
message	12 + length(payload)

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Computer Science, System Architecture and Networking

Conclusion

- I did not address the development view in detail
  - modules:
    - run-time system with node OS
    - simulator, loader, compiler organization
      - perhaps sharing of data structures
  - files:
    - directory organization

- The current system has been built according to this architecture and operates satisfactorily
  - needs improvements in reducing energy

- The architecture as well as the design were driven mainly by extra-functional properties, and directed by careful analysis

- Ideas can carry over to 'regular' systems; however, their value is not obvious then
  - current trends are towards independent services that determine their mode of cooperation by extensive processing (e.g. web services with ontologies)

Some studied approaches

- Virtual machines
  - P. Levis, D. Culler. Maté: a tiny virtual machine for sensor networks, Proc. of ASPLOS-X, 2002.
  - R. Miller, G. Alonso, D. Kossmann, A Virtual Machine For Sensor Networks, Proc. of EuroSys 2007.
- Special-purpose engine
  - S.R. Madden, M.J. Franklin, et al. TinyDB: an acquisitional query processing system for sensor networks, ACM Transactions on Database Systems, Vol.30, Issue 1, March 2005.
  - H. Liu, T. Roeder, et al. Design and Implementation of a Single System Image Operating System for Ad Hoc Networks, Proc. of MobiSys, 2005.
- Macro programming
  - R. Gummadi, O. Gnawali, R. Govindan. Macro-programming Wireless Sensor Networks using Kairos, Proc. of DCOSS, 2005.
  - L. Evers, P.J.M. Havinga, et al. SensorScheme: Supply chain management automation using Wireless Sensor Networks, Proc. of ETFA, 2007.

Approaches (cnt'd)

- Active messages
  - P. Levis, D. Gay, and D. Culler, Active Sensor Networks. Proc. of NSDI, 2005.
  - T. von Eicken, D. Culler, et al. Active messages: a mechanism for integrated communication and computation, Proc. of ISCA, 1992.
- IP to the sensors
  - G. Montenegro, N. Kushalnagar, J. Hui, D. Culler, Transmission of IPv6 Packets over IEEE 802.15.4 Networks, RFC4944, 29 pages, September 2007
- Content-based addressing
  - A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Content-based addressing and routing: A general model and its application. Technical Report CU-CS-902-00, Department of Computer Science, University of Colorado, Jan. 2000.