On Smart Spaces

Status and Challenges in the Programming of sensor networks

Johan Lukkien
System Architecture and Networking
Eindhoven University
Computer Science groups

- Section Information Systems (IS)
  - Expertise Group Databases and Hypermedia (DH)
  - Expertise Group Architecture of Information Systems (AIS)

- Section Specification and Verification (SV)
  - Expertise Group Formal Methods (FM)
  - Expertise Group Design and Analysis of Systems (OAS)

- Section Software and Systems Engineering (SSE)
  - Expertise Group Software Engineering and Technology (SET)
  - Expertise Group System Architecture and Networking (SAN)
  - Expertise Group Security (SEC)

- Section Algorithms and Visualization (AV)
  - Expertise Group Algorithms (ALG)
  - Expertise Group Visualization (VIS)

29% promotions
13% papers
96% patents
33% industrial funding
System Architecture and Networking

Coordinated Applications  Predictable Platforms  Embedded Computations

Resource-constrained networked embedded systems

LaQuSo
Laboratory for Quality Software

QoS for multimedia in home networks

content queries on security movies

monitoring herd behavior

IST WASP

Toolchain for macro-programming

Controller Area Network (CAN) schedulability analysis: Refused, revisited and revised

The Guide to Computing Literature

Technische Universiteit Eindhoven
University of Technology
# System Architecture and Networking Group

<table>
<thead>
<tr>
<th>Johan Lukkien</th>
<th>Kees van Berkel</th>
<th>Antonio Liotta</th>
<th>Rudolf Mak</th>
<th>Reinder Bril</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shudong Chen</td>
<td>Tanir Ozcellebi</td>
<td>Dmitri Jarnikov</td>
<td>Waqar Aslam</td>
<td>Cecile Brouwer</td>
</tr>
<tr>
<td>Richard Verhoeven</td>
<td>Melissa Tjiong</td>
<td>Remi Bosman</td>
<td>Mike Holenderski</td>
<td></td>
</tr>
<tr>
<td>Natasa Jovanovic</td>
<td>Shervin Haji Amini</td>
<td>Ugur Keskin</td>
<td>Sachin Bhardwaj</td>
<td>Martijn van den Heuvel</td>
</tr>
<tr>
<td>Tse Batsuuri</td>
<td>Cagdas Atici</td>
<td>Ionut David</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking
Smart Spaces
Smart Spaces

- Smart Space: a (living) space with
  - embedded technology...
    - ‘regular’ CE & computing devices
    - smart nodes: sensing, actuating, communicating, computing
  - ...that is networked...
    - wireless, mostly
  - ...designed to collaborate and share...
  - .... to interact with / serve users...
    - maximum autonomous operation
    - provide services with minimal user effort
  - ... and the electronics these users carry
    - e.g. body sensor networks, personal electronics
Examples

• Lighting:
  – adaptive lighting
    • preferences coming from user, automatically brought into the space
  – lights as output medium
    • e.g. warning, from outlook appointment

• Sensing:
  – gesture recognition, voice commands
    • to control elements (actuators) in the space
    • to use as input media for private purpose
  – long-term monitoring
    • patient observation, elderly monitoring
Scenarios in a smart space

• Focus not just on functionality
  – given enough time and money we can make just about anything

  Can we make system components such that future, as yet unforeseen, cooperations and adaptations are simply realizable, and actually work?

• Include change scenario’s, e.g.,
  – new devices / users enter the space
    • how to understand the messages?
    • how to generate and install new functionality (e.g. automatically and invisibly coming from/to the new device)
    • how to become part of the space?
  – several users sharing a device
    • how to manage this sharing? access control?
  – saving energy
Smart space requirements

• Smart Space requirements
  – *programmable and configurable* – wished behavior can be imposed
  – *sharing of services* – by concurrent (distributed) applications
  – *separation of coordination and service* – separate application logic from device functions
  – *integration with the regular IP infra structure*

• Basic functionalities provided by nodes to *the network*
  – *‘Code’ load* – to define services, configuration
  – *Discovery* - detection of nodes, services, resources
  – *Service/Resource Monitoring & Management*
  – *Service composition/coordination*– by third party

• Lightweight architecture: how to provide this with low-capacity nodes?
Programming
OK, but how does this work?

- How to use (program) and share the ‘terminals’ in the space?
  - standardized usage profiles
    - e.g. Zigbee, Bluetooth, perhaps UPnP
      - cannot easily share
      - cannot easily use the distributed compute power
  - centralized access point
    - admits management
      - sharing, access control
    - can form physical gateway
      - leveraging e.g. 6lowpan (IPv6 to all terminals)
    - can add semantics
    - needs at least connectivity management within the low-resource wireless domain (e.g. 802.15.4)
    - latency, complexity, mobility, node diversity
Three levels of abstraction

Node level abstraction
Applications on top of a small OS
Profiles, or specific programs

Application level abstraction
Specifies combined node behavior
Determines local programs and connections between nodes
- e.g. patient monitoring

Network level abstraction
Message semantics
- layered stack
- or application-specific payload

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

12
Three levels of abstraction

• **Node level**: application (part) on node
  – the overall application ‘emerges’ as the collaboration of these
  – needs good network abstraction for programmers

• **Network level**: define messages, and semantics
  – supporting layered protocol stack or
    • good for abstraction, bad (?) for performance
  – define the overall application by specifying messages and response to these
    • e.g. along the lines of UPnP

• **Application level**: focus on overall task, e.g. light scenario
  – leads to construction of local programs (and perhaps messaging) realizing this
‘Traditional’ development applied to sensors

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking
Choices in (re)programming

- **Compact code**
- **Machine & OS independence**
- **Interpreter on nodes**
- **Good for coordination code**

- **Large code**
- **Machine & OS dependence**
- **Gives most efficient run-time**

**Deployment options**

- **Before deployment**
  - node-specific code
  - simpler, for a node

- **After deployment**
  - all nodes same code
  - requires selection mechanism on node

**Code specialization per node**

- **Program for network**
  - **Compiler**
  - **Intermediate code (e.g. bytecode)**
  - **Interpreter**
  - **Jit / assembler**
  - **Machine code**
  - **Linkloader (add libs+OS)**

- **Runnable**

- **Deployment options**
  - **Static configuration**
    - config info given once
    - useful mainly for static deployment (utmost performance)
  - **Dynamic configuration**
    - config info input to running program

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

• Virtual machines

• Special-purpose engine

• Macro-programming
Approaches (cnt’d)

- **Active messages**

- **IP to the sensors**

- **Content-based addressing**
OSAS @ TU/e
OSAS Framework

• Open Service Architecture for Sensors (OSAS)
  – *language*, to program the entire network (‘macro-programming’)
    • with concepts of a *service*, *events* and *subscription*
    • and *content-based addressing*
  – *virtual machine definition* (byte code ISA)
    • has access to a set of *system calls* (OS-provided functions)
  – *message format*

• Four tools
  – *Compiler*
  – *Loader*
  – *Runtime system*
  – *Simulator*
    • runs same runtime system

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

WASP Project IST.034963
OSAS Framework: *Programming Model*

- Nodes run services:
  - A service has event *generators* and event *handlers*.
  - A link between a generator and a handler is called a *subscription*.
  - Interaction between services by a *asynchronous remote procedure call*
    - upon firing of the event
    - or simply as part of a handler

- WASP Service Composition Language (WaSCoL)
  - Specification of services with *event-condition-action* rules
    - condition is evaluated in a time-triggered fashion; when true, fires the action which may call a handler
  - Composition of services (i.e. subscriptions)
  - Use content-based addressing to associate node with service
  - Programs can be developed and loaded incrementally
Run-time organization

OS access + computational library

Publisher (e.g. Temp service)
Subscriber (e.g. a temp value accumulator)
Management
byte code interpreter
system calls
service creation
message handling

OS API
e.g. sensor inspection

OS

send (dest, ...) receive (...)
link layer

Hardware

Network interface

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking
Connection by subscriptions

subscription, defines sampling rate and remote handler to call
Language

Node 1 & 2

Address table:
0 : true  // HasSysCall(Temp)
1 : false // NodeID==3
2 : true  // HasService(TempService)

Sys Calls
- HasSysCall
- Temp
- NodeID

TempService

Service TempService($\text{Handler})

on event read_temp when True do
SendToSubscribers($\text{Handler}, \text{Temp}())

subscription AverageTemp
to TempService($\text{Handler}=\text{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)]

Node 3

Address table:
0 : false // HasSysCall(Temp)
1 : true  // NodeID==3
2 : false // HasService(TempService)

Sys Calls
- HasSysCall
- NodeID

AvgTempService

event flush_avg

action ReportTemp

Message

ReportTemp14

State
sum count

AverageTemp
$\text{Handler}=\text{ReportTemp}$

event read_temp

Node 3

Message Subscribe

Temp

Temp

Temp

Temp

Temp
Three levels of abstraction: summary

• **Node level**: application (part) on node
  – a runtime system, with byte code interpreter and service installer

• **Network level**: define messages, and semantics
  – generic WSP format: [header ; (handler, arguments)]
    • specifies the asynchronous remote procedure call
    • specifics *derived from the program* by the compiler

• **Application level**: focus on overall task, e.g. light scenario
  – single program for entire network
  – generates collection of service definitions and subscriptions as *separate messages* to be loaded into the network
  – content-based addressing leads to loading the program *just once*
    • nodes use CBA to determine their part
Q & A
• How can this approach limit communication energy?

• How general is this? Does it need standardization?

• What about multi-hop?

• How do two different programs ‘live together’?

• How do I extend an already running system?

• How to discover nodes and services?

• How to connect to IP?
How can this approach limit communication energy?

- Needs: an interface to pass timing information...
  - derived from subscriptions...
  - ... and neighbor information
- ... and a MAC protocol that uses this
  - e.g. WISEMAC, TRAWMAC
How general is this? Does it need standardization?

- The messaging assumes a *link*; it can be bound to any link
  - a LLC/MAC protocol
  - a 6lowpan transport overlay
  - a general IP overlay
  - Zigbee messaging
What about multi-hop?

- A subscription defines a source-destination route
  - mobility means: re-subscribing
- The language definition is powerful enough to specify general routing
- We have used it to implement a few common routing protocols
  - gradient routing
  - path reconstruction (mobility)
  - shortest paths spanning tree
  - source routing
How do two different programs ‘live together’?

• Message headers contain an ‘application id’
• Service $s$ can now be addressed using ($app_id$, $s$)
How do I extend an already running system?

• The xml-file is:
  – *output* of the compiler – describes system symbols, their meaning and their mapping to integers
  – *input* to the compiler: as a description of the running system
How to discover nodes and services?

• Inject a small heartbeat service, for all nodes
  – reporting installed services, subscriptions, system calls etc. at regular intervals
• Subscribe to it
• Interpret using the xml output of the compiler

Johan J. Lukkien, j.j.lukkien@tue.nl
TU/e Informatica, System Architecture and Networking
How to connect to IP?

- Application-level gateway: translation, including semantics
- IP overlay sensor networks: e.g. 6lowpan
  - can connect transparently to sensors – may lead to ‘expensive’ protocols (SOAP)
- sensor networks overlay IP
  - nodes are simulated on IP; there, powerful system calls are provided
Concluding
Was it useful?

• Light control system
  – actively controlling light intensity according to user preference (coming from new device in the space)

• On-node ECG processing
  – determine timed heart beats
  – combine with accelerometer to have better diagnosis

• Connect a light to the heartbeats 😊
Was it useful?

• Herd monitoring in the WASP project
  – use observations to adjust subscriptions
    • e.g. animal moving -> step detection
  – full in-network processing
  – full over-the-air programming
  – context dependence:
    • inside barn: all nodes reachable
    • outside: use routing

• Smart space resource management architecture
Conclusion

• An integral approach to programming wireless sensor networks
  – node, network, application level
  – limit errors, fast program construction and deployment
  – over-the-air programming, with single image
  – in fact, more general than just sensor networks

• Concepts of services, events, subscriptions, content-based addressing

• Admits mapping onto existing systems

• Addresses requirements of smart spaces
  – also: flexibility and agility
Ongoing work

• Improve security, access control, privacy

• Examine more applications
  – lighting, in-network processing
  – experiment with keeping generated information internal, and use in control decisions

• Complete an energy-efficient MAC