

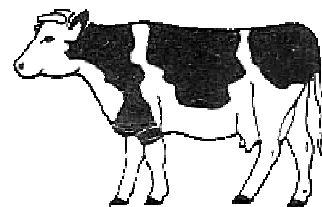
DRIVING FORCES BEHIND MIDDLEWARE CONCEPTS FOR WIRELESS SENSOR NETWORKS

Presentation for RealWSN 2005
Workshop on Real-World Wireless Sensor Networks
Stockholm - June 2005

Kirsten Terfloth
Computer Systems and Telematics Group
Freie Universität Berlin
Prof. Dr.-Ing. J. Schiller

MIDDLEWARE – THE GLUE

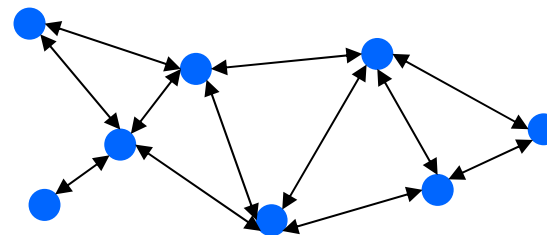
Application, Problem to be solved
= high-level abstraction



Middleware = Mediator



System-oriented issues,
distribution
= low-level abstraction



CHALLENGES FOR APPLICATION PROGRAMMERS

- Embedded Devices
 - ➔ Resource constraints [energy, memory, ...]
 - ➔ Possible failure of nodes
- Distribution
 - ➔ Distributed data sources
 - ➔ Decentralized (self-)organization of the network
 - ➔ Localized algorithms
- Dynamic Scenarios [Topology, Environment, Application]



SURVEY OF EXISTING APPROACHES

- Multitude of programming paradigms to fill in the gap
 - ➔ Event-driven, Distributed Databases, Mobile Agents
 - ➔ Publish/Subscribe Systems, ...
- Multitude of functionalities these approaches provide
 - ➔ Distribution of code updates
 - ➔ Threading
 - ➔ Routing
 - ➔ Data management,



CLASSIFICATION

- What is the abstraction, an application programmer is offered using the middleware approach?
- Categories:
 - ➔ Virtual Machine Abstraction
 - ➔ Group Abstraction
 - ➔ Modular, Service-Oriented Architecture
- Categories neither discrete nor disjoint
 - ➔ Approaches can feature some/none/all characteristics of the class



CATEGORY1: VIRTUAL MACHINE ABSTRACTION

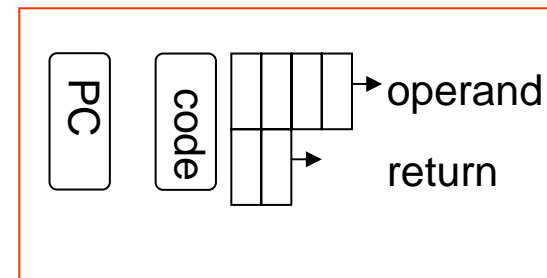
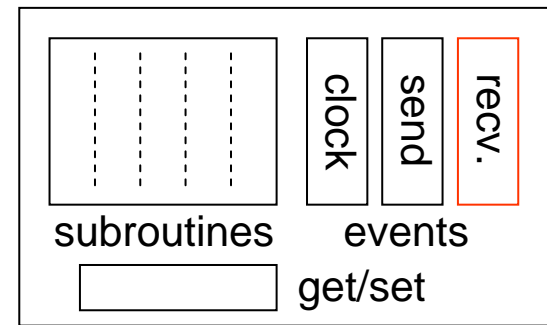
- Hides complexity originating at the underlying layer
 - ➔ Pure firmware
 - ➔ Complete operating system
- Provides safe execution environment for application code
- Often tailored language to the needs of the WSN
 - ➔ Encapsulation of characteristic commands
 - ➔ Special datatypes for sensor readings
 - ➔ Dense byte-code for optimized byte-code distribution



EXAMPLE FOR VM

Maté (Lewis and Culler, UCB 2002)

- TinyOS component Stack-oriented byte-code interpreter
- Fixed-size instructions
- Inherent code distribution via one command and code capsules (24 instructions)

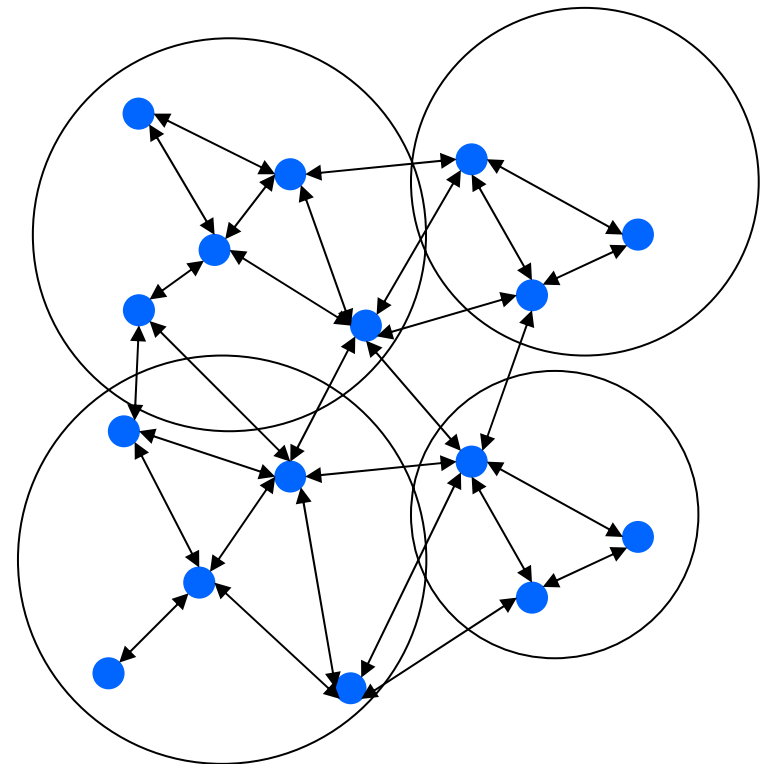


context



CATEGORY2: GROUP APPROACHES

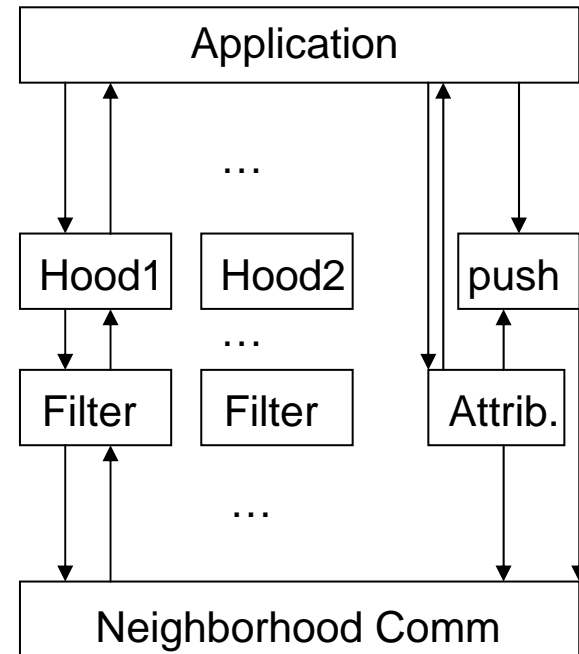
- Focus on network, not nodes
 - ➔ Groups = organizational entities managed by MW
 - ➔ Groups determined by various attributes [network proximity, sensors, ..]
- Aims at scalability, support for heterogeneous hardware, distribution issues



EXAMPLE FOR GROUP APPROACHES

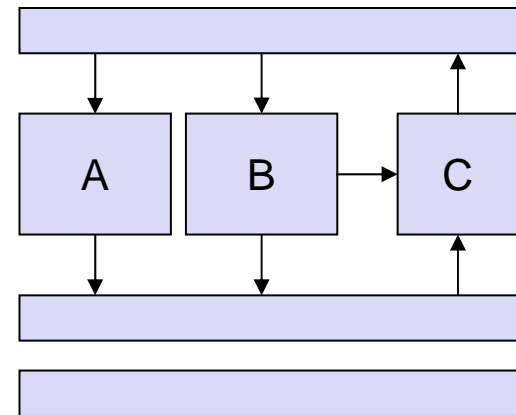
Hood (Whitehouse, Brewer and Culler, UCB 2004)

- Neighborhood = programming primitive
 - ➔ Attributes to be shared
 - ➔ Membership criteria
 - ➔ Interface to neighbors and values
 - ➔ Filter & broadcast



CATEGORY3: MODULAR, SERVICE-ORIENTED APPROACHES

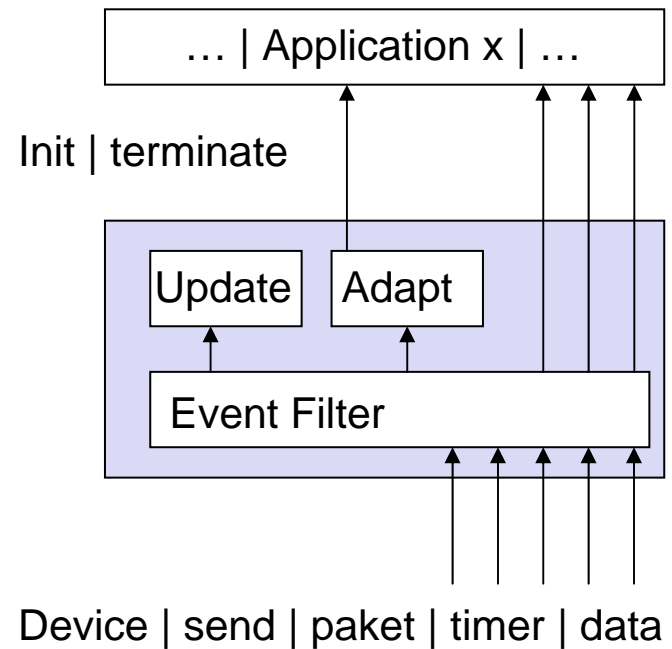
- Hides complexity arising from distribution and communication
- “Classic” solution
 - ➔ Offer a well-defined interface to invoke a set of “basic services”
 - ➔ Hide implementation
- Modularity necessary for adaptation to different applications



EXAMPLE FOR MSOA

Impala (Liu and Martonosi,
Princeton 2003)

- Event-centric architecture
 - ➔ Event and device manager
 - ➔ Application adaptability



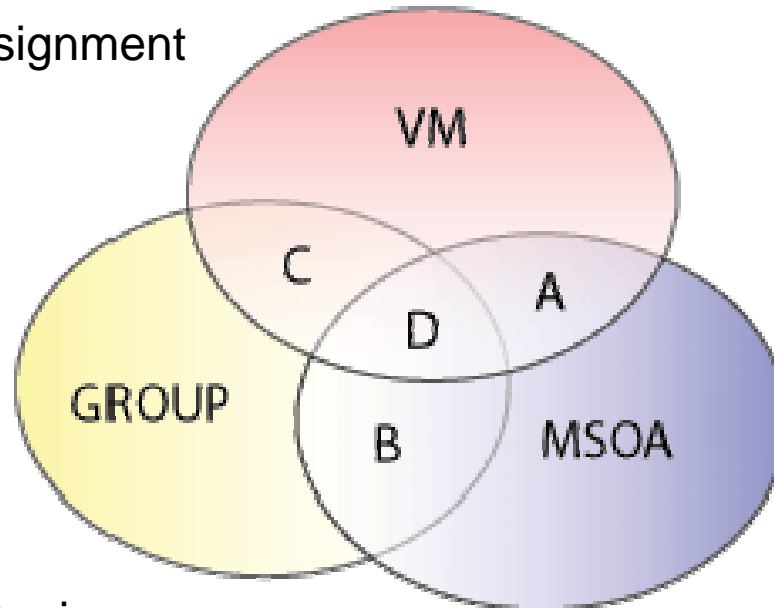
CLASSIFICATION OF EXISTING APPROACHES

C: Generic Role Assignment

A: SWARMS
SensorWare
TinyDB

B: Event-Entities
Issues in MW

D: ?



Maté
ScatterVM

Hood
Abstract Regions
RoamHBA

Milan
TinyGALS



VIRTUAL MACHINES REVISITED

- Two developments of VM in student project | master thesis
- Developed for FU sensor nodes (ESB)
 - ScatterVM
 - ‘Maté’ Approach [assembler style progr.]
 - Small byte-code image
 - VM + IDE
 - RealMachine
 - Graphical programming
 - Event-centric development



SCATTER VM

The screenshot shows the ScatterVM IDE interface. The main window displays assembly code for a file named 'test.asm'. The code includes instructions like MOV, CMP, JNE, PUSH, CALL, and ADD with various comments. A status bar at the bottom indicates 'Run completed successfully.' To the right, a smaller window titled 'Predefined Labels I/O Ports Interrupt Routines' shows a list of predefined labels and routines, including ADD and AND.

Assembly Code (Left Panel):

```

MOV &4096 $10      # read in immediate value
CMP &4096 $10
JNE Error
MOV &4097 &4096    # move memory to memory
CMP &4097 $10
JNE Error
PUSH $2 # sound test -> tonleiter
CALL SYS_SOUND
MOV &4096 $6       # add 6 and -6
MOV &4097 -$6
ADD &4096 &4097
CMP &4096 $0
JNE Error
MOV &4098 $20000   # add
MOV &4099 $25000
ADD &4098 &4099
CMP &4098 $0
JGE Error
MOV &4100 $25      # add
ADD &4100 $30
CMP &4100 $55

```

Predefined Labels I/O Ports Interrupt Routines (Right Panel):

```

MOV &4096 $10      # read in immediate value
CMP &4096 $10
JNE Error
MOV &4097 &4096    # move memory to memory
CMP &4097 $10
JNE Error
PUSH $2 # sound test -> tonleiter
CALL SYS_SOUND
MOV &4096 $6       # add 6 and -6
MOV &4097 -$6
ADD &4096 &4097

```



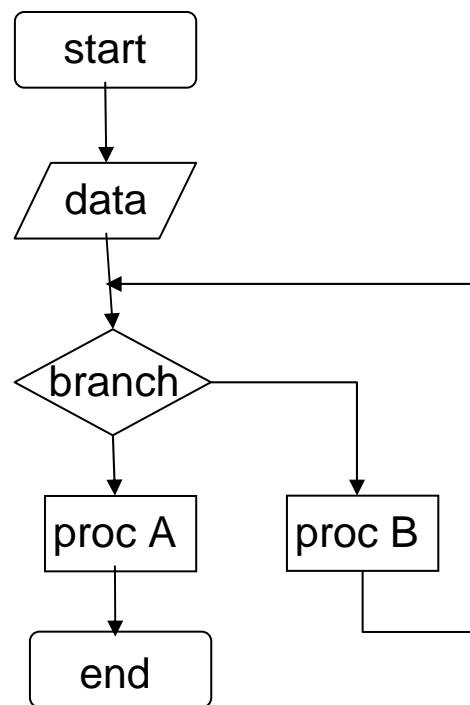
SCATTER VM

- CISC, memory-to-memory architecture
- 16Bit addresses | 8Bit opcode | 16 Bit parameter
- Inspiration Intel 8086 ISA + Interrupt handling
- Strict segmentation of code and data
- Size:
 - EEPROM 6152 Bytes
 - RAM 33 Bytes
- 130 IPS, sandbox byte-code verification

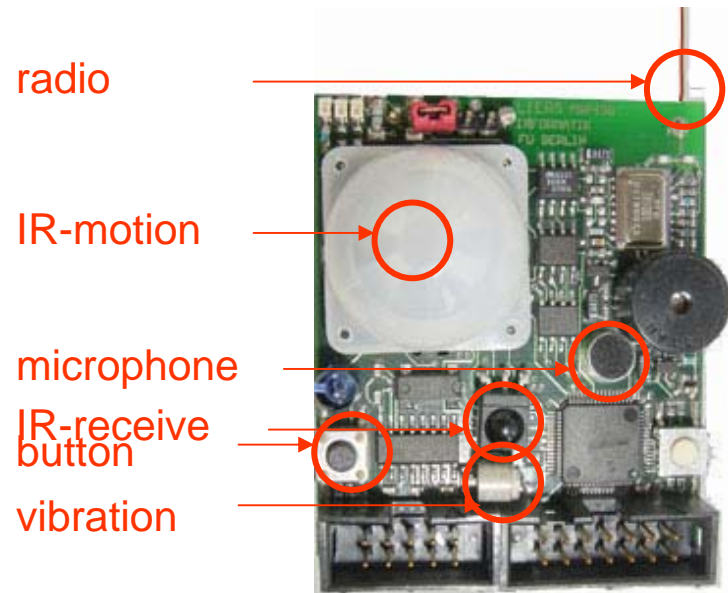
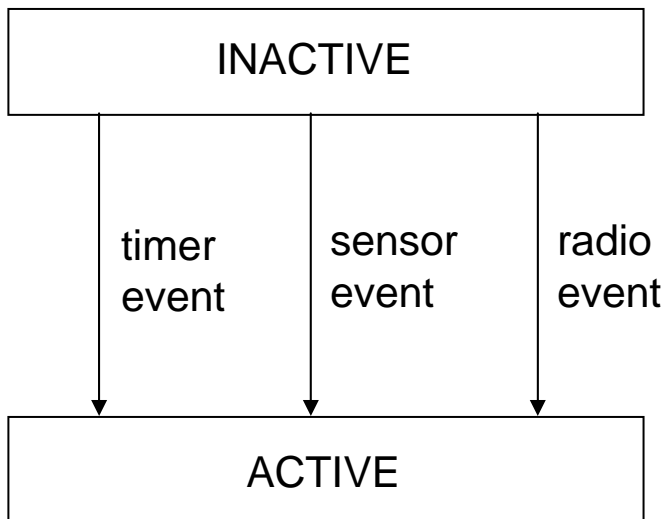


REAL MACHINE

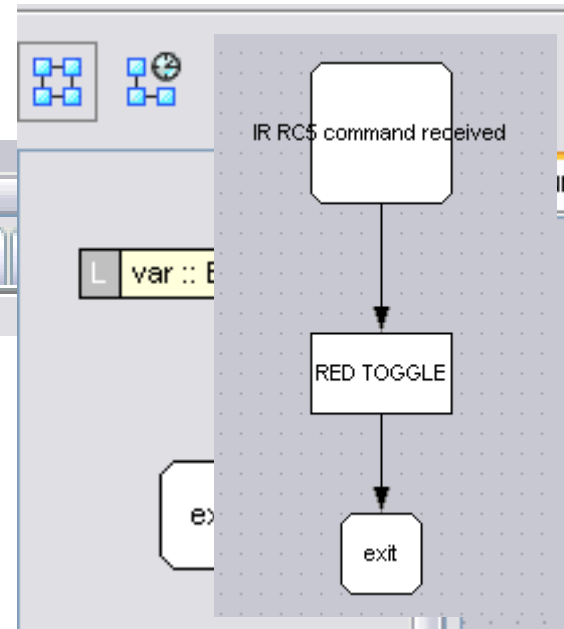
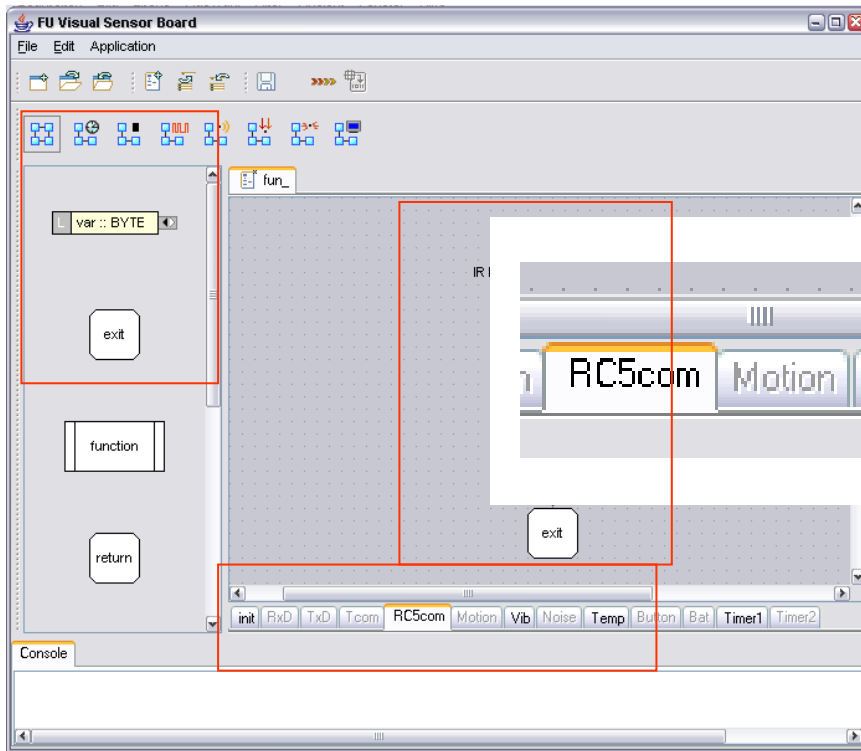
- Idea: Event-centric, graphical programming
- Use flow-chart language
 - ➔ Established symbols
 - ➔ Good for “small” programs
- Separate program and data flow
- Call of one procedure per task



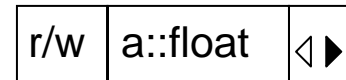
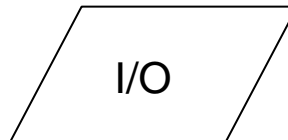
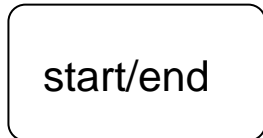
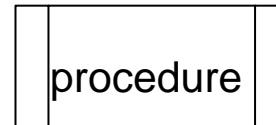
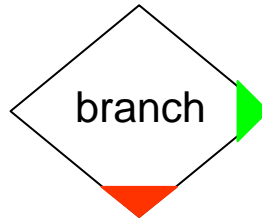
ACTIVITY MODEL



REAL MACHINE



REAL MACHINE: SYMBOLS



variable/const
data declaration



REAL MACHINE TASKS

- Task table central to VM
 - ➔ For all possible tasks one entry
 - ➔ enabled | synchronized | state | prio | addr | pc | ra
 - ➔ Block interleaving multithreading
 - Block size ~ priority
 - Round robbin
- No task – VM shutdown (low-power)



REAL MACHINE RESULTS

- Program is stored in EEPROM
 - ➔ Instruction and parameter fetch need access
 - ➔ Test of issue one instruction (oszilloscope)
 - OS superloop | VM inaktiv - 20 us
 - OS superloop | VM aktiv - 26 us
 - Load of one instruction - 752 us
 - Load of instr. + PC incr - 1460 us
 - ➔ VM is rather slow
- Usefulness has still been proven in real life

