Sensor Network Platform

DR Application

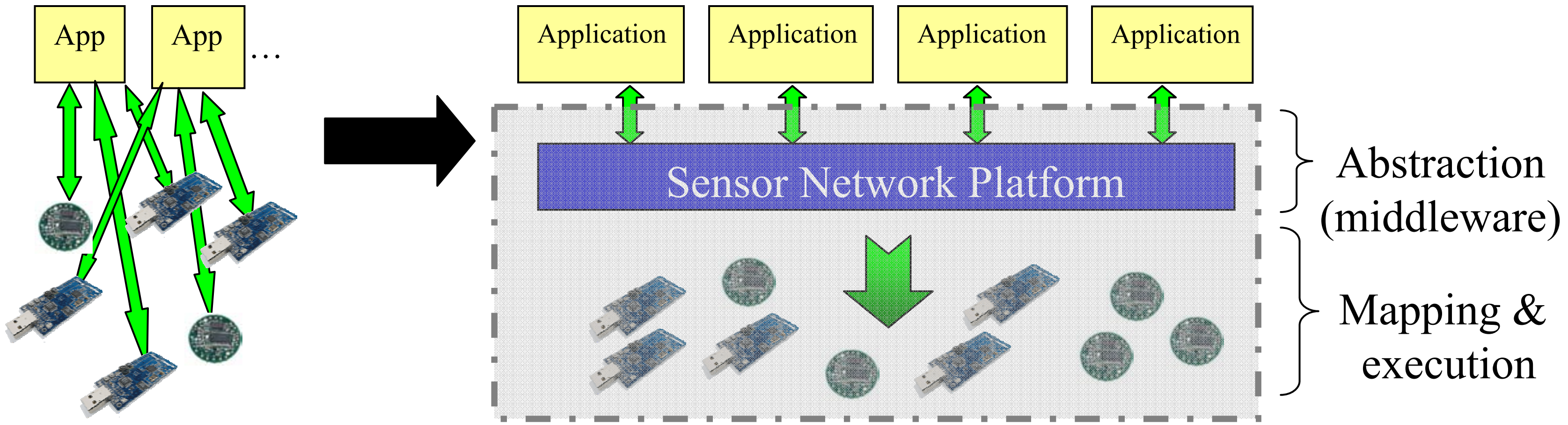
Presented by Prof. Paul Wright
On behalf of Jana van Greunen and Prof. Jan Rabaey

An Analogy

- When you want to write a letter you want to fire-up Microsoft word and get going…
- You don't care if you are using an HP laptop, a Dell, a Sony, an IBM….
- You don't even care if it's a Mac
- So … this talk focuses on the Applications that any home might want to run on any type of mote or hardware…

Sensor Network Platform (SNP)

- Remove burden from programmers by:
  - Providing a clean programming paradigm
  - Abstracting distributed implementation
  - Mapping application onto network at runtime (future work)



Centralized “virtual uni-processor” abstraction

Note: Inputs/outputs are inherently “distributed” in space

SNP and DR Project Goals

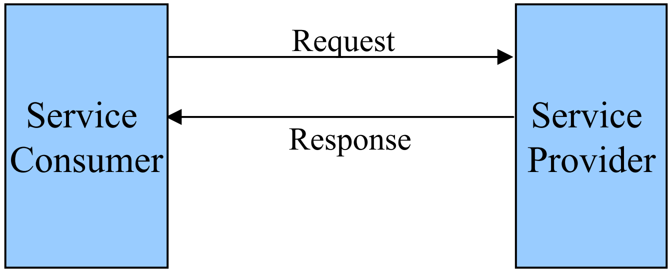
- Provide a unifying framework that makes application programming easier
- Enable code reuse and modular applications
- Capability discovery
- Investigate SNP feasibility
  - Implement a Demand Response (DR) application
  - Use multiple hardware platforms (many mote types)

Assumptions

- Nodes share a common notion of time
  - To send data on regular basis
  - To decide when to sample the environment, or to actuate
- Nodes have location data
  - Nodes know their own locations
  - Nodes know the locations of services they wish to use / can interpret semantic locations, e.g. kitchen, living room
- (Note: time and location is called *scope*)
- From the applications’ perspective, addressing is geographic (this may be translated by SNP into underlying addressing)

Platform Architecture

- The SNP has a service-oriented architecture
  - Decouples function from implementation
  - Decouples service provider from consumer in time/ space
- What is a Service?
  - A function that is well-defined, self-contained, and does not depend on external context or state (For example my thermostat is a “consumer” because it “uses” the heating provider in my basement.)

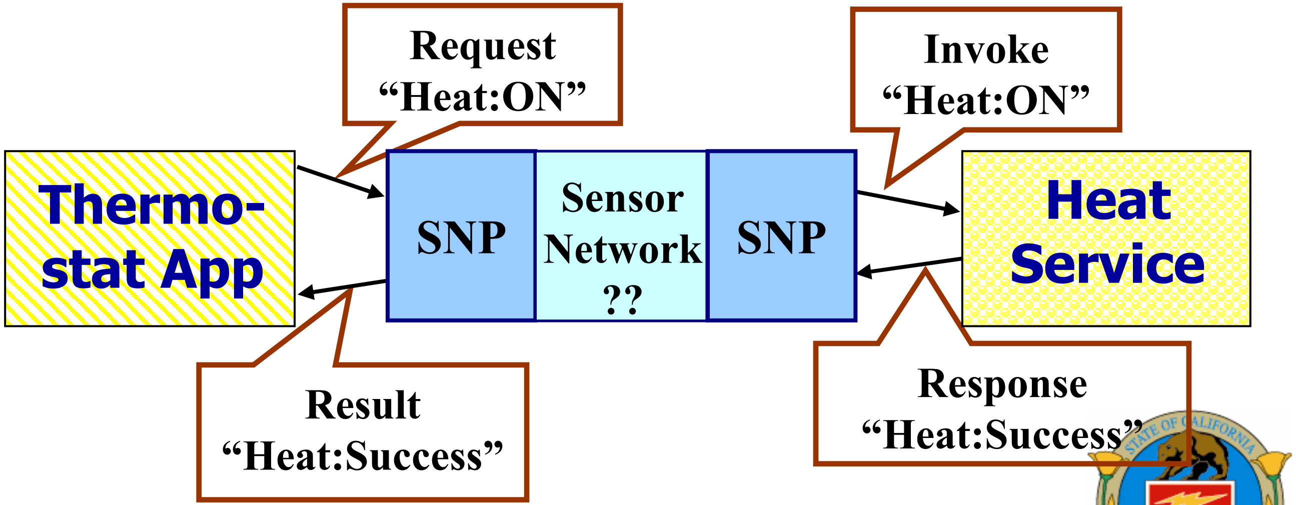


*White paper: “Sensor Networks Services Platform,” M. Sgroi, A. Wolisz, J. Rabaey

Service Invocation

- Invocation starts service execution (instantiation)
  - Arguments: scope & service type
  - Scope is translated to underlying addressing & routing

- Example: Heat service invocation



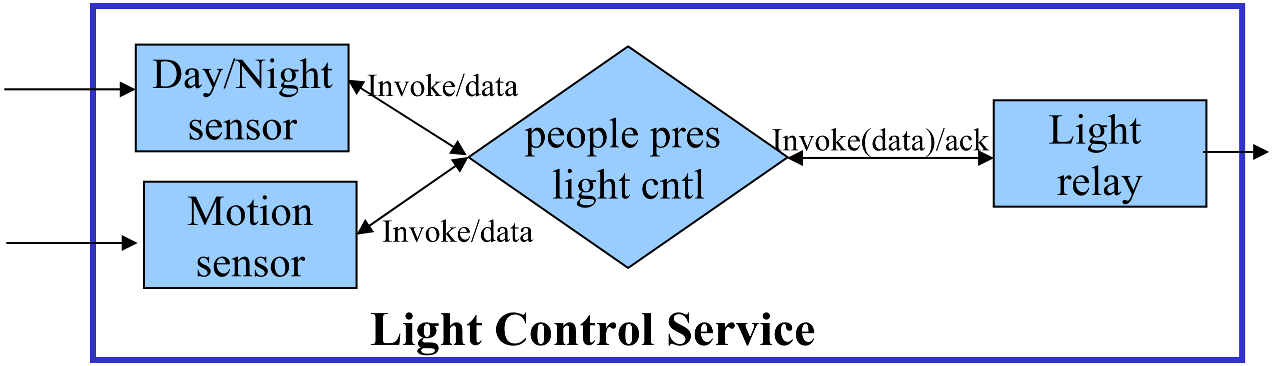
Service Invocation Structure (e.g. in TinyOS running on the nodes)

- Header
  - Requested “provider service”; originating “consumer service”
  - Destination scope (e.g I need heat in the kitchen at 6pm); originating scope (e.g From my thermostat near the front door)

- Body
  - Set of named functions: $(x,y\ldots) = f(a, b,\ldots)$
    - E.g. choosing from a number of possible actions/modes – E.g “give me current price” or “give me total usage over period $t_1>t_2$”
  - Upon invocation/request the functions’ arguments are specified
  - Upon response/result, the functions’ results are filled in

Another specific example…

- An application is a *task* graph of services used and some computation



- Applications become services when they are registered in the *repository*

- Application may export a “Service API”
- Analogy to object-oriented programming
  - Function description (computation) with inputs/outputs

Capability Repository (CR)

- Repository of all available services
  - Services register with the CR by exporting a standard API
  - Applications query the CR to discover new services
- Service API contains
  - A unique ID (within a given scope)
  - A set of "callable" functions with typed arguments & results
  - List of hardware/services used
  - Analogy to a 'remote procedure call'
- Example entry:

<u>People Detector</u>

Type: service

Callable function

bool = AnyoneThere(location, time)

Hardware Used: ADC, temp sense, Motion Service

Implementation

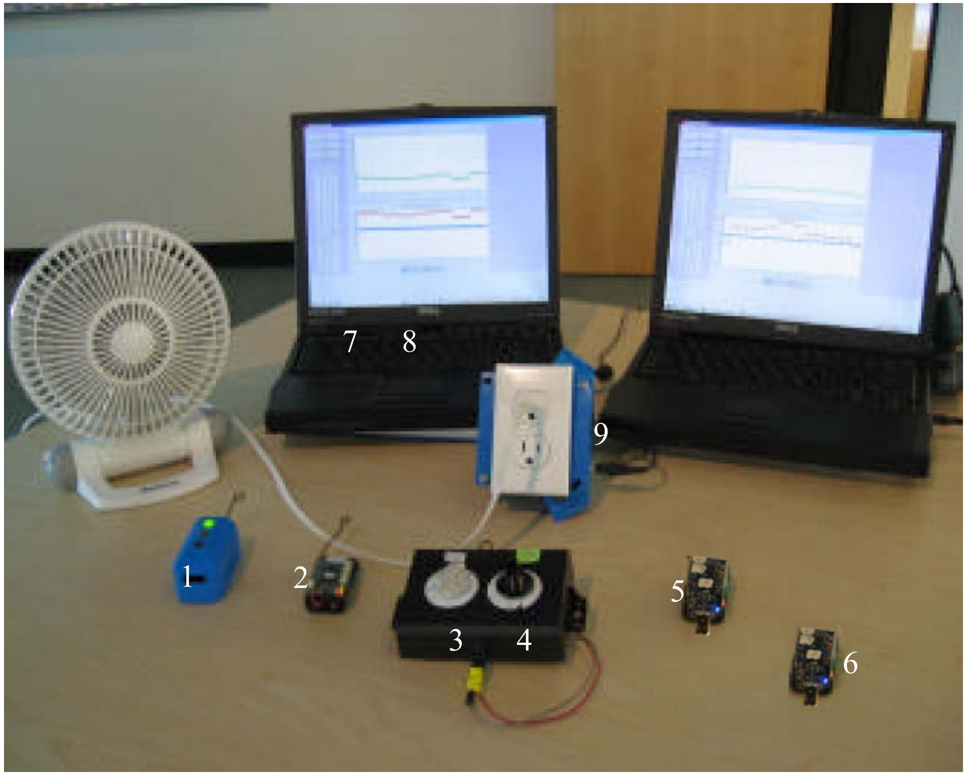
- Prototype to evaluate SNP's:
  - Programming paradigm
  - Modularity & portability (using more than one network)

- Hardware: Mica2 from Crossbow ++ TelosB from Moteiv
- Software: The SNP is written in nesC*
- The application is written just once --- but the middleware is customized for the hardware & software of each "mote"
- Application: Demand Response
  - 4 Bedroom house
  - Demand-driven electricity prices are provided by a price sensor
  - House is outfitted with a smart thermostat, price indicators on appliances, and appliance switches; these are controlled by a sensor network

*NesC is the language developed for TinyOS

Hardware & setup

Node	Type	Application
1	Mica2	– Price Indicator Service
2	Mica2	– Temperature Sensor Service – HVAC Control App
3	Mica2	– Comfort Sensor (smart t-stat)
4	Mica2	– Desired Temperature Service
5	TelosB	– Price Indicator Control App
6	TelosB	– Price Service
7	TelosB	– Display – Bridge Service
8	Mica2	– Display – Bridge Service
9	Mica2	– HVAC Switch Service
x	M/T	– Temperature/Humidity Service to test discovery



Central Capability Repository

- Repository is stored on the laptop & starts empty
- Applications/services register on instantiation
- Repository is soft-state (entries older than 25s are deleted)
- Supports 5 operations: *registerservice*()*, *queryAll*()*, *queryService*()*, *queryScope*(service), and *queryNetwork*(service)
  - » *E.g "I am the price indictor service, here I am"
  - » *E.g "Who else is out there?"
  - » *E.g "Query the heater to see if it's binary or will take temp. settings"
- Service Discovery:
  - On start-up, applications send *queryAll* messages
  - For a particular service, *queryService* returns the invocation format
  - Applications periodically query the CRS for new services

Findings

- Pros
  - Service-oriented platform enables application code to be portable to a wide range of platform types
  - SNP is lightweight enough to be installed on resource limited networks

- Cons
  - This design lacks:
    - Personalization to members of the household
  - Service oriented paradigm cannot provide:
    - Automatic fault recovery
    - Flexible/automatic application deployment for each node – need to hand-load code on motes (planned in future work)

Conclusion

- Presented the Sensor Network (Service) Platform (SN{S}P)
  - A service-oriented abstraction for sensor network programming
  - Enables creation of modular code

- Achievements
  - Implemented the SNSP platform on Mica2 and TelosB motes
  - Demonstrated that the SNSP does enable application code to be portable (multi-platform demonstration)

- Findings/future work
  - Basic services and be run on resource limited networks
  - The SNP needs more flexible deployment beyond “hand loading”