
Distributed Systems – Techs

12. Ubiquitous* computing

yoo-**bik**-wi-tuh s, omnipresent

Ubicomp definition

- Term coined by Mark Weiser in 1991
 - Vision: world of ubiquitous computers that become invisible by being embedded into the physical environment with the goal of supporting people unobtrusively in fulfilling their tasks
 - Named also ***pervasive computing***
 - Example for an application of ubicomp: a *smart* room allocation syst.
 - chairs are able to sense their occupancy status and use this information to automatically derive the occupancy level of the room, which is then displayed at the electronic door plate and by a central “room finder” in the hallway.
 - Technological features:
 - consists of numerous, highly specialized wireless computing devices embedded into our physical environment
 - devices can perceive & control certain params.of their physical environment & can communicate with each other.
 - devices use ergonomic, intuitive, & unobtrusive ways of interacting with people.
-

Why now and what?

- Recent technological advances supporting ubicomp in
 1. processors,
 2. storage,
 3. wireless communication,
 4. sensors and actuators,
 5. energy supply,
 - Construction of energy-efficient technologies is of utmost importance (devices for ubicomp often are wireless).
 6. development of new materials
 - New materials (e.g. flexible displays, film batteries) allow the construction of devices with unconventional forms.
 - Trend towards “more, smaller, cheaper, less energy” will enable the construction of future ubicomp systems.
 - A computer science perspective:
 - new algorithms, protocols & architectures needed to
 - manage & control the enormous amount of networked computing devices &
 - make sense out of the huge amount of data collected by sensor-equipped devices
-

Middleware challenges

- Constrained Resources
 - Network Dynamics
 - Scale of Deployment
 - Real-world Integration
 - Collection, Processing and Storage of Sensory Data
 - Integration with Background Infrastructures
-

Constrained Resources (1/2)

- To allow an unobtrusive integration into physical objects and environments -> devices often have to be wireless and must meet certain size constraints.
- Limited size and energy => resources like computing power, memory size, communication bandwidth, and range are rather limited.
- Example: a matchbox-sized sensing device developed at UC Berkeley, MICA mote
 - ❑ 8-bit processor with 8 MIPS,
 - ❑ 8 kilobytes of RAM,
 - ❑ 128 kilobytes of program memory
 - ❑ a communication bandwidth of 40 kilobits per second over a range of up to 30 meters
 - ❑ runs for weeks or months on a pair of AA batteries.

MICA
WIRELESS MEASUREMENT SYSTEM

- ▼ 2nd Generation, Tiny, Wireless Smart Sensors
- ▼ TinyOS - Unprecedented Communications and Processing
- ▼ AA/Year Battery Life
- ▼ Small Form Factor
- ▼ Wireless Communications
- ▼ Light, Temperature, Acceleration/Seismic, Acoustic, and Magnetic Sensors
- ▼ Developed for DARPA NEST Program

Applications

- ▼ Wireless Sensor Networks
- ▼ Security, Surveillance, and Force Protection
- ▼ Environmental Monitoring
- ▼ Large Scale Wireless Networks (1000+ points)
- ▼ Distributed Computing Platform

The MICA Mote is a second generation mote module used for research and development of low power, wireless, sensor networks. The MICA mote was developed by UC Berkeley's research group on wireless sensors. It consists of:

- Plug-in sensor boards
- TinyOS (TOS) Distributed Software Operating System.
- Atmega 128L processor
- 916MHz or 433MHz transceiver
- Attached AA(2) battery pack

TinyOS is a small, open-source, energy efficient, software operating system developed by UC Berkeley which supports large scale, self-configuring sensor networks. The source code and software development tools are publicly available at: <http://webs.cs.berkeley.edu/tos>

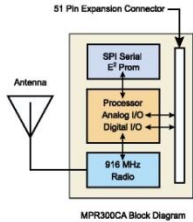

TOS includes:

- Radio messaging
- Message hopping from mote to mote
- Low power modes
- Sensor measurements and signal processing

Processor and Radio Platform (MPR300CB): The MPR300CB is based on the Atmel ATmega 128L. The ATmega 128L is a low power microcontroller which runs TOS from its internal flash memory. The 128L has been selected for its low power and other features. The MPR300 (MICA) uses an ISM band radio transceiver module for wireless communication.

Sensor Boards: Various sensor boards are available from Crossbow and other sources. These boards connect onto the MICA through a surface mount 51-pin connector. The 51-pin connector supports Analog Inputs, I2C, SPI, UART, and a multiplexed Address/Data bus. These interfaces make it easy to connect to a wide variety of external peripherals. Crossbow supplies the following sensor boards:

- MTS101CA Photo diode/Thermistor/Proto and Experiment Board
- MTS300CA Photo diode, Thermistor, Microphone, and Sounder
- MTS310CA Same as MTS300CA but also including Magnetic and Acceleration Sensor



MPR300CA Block Diagram

80 Document Part Number: 6020-0041-01

crossbow technology, inc • 41 daggett drive • san jose, ca 95134-2109

Constrained Resources (2/2)

- The limited resources must be shared among various applications executing in the network and the middleware services itself => ubicomp middleware services must be lightweight in order to fit into the constrained resources
- Ubicomp middleware should provide mechanisms that help to minimize the amount of resources that are needed to accomplish a certain application task.
- Approach: dynamically adapt the performance of hardware, algorithms, and protocols to the varying needs of the application.
 - Examples:
 - adaptive fidelity algorithms that can be tuned to trade off output fidelity for resource usage.
 - exploit application knowledge to decide when to switch off the radio for energy efficiency reasons.

Network dynamics

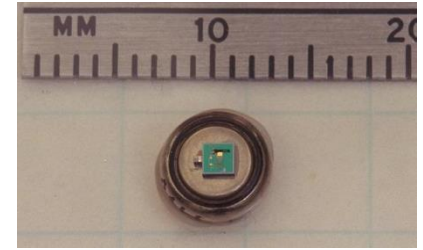
- Ubicom appl. requires the collaboration of spatially distributed devices
 1. devices in an ubicomp appl. tend to be highly specialized, e.g.
 - some sense environmental parameters,
 - some extract information from the collected sensory data,
 - some interact with human users.
 2. many applications require sensory input from many spatially distributed devices (e.g. occupancy level of a room).
 3. the constrained resources of individual devices often require collaboration for solving complex tasks

 - limited communication range of ubicomp devices =>
 1. ubicomp networks not ~ mobile phone networks, where devices communicate directly with a base station
 2. ubicomp devices form **ad hoc networks**:
 - devices act as routers, forwarding messages for their neighbors over multiple hops.
 - powerful devices might act as gateways that connect ad hoc network patches of ubicomp devices to an existing background infrastructure
 - the topology is subject to frequent changes due to device mobility, environmental obstructions resulting in
 - communication failures (e.g., a truck driving by),
 - hardware failures (e.g., depleted batteries, stepping on a device).
-

Handle with the network dynamicity

- *information hoarding*: downloads data during online phases that might be needed later on
- *data-centric communication*: distributed components are identified solely based on the function or data they provide
 - e.g., “some device in my vicinity that can measure temperature”.
 - advantage: tolerate devices going offline by transparently switching over to a device with equivalent functionality
- *lease concept*: where resources allocated for remote peers are associated with a lease, which has to be renewed regularly.
 - If the lease expires due to a missing renewal, the system can automatically reclaim the associated resources.

Scale of Deployment



- Example: Smart Dust – concept introduced in 2001
 - millions of dust-grain-sized devices (motes) would be deployed in the environment in order to monitor various environmental phenomena.
 - A single device consists of sensors, a processor, wireless communication, energy supply.
 - devices are small enough, e.g.
 - stay suspended in air, for example, to monitor weather phenomena or air quality.
 - mixed into paint in order to coat buildings, which would allow monitoring the effects of seismic activity on the structural integrity of the buildings.
 - Impossible to manually configure, maintain, fix, or upgrade individual devices due to their huge number.
 - Impossible to assign unique identifiers (e.g., similar to the unique MAC address) to individual nodes due to the involved production overhead.
 - Totally symmetric situation (all devices are identical initially), the collection of devices *must self-configure* in order to achieve an operational state (e.g., set up a network topology, assign tasks to devices, collaboratively merge and evaluate collected data).
 - The network should be *self-maintaining* in order to fix node failures without manual intervention.
- => Ubicomp middleware should provide support mechanisms for self-configuration and self-maintenance.

Real-world integration

- By definition, ubiquitous computing devices are embedded into the physical environment, typically capturing data about their environment using attached sensors
 - ⇒ a close integration of ubicomp systems with the real world
 - ⇒ physical time and location play a crucial role in ubicomp.
 1. it is often important to know where and when something happened.
 2. time and location are crucial for correlating information from different sources.
 - Ubicomp devices have to share a common understanding of time and location in order to tell whether they were at the same location at the same point in time.
 - common understanding of time i.e. time synchronization
 - common understanding of time location i.e. device localization
 - ⇒ a need for services that manage spatio-temporal data.
 - E.g. a location service maintains an up-to-date view of the current locations of devices in the network.
 - E.g. a history service stores location and time of past events, providing the foundation for queries like “Where did devices X and Y meet last time?”
-

Collection, Processing & Storage of Sensory Data

- core ubicomp functionality
- sensors collect rather low-level data (e.g., time series of temperature readings)
- applications are often interested in highlevel features -- e.g., “in a conference”: used to automatically switch off mobile phones
 - E.g. of functionality: *context service*:
 - derivation of context information requiring the evaluation of sensory data of various types (e.g., noise level, light intensity, air quality) originating from multiple sources..
- Need for energy efficiency and the high energy consumption of wireless communication
 - large amounts of raw sensory data are transmitted to a central location for processing? is often not feasible !!! due to the resulting high energy consumption, bandwidth limitations, and scalability issues.
 - Solution: “*in-network data aggregation*”
 - sensory data are preprocessed as close to its source as possible in order to reduce the amount of data that has to be transmitted.
 - => reduces communication and saves energy by transmitting compact aggregates instead of bulky raw data.
 - => blur the clear separation of communication and data processing typically found in traditional distributed systems and respective middleware

Integration with Background Infrastructures

- Typically: ubicomp devices form infrastructureless ad hoc networks
 - Quite likely: some of the devices will be connected to a background infrastructure such as the Internet.
=> Concept of global “Internet of Things” (e.g. EC programme FP7-ICT) connecting smart artifacts all over the world.
 - Reasons for such an integration:
 - background infrastructures might be used to disseminate information (e.g. room occupancy status) to remote destinations.
 - background infrastructure may provide resources (e.g., computing power, storage) that are not available on typical ubicomp devices.
-

Case Study: Sensor Networks

- subarea of ubicomp: “wireless sensor networks” (WSN).
 - WSN consist of *sensor nodes* — small autonomous computing devices equipped with sensors, wireless communication capabilities, a processor, and a power supply.
 - Example: such a sensor node is the MICA sensing device
 - Counter-examples (UbiComp, not WSN): mobile device networks
 - Appls:
 - Biologists: monitor the behavior of animals in their natural habitats.
 - Environmental research: monitoring environmental pollutions.
 - Agriculture: observing soil quality and other parameters that influence plant growth.
 - Geologists: monitoring seismic activity and its influences on the structural integrity of buildings.
 - Military: monitoring activities in inaccessible areas.
-

Typical usage model of a sensor network

1. A user specifying a high-level sensing task -- e.g., “Report rooms where average noise level exceeds a certain threshold”.
 2. This task is split into many simple subtasks, which are distributed to the individual nodes of the network.
 3. These subtasks collect and preprocess low-level sensor readings.
 4. The resulting sensory data is then aggregated and processed to form a high-level sensing result that is reported back to the user.
- There is a strong need for abstractions that allow easy tasking of the network as a whole.
 - Middleware for sensor networks should support such programming abstractions
-

TinyDB

- A no. approaches treat the sensor network as a distributed database
 - users can issue SQL-like queries to have the network perform a certain sensing task.
 - TinyDB is a representative of this class.
 - Supports a single “virtual” database table sensors:
 - each column corresponds to
 - a specific type of sensor (e.g., temperature, light) or
 - other source of input data (e.g., sensor node identifier, remaining battery power).
 - Reading out the sensors at a node can be regarded as appending a new row to sensors.
 - The query language is a subset of SQL with some extensions.
-

TinyDB example

- Several rooms are equipped with multiple sensor nodes each.
- Each sensor node is equipped with sensors to measure the acoustic volume.
- The table sensors contains
 - room number the sensor is in,
 - floor on which the room is located and
 - volume.
- ? Determine rooms on the 6th floor where the average volume exceeds the threshold 10
- R: Query:

```
SELECT AVG(volume), room // a pair of average volume and the respective room number is returned
FROM sensors
WHERE floor = 6 // selects rows from sensors at the 6th floor
GROUP BY room // selected rows are grouped by the room number
HAVING AVG(volume) > 10 // the average volume of each of the resulting groups is calculated
// only groups with an average volume above 10 are kept

EPOCH DURATION 30s // query is re-executed every 30 seconds
```


TinyDB approach

- Uses a decentralized approach:
 - each sensor node has its own query processor that preprocesses and aggregates sensor data on its way from the sensor node to the user.
 - Executing a query involves the following steps:
 1. A spanning tree of the network rooted at the user device is constructed and maintained as the network topology changes, using a controlled flooding approach
 - flood messages are also used to roughly synchronize time among the nodes of the network.
 2. A query is broadcast to all the nodes in the network by sending it along the tree from the root toward the leaves
 - a time schedule is established, such that a parent and its children agree on a time interval when the parent will listen for data from its children
 3. At the beginning of every epoch, the leaf nodes obtain a new table row by reading out their local sensors.
 4. Leaf apply the select criteria to this row.
 5. If the criteria are fulfilled, a partial state record is created that contains all the necessary data (i.e., room number, floor number, average volume in the example).
 6. The partial state record is then sent to the parent during the scheduled time interval.
 7. The parent listens for any partial state records from its children during the scheduled interval.
 8. The parent proceeds like the children by reading out its sensors, applying select criteria, and generating a partial state record if need be.
 9. The parent aggregates its partial state record and the records received from its children (i.e., calculates the average volume in the example), resulting in a new partial state record.
 10. The new partial state record is then sent to the parent's parent during the scheduled interval.
 11. This process iterates up to the root of the tree.
 12. At the root, the final partial state record is evaluated to obtain the query result.
 13. The whole procedure repeats every epoch.
-

SensorWare

- Class of middleware approaches inspired by mobile code and mobile agents:
 - the sensor network is tasked by injecting in it a program that
 - can collect local sensor data,
 - can statefully migrate or copy itself to other nodes,
 - can communicate with such remote copies.
 - SensorWare is a representative of this class
 - programs are specified in Tcl:
 - functionality specific to SensorWare implemented as a set of additional procedures in Tcl interpreter.
 - Query: takes a sensor name (e.g., volume) and a command as parameters.
 - Value: used to obtain a sensor reading.
 - Send: takes a node address and a message as parameters and sends the message to the specified sensor node.
 - Node addresses: a unique node ID, a script name, and additional identifiers to distinguish copies of the same script.
 - Replicate: takes one or more sensor node addresses as parameters and spawns copies of the executing script on the specified remote sensor nodes.
 - Node addresses are either unique node identifiers or “broadcast” (i.e., all nodes in transmission range).
 - Checks whether a remote sensor node is already executing the specified script.
 - The wait command expects a set of event names as parameters and suspends the execution of the script until one of the specified events occurs
 - The occurrence of an asynchronous activity (e.g., reception of a message, expiry of a timer) is represented by a specific event each.
-

DSWare

- another approach to sensor network middleware is based on the notion of events.
 - the application specifies interest in certain state changes of the real world (“basic events”).
 - Upon detecting an event, a sensor node sends an event notification toward interested appls.
 - The application can also specify certain patterns of events (“compound events”), such that the application is only notified if occurred events match this pattern.

 - DSWare is a representative of this class.
 - supports the specification and automated detection of compound events.
 - A compound event specification contains:
 1. an event identifier,
 2. a detection range specifying the geographical area of interest,
 3. a detection duration specifying the time frame of interest,
 4. a set of sensor nodes interested in this compound event,
 5. a time window W ,
 6. a confidence function f ,
 7. a minimum confidence c_{\min} , and
 8. a set of basic events E .
 - The confidence function f maps E to a scalar value.
 - The compound event is detected and delivered to the interested sensor nodes, if $f(E) \geq c_{\min}$ and all basic events occurred within time window W .
-

DSWare example

- Detecting an explosion event – requires:
 - the occurrence of a light event (i.e., a light flash),
 - a temperature event (i.e., high ambient temperature), and
 - a sound event (i.e., a bang sound) within a subsecond time window W .
- The confidence function is defined:
 $f=0.6 \cdot B(\text{temp})+0.3 \cdot B(\text{light})+0.3 \cdot B(\text{sound})$
(the detection of the temperature event gives us higher confidence in an actual explosion happening than the detection of the light and sound events)
- The function B maps an event ID to
 - 1 if the respective event has been detected within the time window W , and
 - 0 otherwise.
- With $c_{\min}=0.9$, f would trigger the explosion event if the temperature event is detected along with one or both of the light and sound events.