Towards Distributed Awareness - An Artifact based Approach

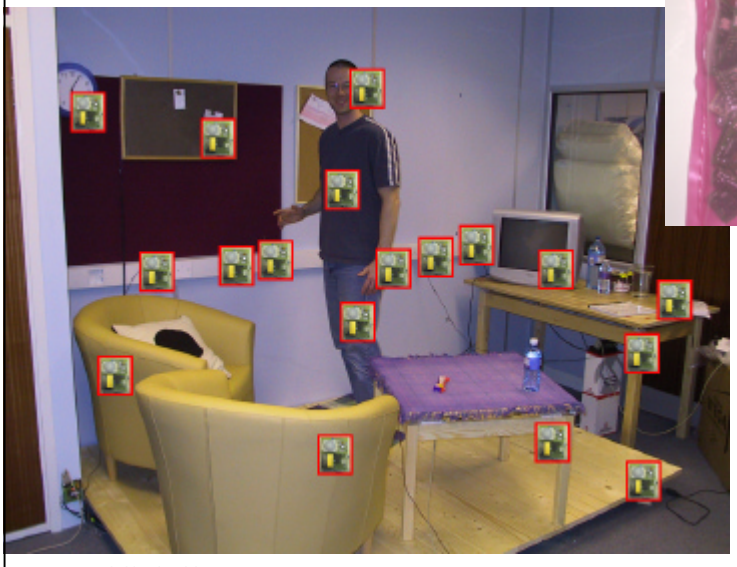
WMCSA December 2, 2004

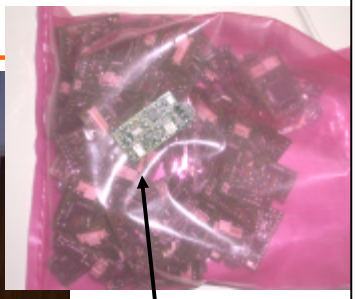
Florian Michahelles, Stavros Antifakos, Albrecht Schmidt, Bernt Schiele, Michael Beigl

ETH Zurich, Switzerland, University of Munich, Germany

TU Darmstadt, Germany, University of Karlsruhe, Germany

Our Vision of a Sensor Enriched Ubicomp Environment





Bunch of Devices

Distributed Awareness Approach

Context perception in a distributed sensing systems

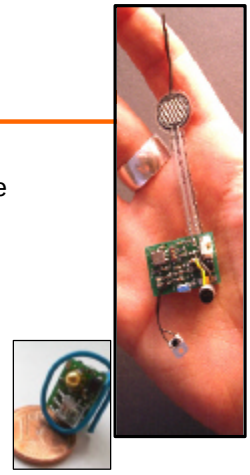
- Some requirements:
  - run several applications on top of these devices
  - reuse implemented parts for new applications
  - change one applications without interfering with another
  - improvement/changes of the infrastructure should not break applications
  - possible to implement on very small device
  - works without backend infrastructure (p2p)
- Based on experience gained in the Smart-Its project
  - Many demos and applications build successfully using this approach
  - Having changing environments using this approach

Artifact centric approach

Platform Smart-Its

Idea

- Device as secondary artefact, integrated into the object
- Independently operating, local deciding with peers
- Integrates Computation, Sensing & Communication
- Post-hoc attachable/embeddable or integratable
- Core and add-on boards



Smart-Its – A Ubiquitous Computing Platform

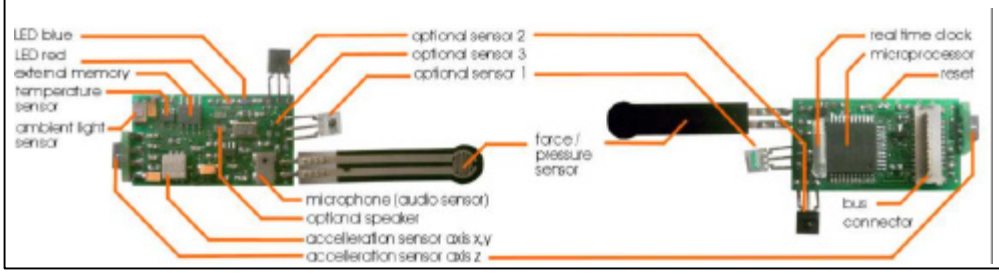
Means for exploring applications and new forms of physical interaction

Building scenarios

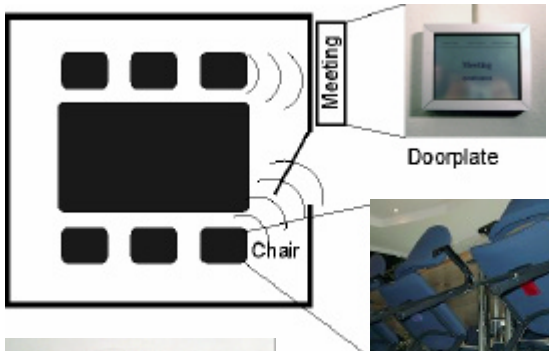
- Rapid-prototyping of interactive applications
- explore interaction with the Ubiquitous Computer

Characteristics

- Some technical parameters: up to 5 MIPS, 128kbyte program, 4k+512kbyte RAM, battery operated, various RF
- Down to 1cm³, lifetime up to several years, simple to program, simple to build/extend

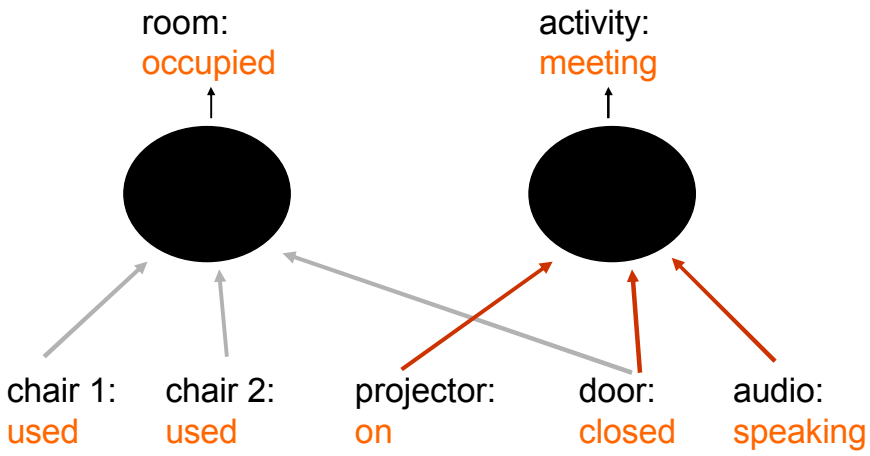


Office – Example Implementation



- **Aware artifacts**
  - Chairs
  - Pens
  - Signs
  - Coffee cups
  - ...

Context Correlation / Time and Space Example



Bottom-up Context Models

Context is Anchored in Artifacts
- Modeling and acquiring context on entity level
- More general properties
- Flexible, extensible, and simple model
- Exploiting domain knowledge

Augmenting Artifacts with
- Sensing
- (Actuation)
- Processing
- Communication

Context Related to Interaction with the Artifact
- Combining context on a higher level
- Time & space correlation

Bottom-up Context - Example

sofa
- free
- occupied with one person
- occupied with two people
- occupied with three people

door
- open
- closed
- degree of openness
- interaction

briefcase
- empty
- loaded
- open
- closed
- interaction

sofa (over the top)
- ...
- jumping on the sofa
- motion of people on the sofa
- temperature on the sofa
- pouring orange juice on the sofa
- pouring wine on the sofa
- pouring milk on the sofa
- cleaning the sofa
- moving the sofa
- sofa placed on the stairs
- sofa upright
- upside down
- sofa flying in midair
- ...

Artifact-based Perception Model

perception based on simple sensors

sensor reading are meaningful
- when related to a real world object that are attached to
- when related to other objects which are aware

implementing sensing and context recognition for a specific object is
- simpler than for a complete system
- more generic and applicable to several applications
- allows reuse of perceptual components

Context-aware Ubicomp systems can be modeled as
- set of networked context-aware artifacts
- time and space relation between these artifacts

A layered architecture for distributed context-aware systems

Artifact layer
- data collection,
- perception and recognition for the particular artifact

Setting layer
- tightly coupled group of artifacts
- all perception and recognition tasks in a group

Application layer
- application-specific perception and recognition
- context information relevant for the application is combined

Artifact Layer

- **Modeling a single artifact**
- **Usually a single sensing / perception node**

context primitives are determined by asking
- what is the artifact and what is its prime use?
- who are the users of such an artifact and in which situation do they use it?

tasks that are accomplished in the artifact layer
- Sensor data acquisition
- Artifact centric perception processing
- History and long term buffers

Artifact Layer API

Description	Function	Type
scan local artifact for sensors available	`scan_i2c() → [<sensor_1>, <sensor_2>, ..., <sensor_n>]`	Discovery
asking for the capabilities/features supported/provided by a local sensor	`cap_i2c(<sensor_j>) → [<feature _1>, <feature_2>, ..., <feature_n>]`	
to prepare feature calculation since get_i2c is blocking	`prepare(<sensor_j:feature_i>)`	Single request
poll sensors/features for values for most recent value	`get_i2c(<sensor_j:feature_i>) → value`	
specify condition: when sensor sends interrupt over I2C	`on_change(<sensor_j:feature_i>, condition )`	Condition trigger
create a buffer to collect sensor data	`create_buffer(<sensor_j:feature_i>, desired length, interval-ms, timewindow-ms, func-id)`	Subscription
access a buffer previously created	`get_buffer(<sensor_j:feature_i> )`	

Setting Layer

- tightly grouped set of artifacts or devices that are cooperating.
- cooperation between artifacts for the purpose of supporting a particular setting
- independent of a particular application

Questions to establish a setting

- What is the relationship among artifacts?
- What is the purpose of the setting?
- Who are the users?
- What perception primitives/contexts are provided?

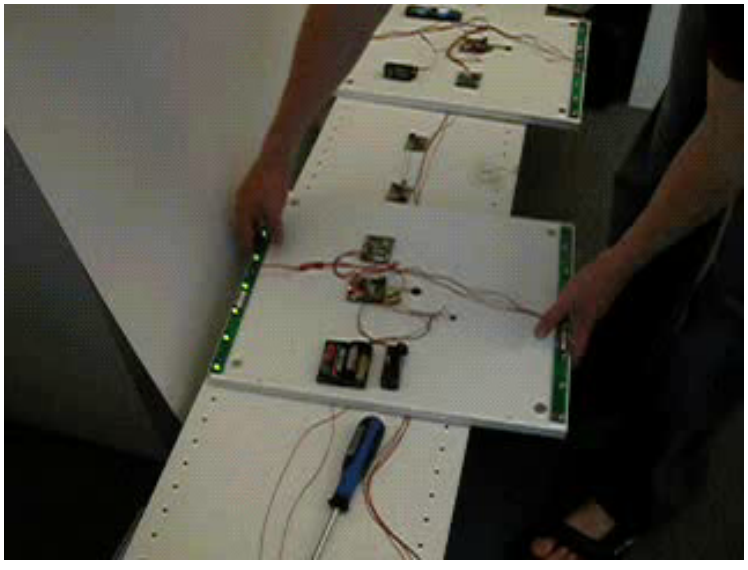
Tasks in the setting layer

- offering collective perception primitives
- collecting and providing setting history

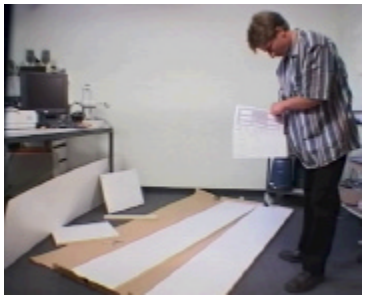
Setting Layer API

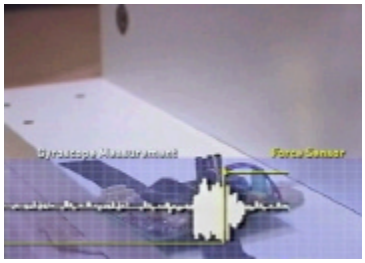
Description	Function	Type
enquire for all devices in a certain physical distance	`hello(distance) → [<id_1>, <id_2>,…, <id_n>]`	Discovery
scan remote artifact for sensors available	`scan_id(<id>) → [<s_1>, <s_2>,…, <s_n>]`	
asking for the capabilities/features supported/provided by a remote sensor	`cap_sensor_id(<id:sensor >) → [<f_1>, < f_2>,…, < f_n >]`	
to prepare feature calculation on remote artifact	`prepare_rf(<id:sensor_j:feature_i>)`	Single request
poll sensors/features for values for most recent value from remote artifact	`get_rf(<id:sensor_j:feature_i>) → value`	
specifiy condition: when remote sensor notifies on condition	`on_remote_change(<id_k:sensor_j:feature_i>, condition )`	Condition trigger
create a remote buffer to collect sensor data	`create_remote_buffer( <id_k:sensor_j:feature_i>, desired length, interval-ms, timewindow-ms, func-id)`	Subscription
access a remote buffer previously created	`get_remote_buffer( <id_k:sensor_j:feature_i>, start_time, length )`	

Case Study: Proactive Instructions

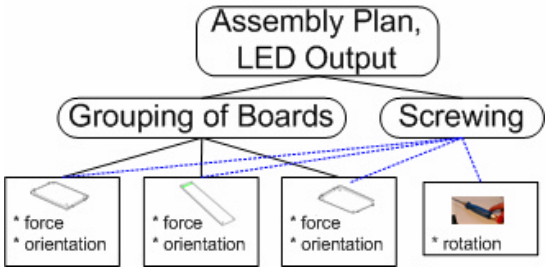


Case Study: Proactive Instructions





- **Boards and screw driver are the artifacts**
- **Settings layer infers the current assembly and activity**
- **Application layer**
  - displays embedded instructions
  - Compare steps to the plan



Case Studies / Evaluation

Several systems implemented / re-implemented

- proactive instructions
- A-life (avalanche rescue system)
- Demo of a restaurant with aware items
- Smart office
- Smart-Its Friends
- Load-sensing environment

Lessons learned

- **Implementing on artifact level allows separation of concerns**
- **Reuse of existing parts in a system**
- **Successful hiding of low-level functionality**
- **Further functionality and new applications are much quicker implemented**

Conclusions

Artifact centered view easies development of context-aware applications

Layered approach
- allows separation of concerns
- enables reuse of perceptual components
- eases application development

Implementation
- Smart-Its and PC
- similar API, C on the MCU, JAVA on PC

Approach successfully applied in several systems

Questions?

Smart-Its Consortium

- ETH Zurich, Switzerland,
- Lancaster University, UK
- TecO, University of Karlsruhe, Germany
- FAL, Victoria Institute, Sweden
- VTT, Oulu, Finland

Funded by the European Disappearing Computer Initiative