# Towards Distributed Awareness - An Artifact based Approach

Florian Michahelles, Stavros Antifakos
Perceptual Computing and Computer Vision,
ETH Zurich, Switzerland
{michahelles, antifakos}@inf.ethz.ch

Albrecht Schmidt
Embedded Interaction Group,
University of Munich, Germany
Albrecht.Schmidt@acm.org

Bernt Schiele
Multimodal Interactive Systems
TU Darmstadt, Germany
schiele@informatik.tu-darmstadt.de

Michael Beigl
Telecooperation Office (TecO)
University of Karlsruhe, Germany
michael@teco.edu

## Abstract

*Perception is a central issue for many applications in ubiquitous computing. However, in current implementations, sensing and distributed perception is re-invented over and over again. Identifying commonalities and synthesizing a common model would be more effective both from an economic as well as scientific perspective. We have analyzed the properties of ubicomp perception systems and used their characteristics as a main input for the design of a perception model. We present a layered perception model, discriminating functionality on artifact, setting, and application level. On the artifact layer, data collection, perception, and recognition for the particular artifact is modelled and implemented. The setting layer deals with perception and recognition tasks for a tightly coupled group of artifacts. In the application layer, perception and application-specific recognition resides. The approach was evaluated by building applications, two of which are also reported in the paper. The wider applicability of the model was assessed by analyzing further applications and how they map onto the artifact-based approach.*

## 1. Introduction

Use of context and context acquisition is a central theme for research and applications in pervasive and ubiquitous computing [6]. In order to provide development support or architectural frameworks various issues in domains such as sensing, perception, ad-hoc networking, and data representation have to be tackled. Our assessment of ubiquitous computing applications shows that device centric sensing as well as distributed sensing is important for a wide range of applications. Most often however, the information of interest, at least from an application's point of view, is on a relatively high level of abstraction and consequently the perception tasks are directly related to the actual application.

When looking at current implementations and when interviewing application developers it becomes evident that sensing and distributed perception is re-invented and re-implemented over and over again in various systems. Often the argument is that the application has very specific requirements (e.g. power consumption, response time) and that therefore a particular, often monolithic architecture and specific implementation is the best solution. While understandable from an individual's point of view this is clearly an undesirable status quo. In particular, as we move from mere demonstrators within controlled research labs to larger scale applications in more open environments there is a stringent need for architectural frameworks including layers of abstraction, modularity, and reusability. While such a overall architectural framework obviously is a long-term research goal, this paper proposes a model for perception as part of such an architecture.

This model focuses on distributed perception systems for pervasive and ubiquitous computing scenarios. It is our particular interest that these abstractions work for different types of applications and are not tailored to a specific scenario. Accordingly, we set out to discover meaningful layers of abstraction with regard to embedded and also mobile devices, concerning distributed, collective, and cooperative perception systems. When looking at ubicomp scenarios it becomes apparent that sensing and context extraction in ubicomp applications is generally related to entities [2, 19]. To create a common artifact-based model we extracted commonalities of a range of different distributed perception systems in ubiquitous computing applications and application classes. Based on these findings, we consider sensing and context information to be related to an artifact or more generally to an entity. Furthermore, artifacts do not only occur in isolation but also in settings where relationships can be established among artifacts. Key issues reflected in the model are networking and mobility and their effects on perception.

Our perception model has three layers of abstraction, namely artifact layer which is directly related to entities, setting layer and application layer. We implemented this model on the Smart-Its platform [5, 24] and developed various applications building upon it. Also, we assess and discuss the model with respect to other, independently developed applications in order to show the applicability of the model to a larger range of ubicomp applications.

In the next section we position our model with respect to related work. Our layered perception model is described in the section 3. This model has been implemented on the Smart-Its platform offering a perception API, as described in section 4. In section 5 two applications, realized with the artifact based model, are discussed in more detail. Furthermore, we show how scenarios, which have been implemented before the model existed, could also benefit from the artifact model as well. Finally, section 6 concludes with a review of our findings.

## 2. Related Work

Previously, several architectures have been proposed to model the flow of sensor data as it is being transformed to context and passed on to the application. The *Context-Toolkit* presented by Dey et. al. [20] separates the context acquisition process from the delivery and use of context. It borrows ideas from developments in graphical user interfaces by using widgets as a type of drivers for sensors. Widgets encapsulate states representing contextual information. Applications can build upon these widgets either by polling or by registering call-back functions. Beside widgets, context servers collect contextual data about a particular entity and context interpreters are used to transform context between different representations. The context toolkit provides a solid infrastructure for managing context and querying context. However, as the context toolkit uses widgets as sensor abstraction it does not give aid for integrating and querying the specific hardware sensors. The issue of integrating different hardware sensors and perception mechanism on a single device is addressed in the *TEA architecture* [23]. This architecture defines layers focusing on the actual perception process, in particular on sensor fusion. The distribution of components and the inclusion of domain knowledge in the perception process is not addressed.

*Context Fabric* [13] is based on loosely coupled network services. It defines an infrastructure for building context-aware applications based on a data store in which the context resides. The main focus is on modelling, distribution and protection of contextual data. The *Location Stack* [10] is an architecture specifically designed for combining different location sensing techniques. It defines a common vocabulary for a set of multi-sensor location systems and implements probabilistic techniques for fusing the data. The *Event Heap* developed as part of the interactive workspaces project [14] is based on a black-board approach. Different parts of applications can use the Event Heap as a common message board to exchange data between loosely coupled components. The main focus of the project is on robustness and fault tolerance. Applying this approach to smart objects would always require one central black-board component, which is not ideal for distributed scenarios. The *Context Information Service* described in [19] takes an approach well known from object-oriented programming. Artifacts are described as entities with states representing their context. Further artifacts can have relations among each other. Using these relations a graph representation for reasoning can be built up.

The work presented in this paper also follows an artifact-based approach. Artifacts represent real world objects with attached sensors and span the range from low level sensors to high level abstractions. The notion of a setting-layer combines artifacts according to their scenario of use into more complex, distributed and networked perception systems. One central issue is collective perception, where distribution issues directly related to sensing and perception are considered.

Sensor networks have been assessed as means for data collection in mobile and ad-hoc settings in different research projects recently [8]. Applications span a wide range, but are most often related to monitoring particular parameters in the environment [7].

In contrast to simply collecting sensed data from the environment, ubiquitous computing follows the goal of enabling new applications by enhancing the environment with communication and sensing devices. Even-though sensing is inherent in this setting it differs from the tasks faced in a classic sensor network approach. When building applications the meaning of sensor data emerges from the coupling of the sensor with the object it is placed on. E.g. the desired temperature of a coffee cup and that of a ice-cream bowl differ substantially. To address this coupling between sensors and artifacts we present an approach directly tying perception to artifacts.

## 3. An Artifact-based Perception Model

Acquiring information about the real world through sensors is a prerequisite for context-aware applications in ubiquitous computing [3]. Considering the context in which sensing happens, i.e. in respect to the objects, artifacts and/or people sensors are placed on, provides vital information. As objects and artifacts in the real world are not totally isolated, but in context and relation with others, the perception model evaluating the sensor readings should reflect this connectedness: a spoon, fork and knife in close co-location may indicate a meal - similarly the perception model has to evaluate the sensor readings regarding the artifacts the sensors are attached to. Sensors and sensor reading have little meaning by themselves, but connected - physically and logically - to an entity they can provide important information. We believe that ubicomp systems may be well modelled by properties of and relations among artifacts where sensors are attached to.

An artifact-based model allows to specify issues on a single artifact level, but also caters for the fact that artifacts may be a part of a larger networked system (setting). When looking from a more global system's perspective some artifacts may be statically coupled whereas others may dynamically and continuously group and regroup in a more open and ad-hoc fashion.

From our analysis we can state the following observation for distributed sensing systems for Ubiquitous Computing scenarios: (1) Sensing, data capture, and perception are related to entities which hold important information for the processing from sensor data to context information. (2) Perception is dependent on the applications that make use of the data acquired. (3) The specific device to which the sensor is attached as well as the environment in which the device is operated are important when selecting the perception methods and architecture.

From these observations several questions arise:

- How is data captured?
- What types of data are captured and how is data provided? In particular what meta data needs to be attached to readings?
- Where does data reside and where is data stored over longer terms?
- How is data accessed and what are the communication models and paradigms?
- How can such a system be realized and what are the processing requirements?

In the remainder of this section an architecture will be presented that takes these observations into account and provides answers to the questions raised.

## 3.1 A layered Architecture

We base the artifact-based model on a layered architecture for several reasons. The analysis of applications showed that structuring the perception process allows separation of concerns. The separation introduces the need for communication between entities and layers; however, as in most cases artifacts and devices are physically distributed in the real world, this does not introduce additional complexity.

By using layers it becomes possible to exchange components (in hardware or software) in one layer without affecting other components. Especially the independence of perception tasks among different layers allows independent testing and evaluation. By simulation of layers, incomplete systems and implementations can be tested.

The model identifies three layers (see Figure 1):

- **Artifact layer**
  In the artifact layer, data collection, perception, and recognition for the particular artifact, is modelled and implemented.
- **Setting layer**
  In the setting layer, all perception and recognition tasks are located that are concerned with a tightly coupled group of artifacts.
- **Application layer**
  In the application layer, application-specific perception and recognition can be found. In this layer context information that is relevant for the application is brought together and used.

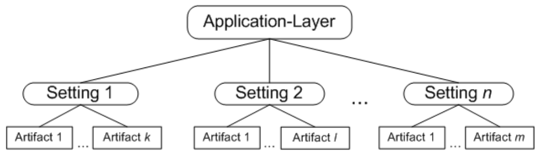

**Figure 1. The layered architecture**

## 3.2 Artifact Layer

In the artifact layer only a single artifact is modelled. Conceptually, we look at the physical artifact rather than at the sensing or computational devices in this layer. In some cases the artifact and the computational device are identical (e.g. a PDA) but in the case of everyday objects the hardware is embedded into the artifact. For modelling the artifact it is central to identify what the meaningful context information is. This is related to the type and usage of the artifact. The following questions help to determine context primitives that need to be supported by the sensing and perception technology included in the artifact:

- 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?

Based on this analysis it is important to identify the technical implications for providing perception and recognition on the level of the artifact. First and foremost is the identification of the required sensors. It is also of particular interest whether the artifact is mobile or embedded in the environment and how energy is supplied. Communication is another central technical issue, in particular what type of communication is supported and with whom communication is feasible. The technology embedded in the artifact with regard to processing, communication, and sensing has to be specified. There are three general tasks that are accomplished in the artifact layer on any one artifact that is modelled.

- **Sensor data acquisition**
  This includes reading data from the actual sensors; and potentially also buffering, time stamping, and controlling the sensor.

- **Artifact centric perception processing**
  Perception processing includes low-level signal processing and basic algorithms to make sensor values more robust. This typically includes the application of filters to sensor data, the integration of data from different redundant sensors, and the fusion of data from different sensors. One aim is to supply more robust sensor and context information related to the particular artifact. The other is to provide higher abstraction of the local sensing. Finally, the artifact structure also allows to represent the real-world structure of the related object.

- **History and long term buffers**
  In various scenarios long term information is of great interest and contains important context information. Keeping the history and abstractions of the data on a particular artifact allows the local calculation of temporal contexts. Accordingly, history of data is maintained in the same nature as the real-world events have happened: this also allows fusion of data across artifacts on the higher setting-layer, if the semantic of different artifacts is known, e.g. meaning of co-location of specific objects.

In general, the output of the artifact layer is abstracted sensor data related to a physical artifact. This can include sub-symbolic or symbolic data representing the context. Additionally there is also access to a history of collected data.

### 3.3 Setting Layer

In the setting layer we look at a tightly grouped set of artifacts or devices that are cooperating. The cooperation between them is for the purpose of supporting a particular *setting*. On a conceptual level a setting presents a local collective perception system that consists of a number of networked artifacts with perception capabilities. The following questions help to establish the facts about the settings:

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

As the perception is distributed and collective it is interesting to establish how the perception primitives within the setting are related. Of special interest is the question if the same perception primitive in different artifacts within one setting is observing the same phenomenon from different angles or if the report is on different observations. This is a non-trivial problem, similar to the registration problem in computer vision.

When looking at artifacts that offer perception and can take part in collective perception systems (settings) at a larger scale, it is important to discriminate between artifacts that are exclusively part of one setting and artifacts that are part of multiple settings. In the latter case conflicting interests between settings can lead to a resource sharing problem (similar to operating systems). This can be resolved by negotiating capabilities between the setting requesting and the artifact offering perception.

On a technical level, how the devices are connected and how communication is realized becomes important. In the straightforward case, a pre-defined group of artifacts (defined at implementation time) is permanently connected. However in different scenarios we also encounter settings where group membership changes in an ad-hoc fashion and communication is only available at certain points in time and space.

The main task performed in the settings layer is to combine information gathered from individual artifacts to meaningful collective context information in the setting. The integration of sensor data across artifacts and in more general data fusion over perception information provided by the artifact-layer leads to setting primitives. These primitives represent the context information of the particular setting and are higher-level abstractions of combined observations. The output of the settings layer is therefore the setting context. Similar to the artifact layer it also provides access to the settings history.

### 3.4 Application Layer

On the application layer loosely and potentially ad-hoc connected and cooperating settings are considered. In this layer the application that uses context is located. The context is collected from settings from the layer below.

For a particular application in the application layer it is of interest which settings are relevant and how the settings are related. The outputs of the settings, abstracted representations of the world modelled in the setting, become the basis for the perception in the application layer. Based on the perception primitives provided by the settings, the application layer performs recognition tasks that are required by an application.

The participation of a particular setting in a number of applications introduces a potential source for conflicts. The resources within a setting, e.g. which primitives are calculated, at which rate perception is performed, and which information is buffered, must then be shared between applications. It can be assumed that applications are cooperative, but for large scale scenarios the conflict resolution or avoidance has to be included in the settings layer.

The communication between the application and the underlying settings is a central issue. The general case is that communication is available in an ad-hoc fashion and that there is no guarantee that all settings are always accessible. In many cases the application can define a minimal core group of setting primitives that are required to ensure useful operation. The application is operational as soon as these settings are available. If more setting primitives become available the context awareness of the application may become better.

The central task in the application layer is to combine perception primitives of settings into meaningful context. This is realized by fusing perception data across settings, which is of particular interest to an application. The calculated context is then related to the domain of the application, e.g. it is tagged with knowledge or related to a plan.

The output of the application-layer is tightly related to each particular application. In general it's output is reflected in the functionality and user interface of the application. Typically there are changes in the presentation of the application to the user. It may be screen based or by use of actuators. The output also includes the application classes, such as context-triggered actions and contextual information as specified by Schilit [22].

The application layer is the least structured one. Its functionality and what is implemented is highly dependent on the actual application. However, providing a defined interface to settings is very helpful when implementing new applications.

## 4. Implementation

This section describes the implementation details of the described model. We start with a short introduction into the underlying hardware including its specific requirements and then present the details of the implementation.

### 4.1 Ubiquitous Computing Hardware

Custom hardware developed during the Smart-Its project [5] was the basis for the design of the artifact-based approach. The hardware was built up in a modular fashion in order to separate concerns as much as possible. It is composed of a sensing and a communication part. Both modules are built upon a low cost micro-controller (Arizona Microchips PIC16F87x). The communication board features a radio frequency transceiver with a digital potentiometer to vary signal strength and runs with a time-slotted protocol. The general sensor board contains a variety of sensors including a temperature sensor, a light sensor, an acceleration sensor, a pressure sensor and a microphone. The sensor selection is similar to what comparable systems use [9, 16]. Additional sensors or specific sensor boards can easily be attached using either $I^2C$ or serial communication. The standard way of building self-contained units is by connecting the RF-module with sensor modules via the $I^2C$ communication protocol. When the sensor boards are used with a PC or PDA hardware serial communication is the default. Combined, they form a basic building block for ubiquitous computing applications. During the project, 160 of these devices were built and distributed to the various project partners.

Besides the custom-built hardware, PDAs, PC's, and notebook computers were used, either as additional mobile units or as background infrastructure.

### 4.2 Software Implementation: Perception API

Implementing the proposed perception model on the hardware described above posed several interesting challenges. The small memory capacity and low computing power of the micro-controller, had a large influence on the design of an API implementing the artifact-based model. Designing a perception API that fits on the devices and at the same time provides a useful set of functions was a major issue. Also offering the API over different communication channels ($I^2C$, serial, and RF) put additional constraints onto the implementation.

There are several issues that have to be solved across different layers. First of all, knowledge about available sensing resources has to be managed. The system needs the ability to discover sensors that are currently available in the network. It requires that sensors can report about their abilities and are addressable. Secondly, different types of temporal sensor access are necessary, depending on the requirements of the actual applications. For example, acceleration values often require frequent sampling whereas for a temperature monitoring system it is sufficient to be notified only when the temperature changes by a significant amount. Thirdly, different filtering and feature calculation functions have to be provided on the devices to limit extensive bandwidth use.

The following three subsections describe the implementation in more detail on each of the three layers.

### 4.2.1 Artifact Layer

The artifact layer deals with properties related to one single artifact. Table 1 gives an overview over the functionality offered in this layer: textual description of each function, exact syntax and classification of the functionality.

The `scan_i2c` and `cap_i2c` functions provide a two-step discovery of available sensors connected to the $I^2C$ bus. This two-step approach was chosen to cope with the memory constraints of the micro-controller platform. First, the scan routine returns all the identifiers of the locally available sensors. Then, in a second step using the capabilities function, only applicable features (sensor capabilities) for the sensor given in the parameter are discovered. This saves memory in contrast to a one-step approach where capabilities returned for all sensors available would consume a lot of memory. Typically this functionality is used just after start up. It enables sensors to be added dynamically to a given artifact. To read sensor values from a given sensor three retrieval types are provided. Single request enables a device to receive only one single sensor value, or alternatively to poll a series of values. For performance reasons a `prepare` operation has been introduced: it allows to initiate the feature calculation prior to the actual `get_i2c` call, which blocks the microprocessor until the $I^2C$ bus is ready.

| 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> )` | |

**Table 1. Artifact-layer Perception API**

The condition triggered notification method, `on_change`, offers atomic feature value conditions ($<, >, =$) that specify when a particular sensor should send an interrupt to the calling device, e.g. a notification if the temperature exceeds 40C. Due to memory constraints all conditions are treated as one-time triggers. Sequences of conditions are interpreted as being logically OR connected. The continuous subscription method, offers continuous feature value series on a timely basis. It is initiated by a `create_buffer` call that allocates memory for a round-robin buffer and starts the feature value calculation on the buffer. The values are determined by a fixed time interval, fixed time window and an identifier of a calculation function (e.g. median, average, sum) to be performed on the feature values. Further, `get_buffer` returns all of the data collected so far in the buffer.

The artifact-layer is completely implemented on the Smart-Its hardware using C [1]. For local use, when the sensor is directly connected to a serial line of a host system, there is also a JAVA wrapper, implementing the same interface.

### 4.2.2 Setting Layer

The setting layer views collections of artifacts and has to establish connectivity among distributed artifacts. Table 2 summarizes the API for the setting layer.

| 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 )` | |

**Table 2. Setting-layer Perception**

The discovery mechanism on this layer requires a distance measure that selects the range of sensor discovery: 0 - represents all local sensors on an artifact itself, 255 - all boards in connectivity range. All values in-between represent a distance measure based on the number of multiple communication hops required to reach a certain sensor. Based on this distance measure, each artifact can either query its neighbors on their primitives individually gained on the artifact layer, or it can remotely access sensors attached to other artifacts. By tuning the signal strength of the radio unit, the distance measure can be directly mapped to physical distances. The retrieval methods are built up the same way as on the artifact layer (see subsection 4.2.1). The only difference is that the universal identifier of the remote artifact to be addressed has to be included as a parameter. The main focus of the setting layer is to combine sensor information across artifacts. To this purpose two different integration paradigms have been implemented:

- *Grouping* of artifacts describes how similar sensor readings from different artifacts are. The correlation coefficient of sensor pairs indicates the degree of dependence. This has proven useful to define semantic proximity of devices [21].

- *Rating*, allows the alignment of multiple measurements on a one-dimensional scale for comparison. This is implemented by a decision tree structure that uses thresholds of various sensors as decision nodes.

Artifacts that execute sensor integration by one of the principles mentioned above are regarded as setting leaders. They also act as gateways and communicate the gathered results to the application layer. These functions are implemented on the Smart-Its devices utilizing a basic message oriented communication. They are provided as a C include file for the CSS compiler. There is also a JAVA equivalent, for cases where the functions are used on more powerful systems and in the back-end.

#### 4.2.3 Application Layer

The application layer is the highest layer of the proposed model and accommodates the main functionality of the application. It takes results from artifact and setting layer and combines them to information directly useful to the application. In many applications this means that the sensing information needs to be fed into some kind of context model or a representation of the task at hand. Finally, the control of the user interface is another eminent part of the application-layer. The application layer can be implemented on a particular Smart-It, which is communicating with other artifacts, or on standard hardware such as laptops or handheld computers, depending on the characteristics of the application at hand. It communicates wirelessly with the custom hardware of the enabled artifacts. In the next Section we present several applications that were implemented using the artifact-based approach.

## 5. Case Studies/Evaluation

In this section we demonstrate the applicability and efficiency of the perception model by means of various applications that have been implemented so far. The artifact-based model fostered the development of the applications by its different layers.

### 5.1 Proactive Instructions

This application, aims to overcome limitations of today's printed instructions [4, 17]. By means of a piece of flat-pack furniture it is demonstrated how instructions can be integrated directly into the objects: sensors attached to the unassembled parts and pieces can recognize the user's actions during the assembly and send the data to a separate computer (see Figure 2). This computer holds the assembly plan, which contains all possible states of assembly similar to a finite state machine. In the first state all items are unassembled. Performed assembly actions trigger state transitions, which either end in the final built up state or fail in wrong states before (see Figure 2). The underlying principle is that the system can track the user's assembly actions and give recommendations based on the knowledge stored in the plan. Instructions are given by low cost LED-strips (Light Emitting Diodes) integrated directly into the parts of the furniture [17]. Each board has two LED-strips that can blink in green or red in various patterns. With these displays little chunks of information can be given at the right time and place. Boards can display assembly points by flashing in green, or the wrong assembly orientation by flashing red. To develop the application we went through a process of examining and classifying the different perceptual tasks that were to be solved.

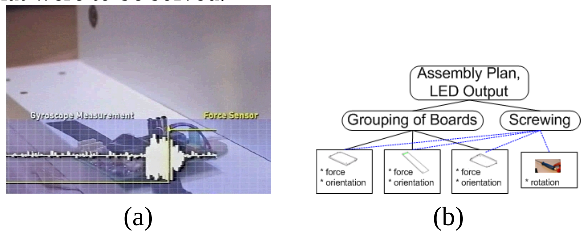

**Figure 2. (a) Smart-Its sensor measuring the user's actions, (b) Perception model**

#### 5.1.1 Artifact Layer

We chose the actual boards of the furniture as the artifacts in the perception model. Besides the boards we included a screwdriver on the artifact layer. The screwdriver is enhanced with a gyroscope to detect when someone is tightening a screw. Here, the artifact based approach proved highly valuable and appropriate, as these objects all have their own typical context and are the smallest computing entities in the application. The low level perception methods in the artifacts determine the orientation of the boards (accelerometer), if they are moving or not, whether they are being hammered on, and if they are experiencing force (force sensors at edges) from another board. The screwdriver simply detects (gyroscope), whether someone is loosening or tightening a screw. These perception primitives define the output of the artifact layer. Equally, they are the input to the setting layer. All perception primitives are calculated locally on each artifact.

#### 5.1.2 Setting Layer

In this layer the perception primitives from a tightly coupled group of artifacts are combined. In the proactive furniture example the boards and tools involved in the momentary stage of the assembly form this group of artifacts. This may create a large number of overlapping settings in the overall assembly plan. However, the actual settings of interest are determined by the current state of the assembly. The setting layer infers the performed assembly steps by combining the perception primitives from the artifact layer. In detail, it determines whether two boards are aligned and if they have been joined or not. Furthermore, it can detect on which board, the screws are tightened. The setting-layer detects specified settings of the boards and reports those to application layer where the entire assembly procedure is tracked by means of a plan representation.

### 5.1.3 Application Layer

The application layer is where the logic of an application resides. The core logic of the proactive furniture example is a plan representing all possible assembly routes. The information about the actions recognized is passed up from the setting layer and fed directly into the assembly plan. In this way the momentary state of the assembly can be determined. Besides tracking the state of the assembly the main task of the application layer is controlling the feedback to the user. In previous work [4] we defined different usage modes depending on the level of expertise of the user. Different instructions are provided depending on whether the user is a beginner, intermediate or an expert. In our prototype instructions can be given either using the LED-strips on the boards, visual instructions on a computer screen or auditory instructions in form of spoken words.

### 5.1.4 Contributions of the artifacts-based approach

The artifact-based approach guided us through the general system design. We started attaching the relevant sensors to the furniture boards (artifacts). In a second step we could chose the appropriate perception primitives for combining sensor readings on the setting layer. Both the artifact- and the setting-layer could reuse the the implementation of the perception API as described in section 4.2, which had validated in previous projects.

## 5.2 A-Life

A-Life aims at improving non-professional avalanche rescue among winter enthusiasts by wearable sensors [18]. As there is an on-going trend towards out-of-bound (or off-piste) skiing, danger of avalanches is often underestimated. There are many incidents of triggered slides causing accidents and even deaths every year. Furthermore, statistics show that in many cases there are multiple victims, and the order in which they are saved can be critical - some victims may survive for hours, whereas others need to be rescued within minutes. If rescuers could find means to address the most urgent cases first, many additional lives would probably be saved. Today, commercial beacon systems tracking systems penetrate the snow with long-wave signals. The signal flux lines of the victims' emitting devices offer the rescue team information on the location of a single victim at a time. A-Life aims at extending these beacon systems by providing rescuers with additional vital sign data reported from sensors worn by mountaineers. At this stage, a prototypical implementation of such a system

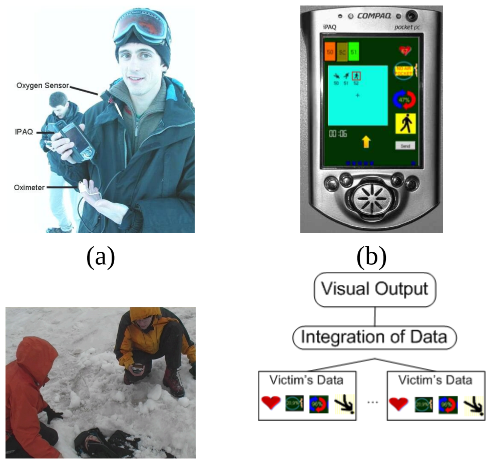

**Figure 3. (a) User with the A-life system, (b) A-Life user-interface, (c) Fieldtest, (d) Perception model**

has been used for participatory evaluations with practitioners in the field (see Figure 3). Furthermore, we discussed the appropriateness and benefits of such a system in several focus groups with emergency physicists, professional rescuers, researchers from the Swiss Federal Institute for Avalanche and Snow[1], mountaineers and a technical consultant. A new prototype incorporating these results is currently on the way.

### 5.2.1 Artifact Layer

Mountaineers are instrumented with an acceleration sensor for measuring orientation of burial, an oximeter revealing heart rate and oxygen blood saturation, and an oxygen sensor reporting on closed air-pockets in front of the victim's mouth. On this level the artifact-layer encapsulates the local sensing efforts at the body of a single mountaineer. In standard operation each device, including sensors and beacon unit, evaluates the data from its connected sensors and provides output using a broadcast command, such that in emergency cases those signals could be received.

### 5.2.2 Setting Layer

In the case of an avalanche, surviving team members set their devices into a different mode and act as rescuers. Each rescuer's device collects the broadcasted victim data from the artifact layer of all victims. The setting-layer on each rescuer device allows to combine the received information into a more integrated view:

A decision tree is used for transforming the victim's data into a representative urgency measure. Taking heart rate as a primary criterion, evidence of air-pockets as a secondary,

[1]http:///www.slf.ch

oxygen blood saturation as a third and victim orientation as fourth criterion, victims can be classified in respect to urgency of aid. For reliability reasons the urgency measure is calculated on each rescuer's device separately.

#### 5.2.3 Application Layer

On the application layer the visualization of the information gathered in the settings layer is implemented. A screen display shows a visual map representing the spatial distribution of victims to enable the rescuer to select the victims such that ways can be kept short. Alternatively, the visualization can separate victims by urgency and provides rescuers with a global view on the emergency.

#### 5.2.4 Contributions of the artifacts-based approach

Similar as in the furniture application the artifact-based aided in the general system design. The artifact-based approach allowed us to decompose the complex problem of avalanche rescue into single portions: the artifact-layer allowed us to model the sensing efforts for a single mountaineer in isolation from others, whereas the combination of readings for a group of mountaineers could be modelled and solved later during the implementation of the setting-layer. The application-layer enabled us to test and develop the visualization on hand-held device without sensors first, and to integrate it with the sensor system later. In the entire development process each layer could be modelled and tested separately. This allowed a much more focussed step-by-step process than solving all problems at once.

### 5.3 Other Applications

The Smart-Its-Friends system was developed (before the perception model existed) as a demonstrator for a new interaction technique with objects [11]. Based upon the common movement of two objects a connection between them is established: they become ”friends”. In the implementation one artifact continuously broadcasts its data, which is compared with the most recent movement pattern of the second artifact. If the difference is below a specified threshold then friendship is established and the devices start beeping. If the devices get separated over certain distances, the friendship dissolves. This shows, that relation among artifacts can evolve, remain and disappear. From the perception model point-of-view, movement sensing can be placed on the artifact layer. The comparison with the movement pattern of the second artifact can be interpreted on the setting layer, also the representation of multiple partnerships among a set of devices could be feasible. Establishing the friendship is located in the application layer.

The restaurant application demonstrates a scenario where menus and drinks are dynamically priced depending on how the ingredients have been handled by the personnel (e.g. whether vegetables have been properly stored in the fridge, or wine bottles have been shaken). It investigates the impact ubiquitous computing technology may have to future environments [12]. On the artifact layer, movements of goods (e.g. a wine bottle) are sensed. Correlation between goods is implemented in the setting layer. The application-layer triggers dynamic pricing and changing menus based on the information from the setting-layer.

Location aware applications could also make use of the artifact-based approach. In the mentioned examples, applications use the information about geographic or co-location to either display the information, to automatically trigger actions or to use that information as additional input to a system. Applications like those developed in the Active Badge [25] are simply rebuilt using the artifact-based approach presented in this paper. Each active badge device would implement a continuous broadcast of its ID on the artifact-layer. Using the `hello` function of the Perception API's setting layer from fixed points in a space would collect the ID's of the surrounding devices and calculate the location accordingly. On the application-layer the location information would be used, i.e. displayed or exploited to forward telephone calls.

Another interesting application area is query processing using networks of sensor enabled objects as used in the Cougar system [26] or TinyDB [15]. These applications use a network of sensor equipped nodes as a distributed database. Queries to these objects in these database-like systems are similar to relational database languages, e.g. SQL queries. With the help of the Perception API an implementation of this database functionality could apply the `hello`, `prepare_rf` and `get_rf` functions. For example a query as

```
  select light from node_1 where
light==artificial_light
```

will be translated into successive calls of

```
  prepare_rf(sensor_1,light,
artificial_light)
```

and

```
  get_rf(sensor_1,light,artificial_light)
```

As the example shows, the Perception API based queries go beyond database queries of Cougar and TinyDB where sensor conditions have to be expressed in a sensor dependent way - e.g. `light>10`. Instead, the Perception API allows to phrase more meaningful queries: `light==artificial_light` allows the formulation of sensor hardware independent queries by using the processing and recognition capabilities of the nodes. This achieves a better separation of concerns between the querying computer that is responsible for data processing and the networked sensing nodes that are responsible for recognition and processing of physical world information.

## 6. Conclusion

In this paper we have presented a model for distributed perception systems for ubiquitous computing applications. Using a layered architecture the perception process is structured, with relation to the artifact, setting, and application. In the artifact layer, functionality related to single devices is situated. In the setting layer perception that is dependent on different artifacts but is independent of a specific application is implemented. In this context, a setting is defined by a set of artifacts which perform collective perception. The application layer takes its input from the settings layers and is where a specific application resides.

The architecture is implemented on the Smart-Its platform. On each layer dedicated functions are provided by an API to ease the development of context aware applications. The implementation includes functions on the microcontroller as well as wrapper classes in JAVA on the backend system.

Two largely different applications, Proactive Furniture Assembly and A-Life, both implemented with the perception model are presented in more detail. They illustrate that the model can help to easily structure and implement even complex applications, with different artifacts and settings. The diverse nature of these applications and the analysis of the proposed model with respect to previously published work show the applicability of the perception architecture to ubiquitous computing applications in general. The findings encourage the chosen approach.

Our findings support that the perception model can aid during the development of many different kinds of sensing based ubiquitous computing applications. By providing simple but powerful sensor abstractions and concepts for access we achieve an accelerating effect on application development with the Smart-its platform.

## References

[1] CCS Compiler for PIC micrcontroller. `http://www.ccsinfo.com`.

[2] GD. Abowd, AK. Dey, PJ. Brown, N. Davies, M. Smith, and P. Steggles. Towards a Better Understanding of Context and Contex-Awareness. In *HUC*, pages 304–307, 1999.

[3] GD. Abowd and ED. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, 2000.

[4] S. Antifakos, F. Michahelles, and B. Schiele. Proactive Instructions for Furniture Assembly. In *UbiComp*, Gothenburg, Sweden, 2002.

[5] M. Beigl, T. Zimmer, A. Krohn, C. Decker, and P. Robinson. Smart-its - communication and sensing technology for ubicomp environments. Technical Report ISSN 1432-7864 2003/2, TecO Karlsruhe, 2003.

[6] AK. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1), 2001.

[7] D. Estrin, D. Culler, K. Pister, and G. Sukhatme. Connecting the Physical World with Pervasive Networks. *IEEE Pervasive Computing*, 1(1):59–69, 2002.

[8] D. Estrin, L. Girod, G. Pottie, and M. Srivastava. Instrumenting the world with wireless sensor networks. In *ICASSP 2001*, Salt Lake City, Utah, May 2001.

[9] J. Farringdon, AJ. Moore, N. Tilbury, J. Church, and PD. Biemond. Wearable sensor badge and sensor jacket for context awareness. In *ISWC*, 1999.

[10] J. Hightower, B. Brumitt, and G. Borriello. The location stack: A layered model for location in ubiquitous computing. In *WMCSA 2002*, Callicoon, NY, 2002.

[11] LE. Holmquist, F. Mattern, B. Schiele, and HW. Gellersen. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In *Ubicomp*, Atlanta, USA, October 2001.

[12] LE. Holmquist, R. Maz, and S. Ljungblad. Designing Tomorrow's Smart Products - Experience with the Smart-Its Platform. In *DUX*, San Francisco, 2003.

[13] JI. Hong and JA. Landay. An Infrastructure Approach to Context-Aware Computing. *Human-Computer Interaction*, 16, 2001.

[14] B. Johanson and A. Fox. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. In *WMCSA 2002*, Callicoon, NY, 2002.

[15] S. Madden, R. Szewczyk, M. Franklin, and D. Culler. Supporting aggregate queries over ad-hoc wireless sensor networks. In *WMCSA 2002*, Callicoon, NY, 2002.

[16] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. Atlanta GA, 2002.

[17] F. Michahelles, S. Antifakos, J. Boutellier, A. Schmidt, and B. Schiele. Instructions immersed into the real world How your Furniture can teach you. In *Ubicomp 2003*, Seattle, USA, 2003.

[18] F. Michahelles, P. Matter, A. Schmidt, and B. Schiele. Applying Wearable Sensors to Avalanche Rescue: First Experiences with a Novel Avalanche Beacon. *Computers & Graphics*, 27(6):839–847, 2003.

[19] J. Pascoe. Adding Generic Contextual Capabilities to Wearable Computers. In *ISWC*, pages 92–99, 1998.

[20] D. Salber, AK. Dey, and GD. Abowd. The context toolkit: Aiding the development of context-enabled applications. In *CHI*, pages 434–441, 1999.

[21] B. Schiele and S. Antifakos. Beyond Position Awareness. In *Proc. of the Workshop on Location Modeling at Ubicomp*, October 2001.

[22] Bill N. Schilit, N. Adams, and R. Want. Context-Aware Computing Applications. In *WMCSA 1994*, 1994.

[23] A. Schmidt, K. A. Aidoo, A. Takaluoma, U. Tuomela, K. Van Laerhoven, and W. Van de Velde. Advanced interaction in context. In *HUC*, 1999.

[24] Smart-Its. `http://www.smart-its.org`.

[25] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. 10(1):91–102, 1992.

[26] Y. Yao and JE. Gehrke. Query Processing in Sensor Networks. In *CIDR 2003*, Asilomar, USA, 2003.