

Building extendable room based sensor clusters for ubiquitous computing environments

Martin Jonsson, Johan Mattsson

The FUSE Research Group, KTH Center for Wireless Systems, KTH
DSV, Electrum 230, SE-164 40 Kista, Sweden
[martinj, johanm]@dsv.su.se

Abstract. Experiences from work with sensors in ubiquitous computing environments are described and summarized. Problems regarding reusability, scalability and ease of deployment of sensor systems are identified. A platform architecture as well as a reference implementation is presented, that supports development of sensor based applications on a low level. The platform eases reusability, sharing of sensors between applications, supports transport, and refinement of sensor data.

Introduction

Providing information from sensors to applications can be very useful in several ways. The behaviour of an application can be adapted to make the interaction more efficient or to increase the ease of use. You can also imagine entirely new types of applications that are designed specifically to make use of some certain sensor or context information. Under the notion of context aware computing there has been several attempts both to define what context really is, and how to design applications that use it [1-2]. Within the area of ubiquitous computing several systems have been presented which incorporate input from different kinds of sensors [3-5].

The increasing use of context information in applications makes it interesting to provide different kind of support that makes it easier for system developers to add sensor components to their systems. The support can be given on different system levels from hardware gadgets to higher-level support such as transport, abstraction and interpretation of sensor data.

The Context Toolkit system from Georgia Tech [1] is an example of a higher-level system with which you can incorporate sensor data in your applications. The system is built with a widget approach making it possible for applications to incorporate context data in their application about the same way as you incorporate a GUI component.

The TEA project presents a system architecture as well as a method to support the design of context aware systems [6]. The system architecture consists of several layers where the bottom layer consists of cues that represent an abstraction of the sensor data. The cues are then combined into *contexts*, which can be seen as a high level description of the current situation. These context descriptions can then be fetched by

the applications from a tuplespace. The provided method gives step-by step support for the choice and assembly of sensors as well as for the application development.

The Dense Sensor Networks system [7] focuses on the transport of sensor data and uses small radio equipped “sensor platform”-circuits. The platforms communicate using wireless multi-hop networking. In an example application, the sensor-platforms were connected to motion sensors and their data were used by an application that provided information regarding free conference rooms.

Experiences of sensor usage in ubiquitous computing environments

In the FUSE research group we have been developing a number of applications within the area of ubiquitous computing. Through this work we have gained useful experiences regarding the use of sensors.

Meeting support with active documents

Our meeting support system called fuseONE [8] targeted the problem of document management in project meetings. The system contained Active Documents that can use information concerning who are in the same room to identify the occurrence of a meeting. When the meeting starts the document presents itself on a public display in the room. The system also supports sending documents between computers in the same room. A context sensitive desktop shows the possible receivers of documents present in the room.

This system relies heavily on information about the current locations of people. The location sensors were so called iButtons [9], small memory circuits containing an identifier of its owner. Whenever the users entered a meeting room, they pressed these buttons against one of the readers scattered around the room, thereby actively revealing their presence in the room. The system that provided the context information to the applications is called Context Shadow [10] This system provides a way to find services that are relevant to a person’s current context. The Context Shadow system can use any type of location sensor to set up a connection between a person and a location.

The VAMPIRE storyteller

In the VAMPIRE project the aim was to create a storytelling agent for kids. The storyteller is implemented as a physical humanoid character, placed in an exhibition area. When a person approaches the agent, it responds by turning its head towards the person. As soon as someone sits down on a mat in front of the agent, it will start to display a scary story using text on its belly. The kids can interact with the story by placing different objects in the palm of the agent. These objects will then appear in the story. Every now and then, the story halts, and the agent speaks out a question that the audience can respond to verbally. This would also affect the content of the story. In this project several sensors was used:

- Infrared motion sensors to detect approaching people
- Pressure sensors to detect whether someone was sitting on the mat
- A RFID reader to detect what objects were placed in the hand of the agent
- Speech recognition to understand the answers to the questions

To collect the motion and pressure sensors, we used a Basic Stamp circuit connected to the serial port of a computer. The RFID reader circuit was also connected to the serial port. Finally, the Speech recognition used a microphone that was connected to a computer running speech recognition software. Several computers were involved in the system and to send sensor data between the different components we used a tuple space implementation called TSpaces [11].

A system for non-intrusive messaging

In meeting situations, incoming messages often have a disturbing effect on the participants. In an attempt to tackle this problem a prototype was created where the presentation of the messages could be varied in different ways, by using different modalities, personal and public displays, peripheral and ambiguous renderings etc. Private messages were e.g. sent to private displays embedded at the position where a person where seated. The displays were built using network connected IPAQ's embedded in the table, with iButton-readers connected to the serial pins in the cradle connector. The iButton readers were used to personalize the display services.

In the middle of the table was a knob that the participants could use to select an acceptable level of disturbances for the meeting. The knob consists of a potentiometer connected to a TINI-board [12], a network-enabled microcomputer. Different notification terminals were associated with certain intrusiveness levels. Incoming notifications were routed to appropriate terminals according to the current impact-level received from the knob.



Figure 1. To the left: The Vampire storyteller. To the right: The non-intrusive messaging environment.

Lessons learned

Each of the described projects had its own specific implementation of sensors and a lot of effort was put into low-level implementations. The degree of specialization of

these implementations made it hard to reuse parts of the system for other purposes. It might also be hard to modify the system to fit changes in the application.

Our experiences have also shown that there are lots of similarities between sensors in terms of usage and implementation. One common example is the issue of *identification*, be it for localisation as in the fuseONE project or identification of objects as in the VAMPIRE project. These similarities increase the possibility to create reusable implementations. Another aspect of these similarities is that one instance of an implementation could be used by several applications. A typical example of this would be location or person sensors, which might provide their data to several applications.

In the fuseONE and specifically in the VAMPIRE project each sensor implementation occupied a laptop-pc, a very expensive and clumsy solution. Even if the number of PCs used probably could have been minimised, a solution based on networked embedded computers would probably scale better and occupy less physical space

The implementation process could be supported in several ways. Simplifications regarding low-level installation of sensors would decrease development time a great deal. General components could be used that only requires smaller modifications to support the sensors being used. Some kind of general tools could also be used for the process of fetching or subscribing for sensor data. Ideally, it should be possible to fetch the sensor data using several different protocols to ease the integration with applications.

Platform architecture for room based sensor clusters

In order to deal with the problems described above we propose an architecture for a platform supporting the development of sensor-based applications. The architecture we have considered includes an embedded computer platform with support for a number of different sensors as well as higher-level support for distribution of sensor data to applications. The architecture has been implemented in a reference implementation. This application is an extension of the knob service presented above. The reference implementation does not cover all features of the architecture described below, but only those required for the specific application.

What sensors should be supported?

Since a sensor could be anything between a video camera and a push button switch, it is very hard to create a support platform that could handle all kinds of sensors. We have identified a subset of sensor types that we want to support. Some of these sensor types could be implemented using the same platform and components, but they differ in the way the data is presented to users as well as in how the data is fetched from the sensors.

Identifiers: Sensors that can acquire an ID or other limited information from a “foreign” object. Sensors of this kind include RFID readers, barcode readers, iButton readers etc. Identifiers could for example be used to identify persons and objects that arrive in a room or to various partially defined locations of the room. Some of the identifiers are able to carry data and other just an ID. The identifiers carrying data

might hold specific information regarding their purpose and use. The information could be references, properties or even executable code. For these sensors the application interface needs to be able to listen for new objects that arrive, ask for currently present objects and read data and IDs from these objects.

Passive environment sensors: Sensors that measure something continuously. These sensors are passive in the sense that they don't react on changes of the data they measure. The passive sensors are mainly used when changes in the environment doesn't require immediate response, and when the fetching of sensor data is triggered by events that don't depend on the sensor value. Sensors of this kind could be temperature givers, linear photo-resistors, sound level microphones, positions of objects, persons etc.

Active environment sensors: Sensors that actively indicates changes in the environment. Change events appear if a sensor value is changed or a threshold is reached. The event notifies or triggers listing applications, which can react on the changes immediately. Sensors of this kind include different kinds of alarms, movement and proximity sensors etc.

Interactors: Tangible interaction devices that can give input to local applications. This could be push buttons, knobs, sliders etc. Tangible interfaces have some nice properties, especially in multi user environments. They are easy to share, user actions are clearly visible to other users and they can also be designed to be an integrated part of the physical environment. We choose to include simpler forms of such interaction devices among the supported sensors, since they often can be implemented using the same components as the other sensors mentioned.

(Actuators): Simple output devices that can be used to provide different kind of feedback and status information, often related to the sensors mentioned above. This could be LEDs, simple speakers, motors etc. This group is not sensors at all, but we have included this group among the sensors since we feel that the issue of feedback is important and since there are similarities in terms of implementation.

Sensor platform architecture

The proposed platform consists of a small, network enabled and programmable micro-controller device where a number of sensors can be attached simultaneously. The micro-controller contains firmware and drivers for the sensors as well as some communication abilities.

The idea is that these platforms could be integrated in a limited physical space such as the rooms of a building. A cluster of different sensors could then be connected to the platform according to the current needs. The platform could later easily be modified to cover future modifications and additions of applications in that locale.

To make the sensor platform reusable and easy to adapt to new sensors we propose that the controller should support a modular high-level programming-language for the most of the upcoming cases. In our reference implementation we have been using the TINI board from Dallas Semiconductors. This platform supports Java Me and TCP/IP.

Since the computing power of micro-controllers is limited, we propose that manipulation and distribution of sensor data is performed on an external sensor server, running on a more powerful computer anywhere on the network.

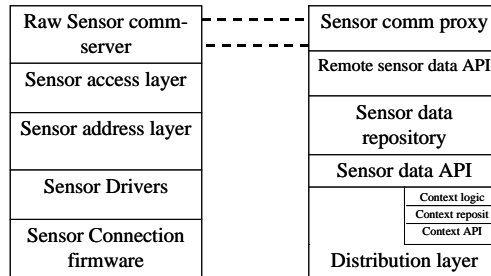


Figure 2. The architecture of the sensor platform

Micro-controller software architecture

Sensor Connection firmware: Software that enables communication with the connected sensors in a unified way. A firmware API contains a set of instructions needed to communicate over any of the available communication ports. The ports or communication channels could be:

- Instructions to read and write the TTL ports on a micro controller
- RS-232 comm. port
- Dallas OneWire network
- IR or IRDA ports
- RF or bluetooth
- Data from peripherals such AD-converters connected to a parallel port

In the reference implementation a potentiometer was connected to a OneWire AD-converter via a OneWire network. (OneWire is a technology from Dallas Semiconductors).

Sensor Drivers: Specific or native implementations to generalize the most basic instruction set for specific sensors or peripherals. Examples of the operations that have to be served by the driver are:

- Read data operations
- Write data operations
- Initialising sensor
- Forwarding interrupts

Sensor address layer: Initialises the sensors and associates specific hardware addresses with access IDs and descriptions that makes it possible to access the sensors without digging into low-level details of how they are connected and addressed.

Sensor access layer: provides a middleware API to the sensors and implements strategies for sharing the raw sensor data. The access layer provides methods to request sensor data and keeps track of event- or poll-based subscriptions for sensor data.

Raw Sensor communication server: Controls the communication channels for distribution of raw sensor data. This layer might also convert the sensor-data to fit the

communication channel. In our reference implementation we have been working with a Dallas TINI-board that support Java Me and IP but not object streams, there so it required translations back and forth to byte-packets for each transmission of data. If the controller board is powerful enough, parts of the sensor server components could be moved to the micro-controller side.

Sensor server architecture

Sensor communication proxy & remote sensor data API: Receives and converts sensor data and delivers it to the data repository. Some applications might be very sensitive for latencies. Then there is an option to use the Remote sensor data API to communicate directly with the micro-controller. To limit the load on the micro-controller it is however preferred to fetch the data from a data repository.

Sensor data repository: A tuple space that stores sensor data and handles incoming sensor events for applications. Provides possibilities for applications to subscribe for changes and events from the sensors. In our reference implementation we have used TSpaces from IBM [11] to implement data repository.

Sensor data API: interface to the repository tuple space. This API is the main interface to the raw sensor data. Typically this API will be remotely accessible to applications through the distribution layer.

Context logic: This layer contains rules that draw conclusions from the sensor-data to infer higher abstractions regarding the local context. The context data can be inferred from multiple sensor-inputs or by some reasoning software. The context logic can also generate events for applications.

Context repository: The Sensor data repository will also contain data with higher levels of abstraction. In [6] a similar architecture is proposed but they offer the context data from a separate tuple space on a higher layer. We propose to have the context as an optional partition of the tuple space and that the refined data is treated similarly as the sensor data.

Context API: Provides an extension of the Sensor data API and provides some extra methods for searching, fetching and manipulation of context data.

None of the context layers has been implemented in the reference implementation.

Distribution layer: The main contribution with this architecture as we see it deals with the low-level layers and the micro-controller platform. On the higher abstraction levels there are several existing frameworks and toolkits that deal with manipulation and distribution of sensor data. We will implement interfaces in this layer that bridges the context data into some existing frameworks:

- Sensor and context data could be wrapped in Context Toolkit [1] widgets.
- Context Shadow [10] could provide means for applications to query for context data.
- The Jini technology from Sun can provide ways of finding and reading sensors remotely. The reference implementation was distributed as a jinni service.
- Distribution of context data using http/xml based SOAP [14] remote calls. SOAP is platform and programming language independent.

- The Event Heap system [13]: Mainly deals with and distributes events from applications.

Conclusions

We have, from our own experiences identified a number of obstacles when creating applications that includes different kinds of sensors. From these experiences, we have created a platform architecture that supports the development of sensor-based services. The architecture has been implemented in a reference implementation where parts of the functionality has been implemented and tested.

References

1. Dey, A.K., Understanding and Using Context, Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing, 5(1), (2001)
2. Schilit, B., Adams, N., Want, R.: Context-aware computing applications. In: First International Workshop on Mobile Computing Systems and Applications, (1994) 85-90.
3. Pham, T., Schneider, G., Goose, S.: Exploiting Location-Based Composite Devices to Support and Facilitate Situated Ubiquitous Computing. In Proceedings of the 2nd International Symposium on Handheld and Ubiquitous Computing (HUC2K), Bristol, UK, September 25-27, 2000. pp. 143-156.
4. Starner T, Kirsch D, Assefa S: The Locust Swarm: An environmentally-powered, networkless location and messaging system. The First International Symposium on Wearable Computers, ISWC'97, Boston, USA, October 1997.
5. José, R., Davies, N.: Scalable and Flexible Location-Based Services for Ubiquitous Information Access. In Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC'99), Karlsruhe, Germany, September 1999. 52-56.
6. Schmidt, A., Van Laerhoven, K.: How to Build Smart Appliances? IEEE Personal Communications 8(4), August 2001. pp. 66-71
7. Conner, S., Krishnamurthy, L. and Want, R.: Making Everyday Life Easier Using Dense Sensor Networks. In Proceedings of ACM Ubicomp, Atlanta Georgia, Oct 2001.
8. Werle, P., Kilander, F., Jonsson, M., Lönnqvist, P., Jansson, C., A Ubiquitous Service Environment with Active Documents for teamwork support. In Proceedings of ACM Ubicomp, Atlanta Georgia, Oct 2001.
9. iButton is a product of Dallas Semiconductor Corp <<http://www.ibutton.com>>
10. Jonsson, M.: Context Shadow: An Infrastructure for Context Aware Computing. To appear at the AIMS 2002 Workshop in conjunction with the ECAI2002 Conference, Lyon, France, July 2002.
11. Wyckoff, P: Tspaces. IBM Systems J., Vol.37, No.3, Aug.1998, pp.454-474, <http://www.almaden.ibm.com/cs/Tspaces>.
12. TINI Board is a product of Dallas Semiconductor Corp <<http://www.ibutton.com/TINI>>
13. Johanson, B. and Fox, A., The Event Heap: A Coordination Infrastructure for Interactive Workspaces. 4th IEEE Workshop on Mobile Computing Systems & Applications, June 2002.
14. Simple Object Access Protocol (SOAP) 1.1, W3C Note 08 May 2000, <http://www.w3.org/TR/2000/NOTE-SOAP-20000508/>