Autonomous Sensor Data Processing for Ubiquitous Computing Scenarios

Lars Nagel[1], Georg Groh[1], Michael Berger[2], Michael Pirker[2]

Content areas:

Multi-Agent Systems: Autonomous Agents
Learning: Case-Based Reasoning
Knowledge Representation/Reasoning: Ontologies

Abstract

The paper investigates local preprocessing of sensor data by autonomous agents. Means of artificial intelligence are applied to facilitate a rich and comprehensive preprocessing in order to reduce the amount of communication in spite of reasonable configuration costs. Scenarios for applications are presented, from which requirements are derived. After a short discussion of related approaches, an architecture for autonomous information processing (AIP architecture) is developed as a base for implementing a framework for the generation of agents. A prototype for computer aided care of elderly people is explained.

1 Introduction

Ubiquitous Computing, Pervasive Computing and Ambient Intelligence are keywords for the idea of computerizing and networking ordinary items and of modeling their local contexts in order to broaden their usefulness, individually and as parts of a larger system. For an effective context-detection, a variety of environment sensors are necessary. We suggest that the task of preprocessing and fusing the large amounts of sensor data in such a Ubiquitous Computing environment can be accomplished best in form of an intelligent, local preprocessing architecture (AIP).
First we will discuss two Ubiquitous Computing scenarios, in which AIP modules can be employed and which demonstrate important aspects of the problem field. Based on these scenarios requirements will be derived for a local preprocessing architecture (AIP) including several aspects such as Filtering, Sensor Data Fusion, Learning and Classification capabilities, Locality, Inter-AIP-Communication and generic interfaces. We will then describe the architecture and some aspects of a prototypical implementation in more detail. A short section on how the architecture was used to realize one of the described scenarios will conclude this paper.

2 Scenarios

Scenarios for Ubiquitous Computing have been suggested in many different areas by many authors. We have chosen two example scenarios which are especially well suited to demonstrate fields of application for the AIP.

2.1 Automotive Applications

In modern cars, several computers process input from numerous sensors that measure great amounts of data to provide safety and comfort. Most of the sensors and tasks are

[1] Technische Universität München
[2] Siemens AG

independent of each other and located in different places. That makes them well suited for intelligent data (pre)processing in local AIP modules. Examples of such higher level applications are:

Automatic air conditioning system: Self-regulating air conditioning systems have become quite common in middle class cars. The necessary data is provided by sensors that could be read out by an AIP module observing the airing, heating, humidity and amount of dust in the air. Typical sensors for this application are solar sensors, measuring the direction of incidence and the radiation intensity of the sun, and sensors for temperature and humidity. Moreover the passenger's settings are part of the input.

Detection of Overfatigue: Many car accidents happen, when drivers are so tired that their reactivity is heavily reduced. Often drivers do not realize, when their attention weakens. Hence, an unbiased computer system could help monitoring the driver's behavior and alerting him by sound, when his driving abilities are decreasing. Appropriate sensors and data could be a camera monitoring his eyes, pressure sensors in the seat, the journey time, the course of speed, the steering behavior and maybe even body sensors taking pulse, blood pressure or breathing rate. As normally a single one of these values is not significant and as false alarms should be avoided, a fusion of sensor data is self-evident.

2.2 Elderly Care

In Germany the proportion of people older than 60 years grew from 14 % in 1990 to 24.1 % in 2001 and will reach 34.4 % by 2030. The number of people who require care will grow from 2 millions today to 2.8 millions in 2020. [Pötsch, 2003; Haustein, 2003] Against this background computer-supported health monitoring of elderly people might be a solution in order to keep the care financially feasible whilst at the same time retaining a maximum autonomy for the elderly.

Such a computer system would be fed by sensor data that reports the behavior of the monitored person, measures bodily functions or retrieves ambient conditions. The sensors indicate, where the person is and what she is doing, and the system logs and processes the information noticing e.g., when she has forgotten to take her pills or when she is in distress. [Ross, 2004] In these cases the person and/or her doctor or relatives are informed. The system's rules should be formulated by application developers with medical knowledge or could be learned by a supervised learning algorithm.

Typically the situations are restricted to a single room and hence can be handled locally. However, there are cases, where data from other sensors improve the reliability e.g. when the person walks to another room, the data from that room confirms the change.

This scenario was implemented as a test case for the AIP module.

3 Requirements

After the examination of operational areas for the AIP module, a few requirements will be derived from the scenarios:

Filtering: An obvious task of a module processing sensor data is filtering. In the elderly care scenario filtering is needed to provide that the medical staff is only informed of alerts and important changes and not burdened with irrelevant messages.

Sensor data fusion: In applications, in which a software agent needs to perceive and picture its environment on the basis of sensory data, it is often necessary to fuse data to get a more accurate or more complete picture. E.g. the detection of overfatigue depends on multiple sensors, and only in combination these sensors will provide sufficient data to model the situation correctly. Providing data on a higher level fusion can also refine the filtering process. Located on a middleware an AIP will thereby be able to substantially reduce data traffic.

Learning: There are two ways to apply learning algorithms in the scenarios. On the one hand the AIP module could be trained, how to classify more complex situations, before it is started. On the other hand it could adapt itself at runtime. In the elderly care scenario, for example, when the user or the doctor is notified, they can give feedback, whether the situation was rated correctly. As the filtering of sensor data is mainly a classification problem, learning algorithms can also support the filtering process.

Locality: In spite of ever faster computers a central computer reaches its limits, when it has to read and process large amounts of raw data. Hence it is advisable, to process or preprocess locally. This separation of concern makes the program logic simpler and clearer and thus more reliable. Moreover, local preprocessing diminishes the amount of communication.

Inter-AIP-Communication: Although locality is given in the described scenarios, local areas often cannot be encircled strictly. In the elderly care home scenario the different rooms of the person's flat are appropriate sections, but changing to another room can be retraced better, when adjacent AIP modules are able to communicate. Another motivation for enabling communication between AIP modules is the joint access to sensors.

Generic sensor interface: Because of the great variety of sensor types it is necessary to provide a generic sensor interface. Every AIP module must allow the connection to every sensor type - during initialization and preferably also at runtime.

Generic management interface: The scenarios showed the need for different means of data processing but also that not all of them are applied in all cases. Since memory space may be limited it should be possible to have an AIP module, which meets the requirements exactly, without wasting resources. That could be achieved by placing a set of fixed modules at the user's disposal. More appropriate is a construction kit with a fixed kernel, to which arbitrary tools can be attached. The application designer then can tailor the module to his needs. In order to initialize and manipulate these tools independent of the actual tool instances, a generic management interface - that is the interface used by the application above - is needed.

4 Related Work

The AIP architecture has to deal with related work concerning several sub-problems such as e.g. choosing the right means for Rule-Processing, choosing appropriate classification and learning algorithms and building and using ontologies for special AIP use cases (as models of the problem domain). For the sake of brevity, we will consider these as background issues and refer to the discussion in [Nagel, 2006].

Context, Context-Modeling and Context-Acquisition are other issue which are clearly related to AIP. Our concept of Autonomous Sensor Data (Pre-)Processing can also be viewed as an architecture for Context-Modeling and Context-Acquisition. Besides general aspects (e.g. definition issues) of context (which are discussed in [Nagel, 2006]) several other Context-Modeling and –Acquisition frameworks have been proposed, such as Dey's *Architecture to Support Context-Aware Applications* [Dey, 1999] or Fuchs's *Context Meta Model* [Fuchs, 2004].

Dey's architecture can be regarded as a *context server* which is a class of systems that extract and aggregate sensor data in order to provide it to applications, but which are not designed to process data. Due to the idea of generating agents to suit a particular scenario, the *Policy Based Adaptive Services for Mobile Commerce* [Rukzio, 2005] are also similar to the AIP. The agents realize the acquisition and rule-based processing of context over a network, but show (therefore) a lack of locality and more sophisticated processing tools. Other architectures specialize on the processing of data from particular types of sensors (such as RFID data, e.g. the *Singularity Architecture* [Rose, 2005]) or aim at a central collection of data continuously provided by a multitude of autonomous ubiquitous computing agents (like *Motes* [Mainwaring, 2002]).

Since in most cases many sensors must contribute data in order to distill a useful context model instance [Wu, 2003], the field of Sensor Data Fusion is of high relevance too. Brooks and Inyar [Brooks, 1998] divide sensor data fusion scenarios in three classes: Scenarios with complimentary, competitive and cooperative sensors. The scenarios that the AIP aims at, clearly belong to the most advanced class, the class of processing cooperative sensors which explains the need for knowledge processing modules in the architecture.

5 Architecture

Based on the requirements derived from the scenarios the AIP architecture is developed. We stated that a generic AIP module to which arbitrary tools can be attached, is preferable to a set of fixed AIP modules. Different implementations of rule engines, Bayesian networks, reasoners, learning components and other tools thus can be connected to the AIP module. The advantage is that for any application and hardware the optimal software can be chosen.

Due to the demand of being able to use multiple and arbitrary processing tools it is advisable to install a central knowledge base to which all processing tools have access and into which all sensor data and processed data is written. Moreover the knowledge base should be the only link between the processing tools, because allowing direct exchange between processing tools is hardly feasible and would unnecessarily complicate program logic and consistent data storing.

The use of a central knowledge base, shared by all other components, suggests a star topology with the knowledge base placed in its center. But to emphasize the importance of the interfaces towards application and sensors a three layer architecture was chosen, consisting of sensor layer, processing layer and management layer. It is displayed in figure 1.

5.1 Sensor Layer

The sensor layer is used to connect arbitrary sensors to the AIP module and to provide data to the knowledge base of the processing layer. The sensor layer consists of the generic sensor interface, the sensor manager and the data supplier.

The sensor interface must allow for pull mode and push mode. In pull mode a sensor sends the actual measured value on request, whereas in push mode a sensor notifies its subscribers, whenever a sensor value is measured. All sensor data is collected by the data supplier that does the forwarding to the local knowledge base or potentially to other AIP modules. The data supplier as upward interface also receives sensor requests from the processing layer.

In sensor requests the relevant sensor can be specified directly by its URL or by its properties, e.g. by the quantity it measures and additional context information. In both cases the sensor manager will be called on to choose a suitable sensor from its catalogue. The AIP module is informed about newly available sensors via the management interface of the management layer. Mobile, wireless sensors communicating with the interface can be marked „in range“ or „out of range“ via push registry.

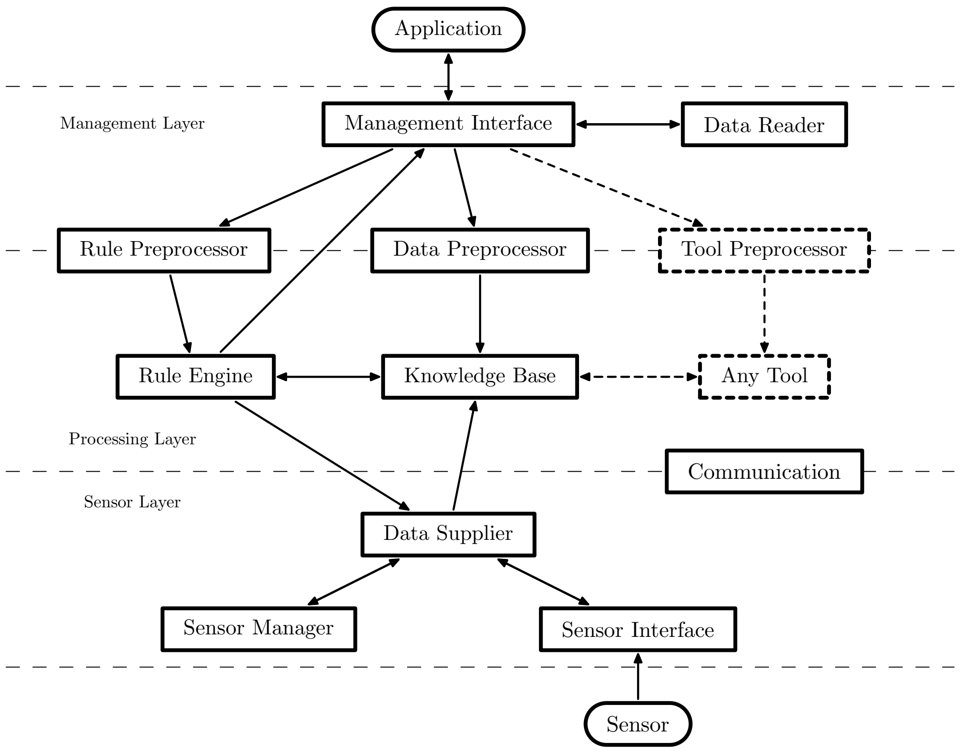

Figure 1: AIP architecture

5.2 Processing Layer

The core of the processing layer is the knowledge base. It gets data from the data supplier and from the application via the management layer. On the knowledge base a rule engine accomplishes two tasks. On the one hand it transforms low level sensor data into data suitable for higher applications, on the other hand it processes this high level data and initiates two types of actions, namely sensor requests and notifications to the application. Sensor requests are sent to the data supplier or to the communication sub module respectively. The communication sub module forwards remote sensor requests via inter-AIP-communication.

Conversion of low level data into high level data is necessary to ease the application programmer's work. As an example it is difficult and hence less reliable to state correctly a condition of the type "motion detector 3 changes to 1". If we assume the elderly care scenario the application programmer would prefer something like "the person is in the bedroom".

The knowledge base is based on an ontology that can be logically divided into two parts. The first part describes the relationship between sensors and the AIP module and is provided by the AIP. Naturally any context data referring to AIP module and sensors are stored here as well as the incoming sensor data. The second part is the one that represents the ontology of the application and has to be written by the application developer. When high level data is deduced from the sensor it needs to be sorted into the world model of the application.

Besides the rule engine other processing tools can be applied. Each of these tools needs data from the knowledge base that offers methods for polling and subscribing. Actually the processing tools could directly work on the data of the knowledge base. Unfortunately there is no knowledge base software on which several tools can work simultaneously. Each tool must retrieve data from the knowledge base, process it and write the results back. Therefore any means of data processing must be adapted to this scheme. Bayesian networks or neural networks, for instance, can store data in their variables (Bayesian network nodes) or input units respectively. Whenever relevant changes to the knowledge base occur, the component’s algorithm needs to be started. The Bayesian network calculates the probabilities for the query variables anew, compares them with given limits and returns the results. The neural network starts the pass through the network, interprets the outputs and sends them back.

5.3 Management Layer

The management layer consists of management interface and data reader. The management interface is used by the application to adapt sensor manager, knowledge base and processing tools. In particular it is possible to add and remove sensors, facts and rules, and furthermore to query the

knowledge base. In reverse direction, the management interface forwards notifications to the application.
The data reader interprets the configuration file passed by the application. Based on its values the initialization is started, preparing the sensor manager and the components on the processing layer, especially the rule engine, the knowledge base and its ontology. During initialization and at runtime, inputs for the knowledge base and the processing tools are sent to the preprocessors first. Due to the preprocessors the management interface can be kept generic, because their task is the transformation of the inputs to the tool's specific format.

5.4 Inter-AIP-Communication

Although a basic idea of this framework is that the AIP modules act locally, it happens that data must be exchanged between AIP modules without the involvement of the application. There are two ways to establish inter-AIP-communication, namely either in the sensor layer or in the processing layer.
If the communication module is part of the sensor layer, only raw sensor data is remotely available. As in the local case, the requesting AIP module should be able to choose whether it wants to receive data in pull or push mode from the remote sensor. When placed into the processing layer, the communication sub module can act in a similar way compared to the processing tools, because it gets data from the knowledge base and also puts data into it. The only difference to processing tools is that the communication process needs to be triggered, e.g. by the rule engine. As an ontology transfer might be necessary before high level data can be exchanged between two knowledge bases of different AIP modules, such an exchange becomes much more difficult to accomplish than a low level data transfer.

6 Implementation

The prototypic implementation follows the AIP architecture and is fully functioning including the inter-AIP-communication, which is realized on the sensor layer. The management interface, however, is restricted to the basic functions for sensor manipulation, rule engine and knowledge base. An extension for additional processing tools is not yet implemented. The entire software was written in *Java* making it platform independent and allowing object-oriented programming.
The implementation of *sensor manager* and *sensor interface* follows the specification *Mobile Sensor API* [Niemela, 2005]. Its generic SensorConnection interface which has to be implemented by every sensor, involves methods for polling and subscribing. Context information about sensors is provided by SensorInfo objects that are held by the sensor manager. When a sensor request comes in, the sensor manager identifies the sensor either by its URL or by its Quantity and ContextType, and the sensor provides data to the requesting AIP module via the data supplier.
The *data supplier* that is not part of the *Mobile Sensor API*, is the interface to the processing layer and fed with sensor requests by the rule engine and by the communication object. Besides sensor and mode (push or pull) sensor requests need to address the sensor's AIP module either by its ID or by a keyword indicating where the sensor is to be searched for. Here, LOCAL refers to the local AIP module, and REMOTE specifies a broadcast. As collection point for all sensors the data supplier forwards sensor data to the respective consumers, namely to the local knowledge base or to other AIP modules via the *communication object*. For this the communication object opens socket connections to other AIP modules.
The *processing layer* contains two factories, namely the KnowledgeBaseFactory and the RuleEngineFactory, which instantiate the knowledge base or rule engine specified in the configuration file. Usable knowledge bases are e.g. RDF / OWL based or relational data bases. They need to implement the interface KnowledgeBase, which provides methods for polling and subscribing and for manipulating and querying the data base. For the prototype *Jena* was chosen in combination with the *Context Engine* [Fuchs, 2004] as its interface. The knowledge base is initialized with three files written in OWL containing respectively the AIP specific ontology, the application-oriented ontology and initial knowledge.
Employed rule engines have to implement the RuleEngine interface that includes methods for adding and removing rules as well as for sending notifications to the application and sensor requests to the data supplier. In the AIP implementation the rule engine *Drools* was chosen, but every Java based rule engine could easily be used. As the common rule languages differ considerably no uniform rule language was established for the management interface. The rule base has to be formulated in the rule language of the employed rule engine.
The management interface is split in two. On the application's side it is represented by the AIPInterface class. Its public interface is identical with the public interface of the AIP class on the AIP module's side and serves therefore as a stub. Besides providing methods to control the tools on the processing layer and the sensor manager the AIP class initializes the AIP module. The XML-based configuration file must specify the AIP module's ID, the IP addresses of application and neighbouring AIP modules, class names of knowledge base and rule engine, and the names of the files containing ontology, initial knowledge and rule base.

7 Use Case

As a demonstration and test of architecture and implementation the elderly care scenario (see section 2.2) was implemented. For this a Java application was written that uses six AIP modules in order to monitor the person's behavior and her bodily functions. Person, environment and
sensors are simulated by software. „Measured“ values are determined by random generators and by actions of the per-

son that can be moved through the flat by mouse clicks. Figure 2 shows the configuration that has been used.
The AIP modules are located in the rooms and on the person's body. Simulated sensors are motion detectors and thermometers in every room, switches in doors, pressure sensors in furniture, RFID readers for tagged items like pill boxes, a pulse meter and a thermometer on the body. Warnings and alerts are displayed on the television screen and sent to the medical staff, i.e. here they are shown in the main window.

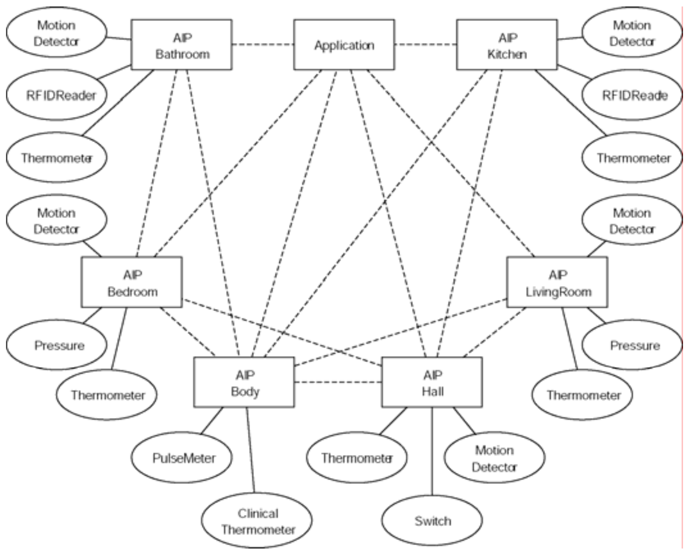

Figure 2: Elderly care scenario

The AIP modules in the rooms are initialized with equivalent ontologies, but with different initial knowledge describing the actual room and its observed objects. The AIP module on the body has a slightly different ontology. The body AIP is actually dispensable, because the connected sensors could dynamically be added to the local room AIP via push registry.

8 Summary

We have argued that an intelligent autonomous data processing layer can be of great value in various ubiquitous computing scenarios. Based upon requirements distilled from the presented scenarios and other scenarios discussed in [Nagel, 2006] we introduced an architecture for autonomous intelligent data processing which is suited to satisfy the requirements.

References

[Brooks and Inyar, 1998] Multi-Sensor-Fusion: Fundamentals and Applications with Software. PrenticeHall, 1998.

[Dey et al., 1999] An Architecture To Support Context-Aware Applications. Tech. Report GIT-GVU-99-23, Georgia Institute of Technology, 1999.

[Fuchs, 2004] A Modeling Technique for Context Information. Mobile and Distributed Systems Group, Technische Universität München, Germany, 2004.

[Haustein et al., 2003] Sozialhilfe in Deutschland 2003. Statistisches Bundesamt, Wiesbaden, Germany, 2003, http://www.destatis.de/presse/deutsch/pk/2003/sozialhilfe_2003i.pdf (URL, Dec. 2005).

[Mainwaring et.al., 2002] Wireless Sensor Networks for Habitat Monitoring. Proc 1st Int'l Workshop on Wireless Sensor Networks and Applications, Atlanta, pp. 88-97, 2002.

[Jena, 2005] Jena - A Semantic Web Framework for Java. http://jena.sourceforge.net/ (URL, Aug. 2005).

[Nagel, 2006] Autonomous Sensor Data Processing. Diploma Thesis, Dept. Of Informatics, Chair for Applied Informatics and Cooperative Systems, Technische Universität München, 2006.

[Niemela et al., 2005] JSR 256: Mobile Sensor API. http://www.jcp.org/en/jsr/detail?id=256 (URL, Nov. 2005).

[Pötsch et al., 2003] Bevölkerung Deutschlands bis 2050. Statistisches Bundesamt, Wiesbaden, 2003, http://www.destatis.de/presse/deutsch/pk/2003/Bevoelkerung_2050.pdf (URL, Dec. 2005).

[Proctor et al., 2005] Drools. http://www.drools.org/ (URL, Oct. 2005).

[Rose, 2005] The Singularity Architecture. http://www.i-konect.com/singularity/docs/SingularityArchitecture.pdf (URL, 2005).

[Ross, 2004] Managing Care Through The Air. IEEE Spectrum, 3 Park Avenue, New York, USA, 2004.

[Rukzio et. al., 2005] Policy Based Adaptive Services for Mobile Commerce, 2nd Workshop on Mobile Commerce and Services (WMCS 2005), Munich, Germany, 2005.

[Russell and Norvig, 2003] Artificial Intelligence: A Modern Approach. Prentice Hall Series in Artificial Intelligence, New Jersey, USA, 2003.

[Wu, 2003] Sensor Data Fusion for Context-Aware Computing Using Dempster-Shafer Theory. PhD Thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, 2003.