A Flexible Concept to Program and Control Wireless Sensor Networks

Bernhard Hurler
Institute of Telematics
University of Karlsruhe
Germany
Telephone: +49 (721) 608 6400
Fax: +49 (721) 388097
Email: hurler@tm.uka.de

Martina Zitterbart
Institute of Telematics
University of Karlsruhe
Germany
Telephone: +49 (721) 608 6400
Fax: +49 (721) 388097
Email: zit@tm.uka.de

***Abstract*— Wireless sensor networks have become a very attractive research topic in recent years. Many academic and professional research groups made efforts to construct operative hardware devices and sophisticated software to meet the special conditions in their projects. But there has been little done to create a general structure for smart sensors to cooperate and to offer their services to human or software clients. In this paper we present first results of our investigations in this topic. As a test scenario and source of inspiration we set up a sensor network prototype in an office situation, where the physical environment should be measured and adjusted according to specific conditions. In particular the light and humidity state of potted plants within an office should be autonomously adjusted to the plants' special needs as most research associates in our lab forget to care for their plants on a regular basis. On the basis of this prolific scenario we introduce a first stage middleware system architecture providing service distribution and accomplishment within wireless sensor networks. Core components of the architecture have been implemented in hardware and software to show the feasibility and abilities of our approach.**

I. Introduction

In recent years the field of wireless sensor networks has attracted considerable interest among numerous research groups all over the world. Efforts has been made to create small and power efficient hardware (e. g. SmartDust [1]) to allow small battery powered devices enhanced with sensor capabilities to communicate wirelessly and give information of the physical world. There are proposals as well for specialized software running on devices with limited power, memory and computation resources. TinyOS [2] is an example for an attempt to provide basic functions of an operating system on small devices. Also middleware aspects has been the focus of research endeavours yet. CORTEX [3] provides an architecture for sensor environments, which enables sensors and actors to communicate via gateways. These gateways allow QoS-communication and can provide real-time guarantees, but consequently must be well equipped with energy and computing resources compared to the above mentioned sensor types.

The various approaches are quite successful in executing the specific tasks or example scenarios they are designed to. Nevertheless they tend to have a monolithic implementation and do not provide a generic architecture to implement new tasks or change ongoing tasks in a simple and well structured fashion. On the other hand middleware architectures are proposed to simplify the access to sensor networks by installing sophisticated infrastructure, but thus affecting the "small & simple" paradigm of sensor networks.

Therefore we introduce in this paper a generic system architecture for sensor networks, which could function as a basis for a middleware component allowing easy and flexible access to the functions of a sensor network.

In the following section we present our approach to a generic system architecture for sensor networks middleware. Section III contains the description of the flower pot scenario, which we build up in hardware and software to evaluate our ideas. Then we give insights into the hardware and software implementation of the architecture and the flower pot scenario. In the last section we give an overview of the current stage of our research efforts and our mid-term plans.

II. Architecture

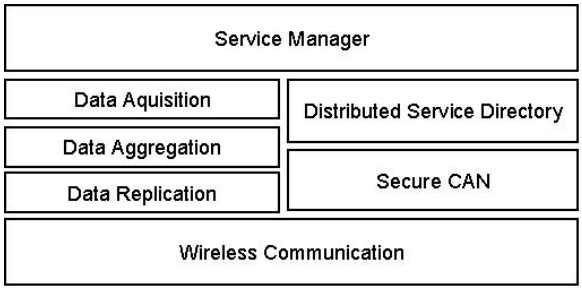

Fig. 1. Architecture for sensor networks

In fig. 1 we present a common architecture for sensor networks.

The lowest layer of our architecture is a basic communication layer. It provides wireless communication between the single sensor nodes. The sensor network will span the space of all nodes which can be addressed by the particular communication layer used in a setting. This can be a single room with perhaps tens of sensors if the communication

technology is limited to a relatively small range and provides only direct communication. It may though comprise a whole office, a campus, or a nearly arbitrary region or combination of sensor equipped areas if the communication range is large enough and/or multi-hop connections are possible.

The Data Manager (including *Aquisition*, *Aggregation*, and *Replication*) is responsible for the definition and handling of application specific data types. Primitive data types are defined for physical values, which are measured and sent by sensors. They consist out of an identifier for the specific value, its age, its accuracy, and its origin (which may be a sensor-ID, location information, a functional description, or any other appropriate information, which identifies the ”author“). Complex data definitions may additionally include aggregation rules, so that data composed of many single data portions can be collected and combined.

The *Distributed Service Directory* serves as a database for service descriptions. It is used by the *Service Manager*, which can insert, lookup, and alter descriptions in the Service Directory. The integrity of the service descriptions is crucial for the functionality of the whole sensor network. The Distributed Service therefore is based on a *Secure CAN* [4], which provides robustness and security.

The Service Manager is responsible for receiving and accomplishing services. A sensor network’s operation depends on the services it contains, respectively on the services it executes. Services are executed to collect sensor data and to alter the physical environment with the aid of actuators. To fulfil more sophisticated tasks, in which numerous sensors and actuators are involved, a whole bunch of single services can be necessary. Services can be composed of other services thus allowing chronological cycles or conditional execution of services.

In this paper we concentrate on the latter modul, which is primarily responsible for a functional sensor network. Nevertheless other parts of the architecture are referred where appropriate.

III. Scenario

A sensor networks consists out of several small devices, which are autonomous in their communication and computation capabilities. All of them can be equipped with sensors and actuators to measure and alter physical values in their environment.

To have a sensible and prolific test case for our middleware approach we designed a sensor network scenario with a couple of sensors and actuators. We constructed a flower pot capable of sensing humidity, a correspondent actuator granting the right amount of water for the flower, and a jalousie regulating the light conditions in cooperation with a light sensor (see fig. 2). All sensor (actuator) devices act autonomously and are expected to fulfil their respective primitive tasks (i. e. measuring or altering physical values). To accomplish the common goal of cultivating the flower they have to communicate with each other and react to static preconditions or dynamic change of the environment.

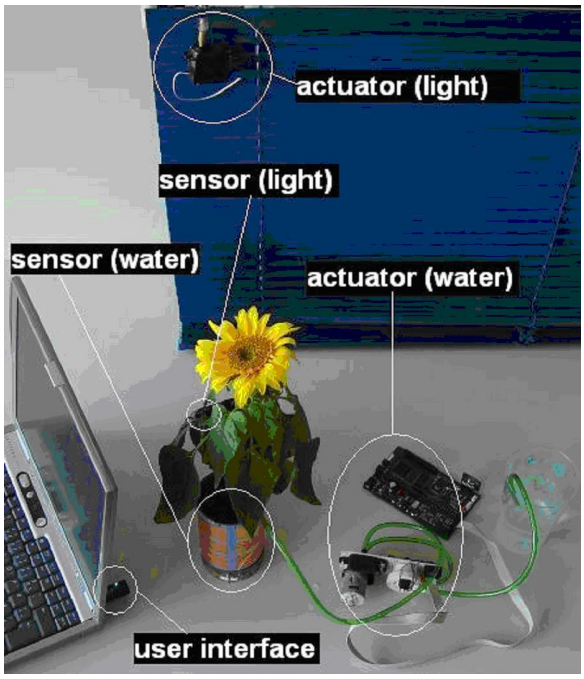

Fig. 2. Autonomously cultivated flower (with automatic water and light adjustment by a sensor network

This scenario has most of the components and characteristics with which middleware systems have to deal. Multiple tasks have to be fulfilled whereas several small tasks are necessary to accomplish a common goal. The tasks may change over time and have to be altered or replaced either by machine or human interaction. The user issues a command (”cultivate this flower“) to the sensor network and the sensor network self-organizes to fulfil this task.

Fig. 3. ”Saucer“ for the flower pot with micro controller and Bluetooth module

IV. Hardware Details

The whole ”flower“ scenario consists out of four individual smart devices. Each of them is equipped with an Atmega128L

micro controller, a Bluetooth module (class 2), and device dependent sensors or actuators (see fig. 3).

The two sensors used for the flower pot scenario are a light sensor and a humidity sensor. The light sensor is a simple photodiode attached to the upper part of the plant. Dependent on the illumination it generates a voltage, which is read out by the micro controller. A special flower pot serves as a humidity sensor. The pot is surrounded by two thin cuprious films. They build up a capacitor, whose capacity depends on the pot's content respectively the humidity of soil.

The two sensors' counterparts are two actuators, which are responsible for light adjustment and water regulation. A motor can change the angle of the jalousie's lamellae, thus giving the opportunity to shade the flower or to brighten up. A water pump regulates the soil's humidity by pumping water out of a container into the flower pot.

The micro controller is responsible for the internal communication with its respective sensor or actuator and with the Bluetooth module. Furthermore it provides the smart part of the whole device, which is respectively the implementation of the architecture presented in the last section. In particular it provides the (external) communication with the sensor network via Bluetooth and the distribution and execution of primitive or complex services.

V. Implementation of the Service Manager

Each smart (sensor or actuator) node has an own service manager. It can receive new services, exchange altered services, or delete obsolete ones, and is responsible for their execution in the correct order. The services itself are written in a newly developed language, which allows a nested description of simple and complex tasks to be accomplished by single devices or by the whole sensor network. All service descriptions consist out of four basic service types. There are two primitive types (query and order) and two complex types (conditional and repetitive).

A. Primitive Services

Primitive services are the basic services of each smart node. They are executed only once and cannot initiate the execution of other services. All devices provide at least one *primitive service*. Smart nodes equipped with a sensor can be queried by a *query service*. Its description has following attributes:

- service identifier,
- effector,
- receptor,
- physical value,
- required accuracy,
- maximum age.

The *service identifier* is unique in the entire sensor network. If the service manager receives a service with an already existing identifier, the old service description gets obsolete and is replaced by the new one. Thus it is very easy to either delete services or to alter their functionality by just sending a new description with the same service identifier. The *effector* attribute designates the smart node, who is responsible for the execution of the service. Accordingly the *receptor* attribute designates the smart node, who is interested in the result of the service respectively the sensor data. The sensor data itself is defined by the *physical value*. Currently this attribute comprises implicitly the complex semantics of data description like range of data, categorisation, detail level, etc. (see hereunto section VII). Additionally the *required accuracy* and the *maximum age* of sensor data can be declared, which is primarily for future use, when data is replicated and/or aggregated in the sensor network.

The second *primitive service* is the *order service*, whose attributes are:

- service identifier,
- effector,
- receptor,
- physical value,
- amount,
- priority,
- valid time.

The first four attributes has the same meaning as in the *query service*. The attribute *amount* denotes, how the accordant *physical value* should be altered. This change could be bidirectional and therefore reversible (in the jalousie case) or unidirectional and irreversible (in the water pump case). The attributes *priority* and *valid time* are necessary to prevent or resolve concurrent access to a single actuator device. If a *order service* sets an actuator device to a given value for a specified time period, it prevents other *order services* with lower priority from accessing the actuator until it is released (determined through *valid time*). This can be necessary, if two services pursue different goals, resulting in a permanent oscillation of the actuator value, or if a human wants to take control of an actuator (for example to stop watering during a meeting).

B. Complex Services

Complex services are services which can initiate the execution of other services. These initiated services are not limitted to *primitive services*, but can be *complex services* as well. There are two service types belonging to this category. To implement recurrent tasks in the sensor network there is the so-called *repetetive service*. In order to react reasonably to a changing environment a complex *conditional service* is available. The latter one has the following description:

- service identifier,
- effector,
- service identifier #1 of a *query service*
- service identifier #2 of a *query service*
- comparator,
- then-list of service identifiers,
- else-list of service identifiers.

The *service identifer* is unique as in the case of *primitive services* and is used analogously to create a new, delete, or alter an old service of this type. The *effector* attribute tells which smart node is responsible for the execution of the *conditional service*. In contrast to the *primitive services*, where the

executor is limited to the appropriate sensor or actuator node, the service description programmer can here nearly arbitrarily choose a smart node as executor. Nevertheless not all nodes are equally appropriate, because the *conditional service* depends on the result of two *query services* (i. e. *service identifier #1 and #2* compared by the *comparator*). Obviously these two *query services* are not compulsorily executed on the same node as the *conditional service*. Thus communication costs can be minimized choosing the *effector* of the *conditional task* cleverly. After evaluating the two *query services* one of the two identifer lists are executed. The execution location of these dependent services affects the "ideal" *effector*, too.

The other *complex service* is the *repetitive service*, whose description consists of following attributes:

- service identifier,
- effector,
- duration,
- frequency,
- repeat-list of service identifiers.

The *repetitive service* is used to implement tasks of the sensor network which have to be fulfilled multiple times and over a period of time. The attributes *duration* and *frequency* specify how long and how often the services in the *repeat-list* are iterated.

C. Service descriptions of the flower scenario

In the flower pot example there are two main services, which are executed at the same time. One service makes sure that the light is appropriate for the flower; the other one takes care of water provision. The first one is a *repetitive service* which continuously reruns two *conditional services*. Both of them evaluate a *query service* on the light sensor attached at the flower. Assuming that there is too much light for the flower the first *conditional service* makes the jalousie decrease the solarisation; the second one does vice versa in the case of too little light.

The second *repetitive service* responsible for appropriate humidity in the flower pot is similarly implemented. If changes in the service description occur, e.g. a new plant with different needs is bought, it could be propagated with very little effort to the sensor network. It then reacts immediately to the modifications without the need for "wired" contact to the smart devices.

VI. Wireless Communication - Hardware and Software

To store new services in the smart devices Bluetooth is used as communication technology. We use a Bluetooth stack developed especially for the use on micro controllers with small memory (see [5] for more information on this). The devices are communicating via Bluetooth directly or via a dedicated central desktop computer. We decided to build up our flower scenario with the centralised approach. This allows observing the communication activities in the sensor network and gives the possibility to change network characteristics (like accessibility and communication speed) and evaluate the effects easily. Bluetooth is used for our test bed, because off-the-shelf hardware is cheap and available. Nevertheless the proposed architecture itself is independent of the underlying networking layer.

VII. Current stage and mid-term goals

At the current stage we are able to distribute services via Bluetooth and to make the devices accomplish their common task autonomously. Both single primitive services and complex services can be changed or replaced easily and propagated wirelessly to the sensor network.

Still there is work to be done to define properly the communication protocol of the Service Manager responsible for exchanging services and user data between the different smart devices. The prototype implementation of the protocols serves as a proof of concept but has to be refined in the future. We already began to address the in section V mentioned problem of the *physical value*. Currently this attribute of the two *primitive services* is only a single identifier, but contains a lot of semantics. By defining a reasoned data description language, following the abstraction principles like in the service description language, the flexibility of our programming concept will be further increased. Another problem of the service programmer is to decide on which nodes services should be executed. To assist the service programmer, we plan to implement algorithms, which can optimize a set of services with regard to communication costs by autonomously choosing the *executor* node.

The partial implementation of our general architecture and its application to the flower pot scenario gave promising results. Based on the knowledge we could obtain, while dealing with our prototype, we are confident to improve and complete the current prototype to a functional and useful middleware for wireless sensor networks.

References

[1] B. Warneke, M. Last, B. Liebowitz, and K. S. Pister, "Smart dust: Communicating with a cubic-millimeter computer," *IEEE Computer*, vol. 34, no. 1, pp. 44–51, Jan. 2001.

[2] P. Levis and D. Culler, "Maté: a virtual machine for tiny networked sensors," in *ASPLOS*, Dec. 2002.

[3] P. Veríssimo, V. Cahill, A. Casimiro, K. Cheverst, A. Friday, and J. Kaiser, "Cortex: Towards supporting autonomous and cooperating sentient entities," in *Proceedings of European Wireless*, Florence, Italy, Feb. 2002.

[4] H.-J. Hof, E.-O. Blaß, M. Zitterbart, and T. Fuhrmann, "Design of a secure distributed service directory for wireless sensornetworks," in *Proceedings of EWSN*, Berlin, Germany, Jan. 2004, accepted and to be published.

[5] T. Fuhrmann and T. Harbaum, "Using bluetooth for informationally enhanced environments," in *Proceedings of the IADIS International Conference e-Society*, Lisbon, Portugal, June 2003.