

Towards a Service Oriented Architecture for Wireless Sensor Networks in Industrial Applications?

Rudolf Sollacher, Christoph Niedermeier, Norbert Vicari **
Maxim Osipov ***

(e-mail: firstname.lastname@siemens.com)

** Siemens AG, Corporate Technology, Munich, Germany

*** OOO Siemens Corporate Technology, Moscow, Russia

Abstract: We discuss the introduction of service oriented architectures to wireless sensor networks (WSN) in industrial applications. We give an example for a WSN architecture as applied in the EU project SOCRADES in order to explain constraints preventing a full-fledged service oriented approach. Such an approach appears to be beneficial for applications like diagnostics or monitoring where service composition can provide new functionalities. However, the limited resources in WSN must be taken into account. For control applications additional constraints like determinism or latency bound severely limit a loose coupling of services. As a consequence we propose a support by appropriate design and engineering tools.

1. INTRODUCTION

Important trends like urbanization and demographic change will lead to an increasing scarcity of natural resources like energy and water, a growing need for environmental protection, increasing mobility, regional shift of economic gravity, individualization and shorter product life cycles. The corresponding challenges for manufacturing can be mastered with the concept of an “intelligent factory” with its seamless product and production life cycles linking the real and the digital world (Schott [2007]); product requirements, product use and service, plant optimization, maintenance and operation are tightly linked to product and production design, simulation and virtual commissioning. The life cycle of an intelligent factory is characterized by four main steps: (i) Design and modernization is based on a holistic modeling, (ii) engineering takes advantage of autonomous components like working cells, (iii) commissioning is simplified by self-configuration of these interconnected autonomous components and (iv) operation benefits from self-optimization and self-healing functionality. As a consequence, each of these technological components carries with it an evolving “digital shadow” including knowledge of its state and current environment.

This paper discusses wireless sensor networks (WSN) as one part of these autonomous technological components and their service-orientated integration in industrial automation environments as planned in the European research project SOCRADES¹. A major goal of this project is to create new methodologies, technologies and tools for the modeling, design, implementation and operation of networked hardware/software systems embedded in smart physical objects. Typically, each entity is constituted of hardware, sensing/actuating resources, control software and embedded intelligence. These entities are capable of working in a pro-active manner, initiating collaborative

actions and dynamically interacting with each other in order to achieve both local and global objectives, down from the physical machine control level up to the higher levels of the business process management system.

One special - and due to its wireless communication infrastructure technologically very outstanding - instance for such embedded networked hardware/software systems on the sensor level of the automation pyramid are WSN.

2. WIRELESS SENSOR NETWORKS

2.1 Sensor node hardware

The SOCRADES WSN architecture bases upon the assumption that the WSN will be connected to a wired network, utilizing a Gateway. While an Ethernet based wired control network allows for high bandwidth, a WSN is much more resource constrained. Sensor nodes used in SOCRADES are based on low power hardware components widely used for wireless sensor platforms. The microprocessor (MSP430F1611) runs at 8 MHz and provides 10 kByte RAM and 48 kByte plus 256 Byte flash memory. The transceiver chip (CC2420) supports the IEEE 802.15.4 standard with maximum transmit power of 0dBm and maximum bandwidth of 250 kbit/s.

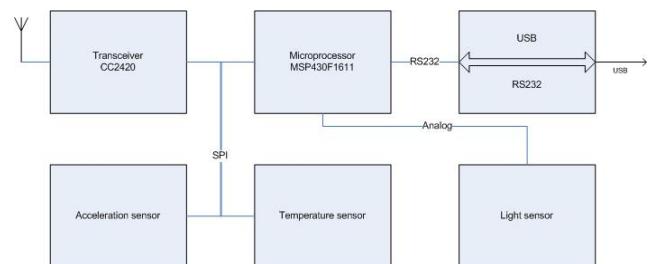


Fig. 1. Hardware architecture of a typical sensor node.

¹ <http://www.socrades.eu>

Even Bluetooth based approaches will not have more than $3 * 723 \text{ Kbit/s} = 2,1 \text{ Mbit/s}$ in the best case. This is orders of magnitudes less bandwidth than is available in Ethernet based networks, which provide 100 Mbit/s (i.e. PROFINET) or even more. Thus, the Gateway's task is to translate the data from the (mostly wired) control level into some "compressed format", so that it can be transported to the WSN.

Although wireless sensors may be powered by wire, we consider the case of battery-powered or energy-autarkic sensor nodes. Only these truly wireless systems will leverage the full cost benefit of no wiring.

2.2 Sensor node software

As sensor nodes are usually quite restricted in resources (memory size, processing power, battery capacity), software for sensor nodes must fulfil the following requirements in order to fit into such systems and, at the same time, run with sufficient performance and in an energy-efficient way:

- Extremely small footprint
- Extremely low system overhead
- Extremely low power consumption

On the other hand, as design and development of software for embedded systems can be very difficult, a simple and compelling programming model for the application programmer is needed. The programming model shall provide the following features:

- Suitable abstractions of low level hardware interfaces (e.g. communication interfaces with radio units or measuring devices)
- Safe encapsulation of mechanisms and data that are not to be modified by the application programmer (e.g. lower layers of the communication stack, middleware functionality, facilities as timers, clocks, power management modules)
- Event orientation (asynchronous rather than synchronous processing, i.e. no polling for incoming communication messages or sensor events, but rather triggering of event handlers)
- Basic mechanisms for sensor data handling, networking, scheduling, power control, etc.

These requirements are partly fulfilled by a class of operating systems / platforms that are specially designed for the extreme resource restrictions imposed by wireless sensor nodes. The sensor operating system TinyOS² based on the nesC language (Gay et al. [2003]) has been designed to fulfill the requirements listed above. Therefore, TinyOS is chosen as the basis for the sensor node software architecture in this work package.

2.3 Network topologies

It is very likely that so-called mesh networks will be utilised in process automation or monitoring applications. A mesh network allows for redundant connections because there is more than one path from one node to another. If the mesh topology is not pre-configured at engineering time, the network has to be self-organising in order to be

² <http://www.tinyos.net>

able to route data from one node to another. With this, the mesh network can be made self-healing. That is, if a connection (or a node) breaks, the network can reorganize itself in order to fill the gap induced by the broken entity.

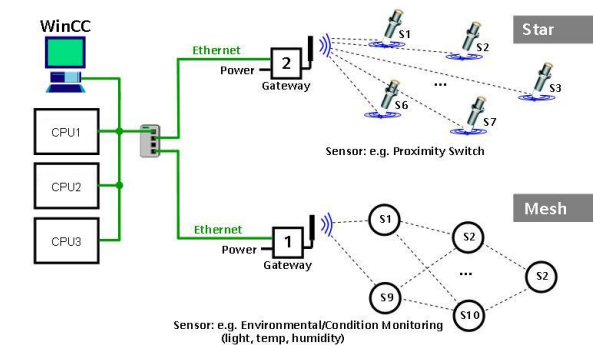


Fig. 2. Network topologies considered in SOCRADES. The star network applies mainly in factory automation while the mesh topology will be the dominant topology in process automation.

In manufacturing automation, the deadlines are typically rather short and determinism is required. Correspondingly, a star network with a strict TDMA approach seems to be much more appropriate, as it allows much better control of the timely behaviour of the network.

2.4 Gateways

As a key goal of SOCRADES is to specify a service-oriented framework for device-level infrastructures, it is among other things necessary to specify and implement an enhanced version of the device-level SOA infrastructure based on the Device Profile for Web Services (DPWS) - for encapsulating intelligence and sensing/actuating skills as services, as well as to specify associated frameworks for management and orchestration of device-level services.

To match the SOA paradigm, within the WSN infrastructure at hand, communication with the concentrators is performed using OPC UA (OPC Unified Architecture). The underlying communication protocols used by OPC UA are either based on Web Service protocols and data formats or on binary protocols and data formats. It allows access to information like "runtime" data, events, alarms, methods as well as higher level information (meshed objects) which can be - as aimed at in the SOCRADES project - defined and modelled by DPWS.

2.5 Typical use cases

In a manufacturing environment WSN will be used for monitoring and for control purposes. Typical use cases associated with these applications are:

- Device integration: A new wireless device is joining the network. First, it has to be integrated into the network. Possible properties to be negotiated or assigned are a node ID, a network ID, communication links to neighboring nodes and channels for medium access and routes to one or several gateways. The device

finally publishes its description and basic services. In a next step, the device has to be integrated into the application. Parameters relevant for sensing etc. have to be set by an external authority.

- **Data collection:** The network delivers data to a gateway as illustrated in figure 3. This service may be requested on demand regularly or just once, or event based.
- **Data aggregation:** Sensor data and device state values are aggregated in order to provide relevant information instead of raw data. Examples are calculation of averages or diagnostic information. While such a service could be realized by data collection and subsequent aggregation at the gateway, it may often be more energy efficient to perform the aggregation inside the WSN; this is illustrated in figure 4. This service may as well be requested on demand regularly or just once, or event based.
- **Decentralized control:** Sensor data and actuator state values are processed to yield some new actuator state. This use case can be considered as a special case of data aggregation with the gateways replaced by the actuators.
- **Software updates:** New software or patches have to be distributed and deployed to wireless sensors and actuators.
- **Device disintegration:** Finally, a device may be removed. Correspondingly, the information about this device should be removed from the network.

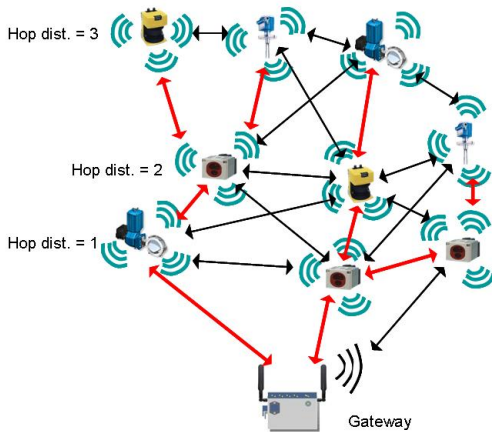


Fig. 3. Collection tree: Sensor nodes send their data to a neighboring sensor node which is closer to the gateway.

3. INTRODUCTION TO SERVICE ORIENTED ARCHITECTURE

3.1 What is SOA?

Service-Oriented Architecture (SOA) is an architectural concept for designing and implementing distributed systems such that functionality is encapsulated into interoperable services. The main goal is to partition business functionality in a way that it can be orchestrated in a loosely-coupled and re-usable fashion. Also, the integration into workflow systems is facilitated by this architectural style.

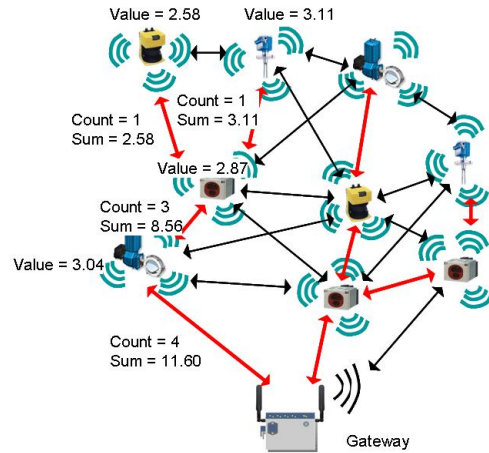


Fig. 4. In-network aggregation: Sensor nodes forward the sum of sensor values and a count value to a parent node in the aggregation tree.

Currently, no clear and concise definition of SOA exists. However, a common denominator is the service abstraction, the concept of loose-coupling, together with service orchestration. A service is the smallest building block of a SOA. Service orchestration ensures that separate and independent services interact in a way that a larger business application's goals are met. As an architectural concept SOA abstracts from any particular operating system and any implementation details. It does not matter which platform and which language is used to implement some service. The functionality of a service is entirely defined by the service interface. Services cooperate by passing data from one service to the next, according to the business process or service orchestration. In an abstract sense, SOA is a distributed variety of object-oriented and modular programming.

3.2 What is a Service?

A service is an implementation of a clearly defined self-contained function that in principle operates independent of the state of any other service. It has a well defined set of platform-independent interfaces and operates through a pre-defined contract with the consumer of the service. Services are loosely coupled and all interaction takes place through the interfaces. Loosely coupled here means that any service only needs to know how to operate on its own interface and does not need to depend on any other service's implementation. Only the orchestration gives the semantic and stateful composition into a larger application. In the typical setting of Internet applications, data between the consumer and the service are passed in specialized XML formats over a variety of possible protocols. The main protocols that Internet services use today are SOAP (Simple Object Access Protocol) and REST (Representational State Transfer).

3.3 Related approaches

TinyDB is a query processing system based on a routing tree (Madden et al. [2005]). Given a query, data are collected, filtered and aggregated. However, TinyDB is not a service infrastructure. A similar statement holds

for Cougar (Gehrke and Seshadri [2000], Yao and Gehrke [2002]).

ATLAS is a service-oriented sensor platform with middleware based on OSGi (King et al. [2006]). So-called Atlas nodes, modular hardware platforms with sensor/actuator, microprocessor and communication layer, provide only limited processing power; most of the service functionality is executed on a stand-alone server where Atlas nodes register. Although wireless communication interfaces are planned, low power operation was not a major design criterion.

In RASA (Resource Aware Service Architecture), services are software modules which can be injected at runtime and installed by other sensor nodes (Blumenthal and Timmermann [2006]). Homogeneous sensor node platforms are assumed. Service messages consist of code and public data; the code will be installed by the receiver node, if not yet done, and the code is used to process the message's public data with the receiver's public data. Details of an implementation and performance evaluation is not provided. A drawback of this approach is the regular transmission of code increasing the communication load and thus reducing sensor network lifetime.

OASiS is an object-centric, ambient-aware, service-oriented sensor network programming framework and middleware (Kushwaha et al. [2007]). Physical phenomena like a moving heat source and its effect on temperature sensors are represented by finite state machines and correspond to logical objects. One node among those detecting such a phenomenon becomes the responsible object node. This role can move to another node, if the phenomenon moves or if an object nodes is low on battery. Object nodes also request and compose the services needed for a specific application. Middleware services provide supporting functionality; a node manager responsible for message routing between services, a service discovery, an object manager and a service composer.

Within the EU-project SIRENA (Jammes and Smit [2005]), a service-oriented architecture for industrial devices was developed. Six levels of functionality constitute the interaction patterns of device-level SOA: Addressing, Discovery, Description, Control, Eventing and Presentation. Web services are proposed as the preferred vehicle for implementation. Due to its still large overhead, this approach can not be applied directly for wireless sensor networks with limited computational power and tight energy budgets.

3.4 Advantages of Service-oriented Architectures

- (1) **Platform and Programming Language Independence** - Since services can be published and consumed across development and operating platforms, an application can leverage existing legacy components that reside on different types of servers and were built using different technologies. The main challenge is to integrate heterogeneous components into a larger application. This applies for wireless sensor networks if standardized protocols for communication between services exist.
- (2) **Focused Developer Roles** - Since a service is a discrete implementation independent of other services,

developers in charge of a service can focus completely on implementing and maintaining that services without having to worry about other services as long as the pre-defined contract is honoured. However, the semantics and orchestrability of such a service must be made explicit and must be documented and managed accordingly. For WSN-services this information should be stored on an external server in order to save resources.

- (3) **Location Transparency** - Services are often published to a directory where consumers can look them up. The advantage of this approach is that the service can change its location where it is executed at any time. Consumers of the service will be able to locate the service through the directory. The underlying traditional assumption that communication is basically for free does not hold for WSN. Communication costs energy.
- (4) **Code Reuse** - Since SOA breaks down an application into small independent pieces of functionality, the services can be reused in multiple applications, thereby bringing down the cost of development. This demands that the principle of stateless business logic encapsulated into a small and precisely defined service is honoured at all times. TinyOS with its concept of components and interfaces provides this kind of feature locally. Distributing functionality can save memory provided energy consumption is taken into account carefully.
- (5) **Greater Testability** - Small, independent services are easier to test and debug than monolithic applications. This leads to more reliable software. In WSN this benefit can not be leveraged in general, since distributed functionality is severely restricted by the unreliable wireless communication.
- (6) **Parallel Development** - Since the services are independent of each other and contracts between services are pre-defined, the services can be developed in parallel - this shortens the software development life cycle considerably. This holds for services in WSN as well.
- (7) **Better scalability** - In principle loosely coupled and distributed systems do scale better. There can be multiple instances of a service running on different servers. This increases scalability. However, the impact of communication, load balancing, and management may interfere with scalability. This is particularly important for WSN with constrained resources.
- (8) **Higher availability** - The same argument as above applies to a possible high availability. Services in industrial WSN are usually location based. This limits the possibility for redundant designs and thus higher availability may not be given.
- (9) **Dynamic deployment** - New services can be deployed on demand or existing ones can be updated. In WSN this is accompanied by high communication costs and should thus be the exception rather than the rule.

4. REQUIREMENTS ON WSN IN INDUSTRIAL AUTOMATION

WSN to be applied in industrial automation are expected to show deterministic behavior. Service discovery, a basic

component of service-oriented architectures, in general is not deterministic.

Flexibility and energy efficiency have to be combined in a useful way. For instance, introduction of an appropriate in-network processing feature may facilitate new kinds of requests and at the same time save energy. This is an interesting perspective for implementing new diagnosis or monitoring tasks whose response behavior is usually not very time critical. This is less applicable for control tasks with their often tight timing constraints.

Similarly, constraints like determinism or realtime capability together with regulations setting limits on e.g. latencies may severely restrict a full-fledged service oriented approach. In order to meet such constraints and regulations, a sophisticated design approach is required and - accompanied with it - support by accurate simulation tools in engineering.

As an example for the consequences of WSN requirements for software architecture design let us take a closer look at in-network data aggregation as illustrated in figure 4. This aggregation scheme is very energy efficient as the communication effort is reduced to a minimum. In a TDMA-scheme, each node must listen to a request from its parent node and forward this request to its children and on the way back it has to receive the results from its children, take the sum of these results with its own sensor value, sum the count values and increase the sum by one, and finally forward the resulting sum and count values to its parent node.

The crucial point here is that energy efficiency is reached by a tight coupling between networking and data processing. Using a SoA approach such a service would be realized by composing more elementary networking and data processing services. For example, the networking service provides at least the following interfaces used by the data processing service:

- **MessageReceived(NodeID, Msg)**: Signals reception of a message from neighbor node with ID **NodeID** and message **Msg**. This interface is used by the data processing service to extract data from the message and process them.
- **SendMessage(NodeID, Msg)**: Sends a message **Msg** to node **NodeID**.
- **getChildren(Children)**: Provides a list **Children** with IDs of children nodes. This interface is used by the data processing service which waits for all children to deliver their results before the own result is forwarded.
- **getParent(ParentID)**: Provides parent node ID **ParentID**. This interface is used by the data processing service to forward its own result to the proper parent node.

Data processing services at different sensor nodes are linked by their corresponding networking services. A request for data aggregation is propagated along the routing tree starting from the gateway. This request specifies the data aggregation task, e.g. whether temperature of pressure sensors shall be aggregated or whether the task should be completed only once or in regular intervals. By forwarding the request to the children nodes the parent

subscribes for the aggregated results for the respective subtrees.

5. WSN SOFTWARE ARCHITECTURE

While it is possible to use a virtual machine in combination with TinyOS in order to be able to provide dynamically deployable services, the possibilities of such an approach are quite limited. Instead, we propose to provide a set of standardized service interfaces that are implemented by service components being either part of the middleware or being specified at application level. The association of selected components with a particular node program has to be made at compile time; dynamic loading of these components is not possible. The seemingly inflexibility of this approach can be alleviated by having each service component implementing an activation interface that allows dynamic activation and deactivation of service components. A service management component that keeps track of the availability and activation state of all service components can provide information to other nodes in order to support dynamic service discovery mechanisms. Depending on their capabilities and context, nodes can be programmed differently, thus providing only those service components that make sense. If real dynamic service deployment is desired, the proposed mechanism may be extended by using a virtual machine based approach as mentioned above.

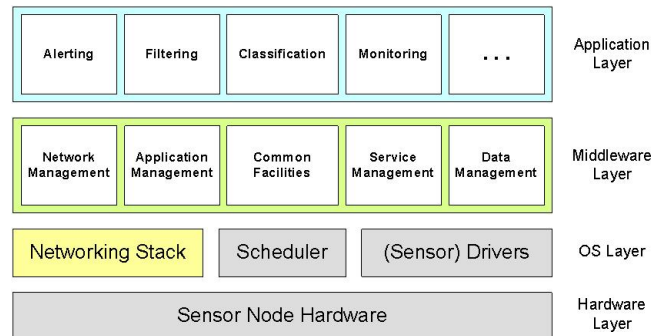


Fig. 5. Software architecture of SOCRADES WSN.

Figure 5 shows a layered architecture for the proposed static service-oriented approach:

- The Hardware Layer provides interfaces to the basic hardware components.
- The OS Layer consists of the networking stack, the scheduler and various sensor and hardware drivers.
- The Middleware Layer consists of a set of management components and a set of so-called Common Facilities. These facilities include mechanisms as timers, time synchronization, localization, sensor data handling (sensor reading, sensor calibration, sensor fusion) etc. As these mechanisms may be used by several application service components, it makes sense to provide them as part of the Middleware Layer. The management components are designed to provide external communication for application service components (Data Management), allow control and configuration of service components (Service Management), and provide means for configuring and controlling applications (Application Management) and the

networking stack (Network Management). In particular, the Network Management, Service Management and Data Management services provide access to local application services for other nodes.

- The Application Layer consists of application service components that perform application specific tasks like monitoring particular sensor data, classify and filter those data in order to identify interesting events or generate alerts if critical states appear. These components are built on top of the middleware service components (management components and common facilities); they are supposed not to perform any external communication directly but use the middleware components for those purposes. Thus, the application service components are only referencing interfaces of components that are residing on the same node whereas the middleware components are performing all kinds of inter-node communication.

The implementation of the proposed architecture must take into account that in TinyOS all software components must be merged into one monolithic application binary. This compilation and linking process does not produce binary modules that correspond to the components at source code level. In order to realize the respective middleware components, mechanisms that allow guaranteed delivery and control of the latency of messages have yet to be specified. Furthermore, service addressing schemes allowing transparent access to services based on the properties of the data they offer are required.

The WSN gateway that constitutes the interface of the sensor network towards other systems is the natural place to implement a full SoA approach due to the fact that it is externally powered and has sufficient resources both in terms of CPU capacity and memory space. The gateway represents all services provided by the sensor network. Service composition is done via the networking services of the sensor nodes as exemplified with in-network data aggregation. Deployment of new services is done via the gateway and must be accompanied by wireless deployment of new sensor node software.

6. CONCLUSION

Trying to apply a service-oriented approach to WSN in industrial applications we have found a couple of restrictions due to external constraints and due to the limited resources available in WSN. Although a standardized service interface at the application layer is desirable, it must take into account other requirements like energy efficiency or the above mentioned constraints. As the energy costs of communication are unlikely to drop significantly in the future, it finally depends on progress in battery capacity and energy harvesting efficiency whether SOA can be implemented to a higher degree in WSN. The currently best way to guarantee flexibility and energy efficiency is by providing appropriate tools for design and engineering as these steps always precede an implementation of modified or new functionality in industrial applications.

ACKNOWLEDGEMENTS

This work is supported by the SOCRADES project, a part of the Information Society Technologies (IST) initia-

tive of the European Union's 6th Framework Programme, under grant no. IST-5-034116. The authors acknowledge collaboration with Christian Hock, Thomas Grosch and Jens Makuth from Siemens AG, Industry, Nuremberg, Germany, with Lutz Rauchhaupt, Spiro Trikaliotis and Marko Kraetzig from ifak - Institute for Automation and Communication, Magdeburg, Germany, and with Francois Jammes and Harm Smit from Schneider Electric, Grenoble, France.

REFERENCES

- J. Blumenthal and D. Timmermann. Resource-aware service architecture for mobile services in wireless sensor networks. In *Proc. International Conference on Wireless and Mobile Communications ICWMC '06*, pages 34–34, 2006. doi: 10.1109/ICWMC.2006.80.
- D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. *ACM SIGPLAN Notices*, 38(5):1–11, 2003. URL <http://www.tinyos.net/>.
- Johannes Gehrke and Praveen Seshadri. Querying the physical world. *IEEE Personal Communications*, 7:10–15, 2000.
- F. Jammes and H. Smit. Service-oriented architectures for devices - the sirena view. In *Proc. 3rd IEEE International Conference on Industrial Informatics INDIN '05*, pages 140–147, 2005. doi: 10.1109/INDIN.2005.1560366.
- Jeffrey King, Raja Bose, Hen-I Yang, Steven Pickles, and Abdelsalam Helal. Atlas: A service-oriented sensor platform: Hardware and middleware to enable programmable pervasive spaces. In *Proc. 31st IEEE Conference on Local Computer Networks*, pages 630–638, 2006. doi: 10.1109/LCN.2006.322026.
- M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits. Oasis: A programming framework for service-oriented sensor networks. In *Proc. 2nd International Conference on Communication Systems Software and Middleware COMSWARE 2007*, pages 1–8, 2007. doi: 10.1109/COMSWA.2007.382431.
- Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005. ISSN 0362-5915. doi: <http://doi.acm.org/10.1145/1061318.1061322>.
- Thomas Schott. Global megatrends and their effects on the production of the future. In *Conference Proceedings, 5th International Conference on Industrial Informatics, July 23-27 2007, Vienna, Austria*, volume 1, page 18, 2007. URL http://www.indin2007.org/keynote_schott.php.
- Yong Yao and Johannes Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002. ISSN 0163-5808. doi: <http://doi.acm.org/10.1145/601858.601861>.