Driving Forces behind Middleware Concepts for Wireless Sensor Networks

Kirsten Terfloth and Jochen Schiller
Freie Universität Berlin
Takustr. 9
14195 Berlin
+49 (0) 30 838 75211
{terfloth, schiller}@inf.fu-berlin.de

ABSTRACT
In the first days of the emergence of Wireless Sensor Networks (WSN) programming software involved expertise in both hardware and networking. Since then, various middleware architectures have been proposed to achieve a suitable abstraction from distribution and management tasks of sensor devices. This allows users to focus on the application development.
The approaches suggested so far differ in concept, functionality and envisioned abstraction. This paper surveys some representative approaches and states that only three different flavors of middleware exist, each of them addressing some characteristic abstraction middleware architectures are eager to provide. For future middleware implementations it will be beneficial to carefully balance the components of each class to obtain a mature design.

Keywords
Wireless Sensor Networks, middleware, classification.

1. INTRODUCTION
Wireless Sensor Networks (WSN) have recently received a lot of attention within the research community since they demand for new solutions in distributed networking. A common scenario associated with these networks is that tiny nodes, equipped with several sensors and hardware for wireless communication, are deployed randomly and in large numbers within a certain area. In order to report the data they gather in their proximity to an interested application or user, nodes connect to their neighbors and send valuable information on a multi-hop path to its destination.
New questions arise from the architecture and purposes of the network as well as from the embedded nature of the sensor nodes. The resources of single nodes, especially in terms of energy and memory, are very limited, thus system-oriented issues cannot be ignored in the software development process. Failure of nodes may be common if we assume the network to consist of huge numbers of cheap, error-prone nodes that, once deployed, shall never be touched again. The devices may simply run out of energy, loose connectivity due to shadowing or death of surrounding nodes.
Unlike other ad-hoc networks, nodes participating do not have their own objectives for being a member but rather serve a global effort or task introduced by a user. This leads to a shift from a local view of a node, towards a more sophisticated, data-centric understanding of the network as a global component. An application programmer wants to treat the network as an entity, to get information about certain data streams or events, and not to be concerned with single nodes. This perspective introduces new design and optimization goals: Prolonging lifetime and connectivity of the whole network becomes crucial while fairness or motivation issues do not have to be taken into account.
Also, sensor nodes deployed close to one another are likely to sense similar values within the same time interval. This spatial and temporal correlation can be exploited to optimize the network behavior.
To be able to provide a high-level interface to programmers, software is needed in between the system-oriented sensor node and the application. The following sections analyze and discuss the nature of such a middleware abstraction layer.

2. MOTIVATION AND BACKGROUND
Striving for a middleware layer in the domain of restricted devices like sensor nodes may seem inadequate at first. Aiming at the most efficient software possible, draining the last bits of performance and energy-savings seems to interfere with an architecture that allows customers with possibly limited programming skills to employ and use wireless sensor networks.
But while middleware certainly introduces some overhead compared to tailored software, a valuable abstraction can encapsulate functionalities very different applications depend on. This effectively reduces memory consumption due to reusability of components and also simplifies application development.
Looking at a functional level of a middleware, two areas of problems have to be dealt with properly: the embedded nature of the networked sensors and the distribution of a large amount of nodes. The different strategies to what extend these challenges are met within current approaches are discussed in Section 3.
On a conceptual level, middleware offering a well-known programming paradigm to the programmer is advantageous, since it establishes a specific view on the network. Users are able to use familiar structures, programming styles and patterns to implement the tasks they have in mind.
The challenges for middleware design have already been studied with focus on programming paradigms and general design principles. Römer [8] identifies several requirements that have to be fulfilled to efficiently support applications for WSN. Furthermore, he examines different programming paradigms used in existing middleware abstractions and discusses their advantages and disadvantages. Krishnamachari et al. [4] go a step further and develop a two-layer-architecture depending on their analysis. They combine a cluster layer to organize groups of nodes and a resource management layer, which is in charge of allocation and adaptation of resources depending on the current

state of the network. A definition of a set of common services a middleware should provide, including a standardized service interface to support several applications with possibilities to specify the quality of service needed, is suggested. As has been proposed in [1], they also emphasize on the importance of a lightweight and data-centric design which relies on localized algorithms.

When looking at existing middleware approaches for WSN, some interesting similarities can be found throughout the different programming paradigms and approaches. The major contribution of this paper is to identify and give an understanding of the driving forces in this area, and to point out the strengths of the different concepts applied.

3. REPRESENTATIVE APPROACHES

The following examples have been carefully selected from a huge pool of existent approaches to give an overview of the spectrum of concepts as well as functions existing platforms and suggested architectures for middleware supply. All projects are introduced in a short manner with focus on the services they feature from an application programmers' perspective.

3.1 Hood

Whitehouse et al. propose a middleware architecture called Hood [11] that provides an interface to a subset of the sensor nodes, called a neighborhood. Based on criteria for choosing neighbors and a set of attributes to be shared, a user can specify different kinds of neighborhoods. Hence a neighborhood may e.g. be formed by those nodes within a one-hop distance that are able to provide temperature readings. Hood basically handles any management issues arising, like supervision of neighborhood lists, data caching and sharing among nodes and the definition of messaging protocols.

Communication within a neighborhood is based on a broadcast/filter mechanism. If a node wants to share one of its attributes, it simply broadcasts the value. Incoming packets are filtered, and nodes can determine whether or not the received attribute is of interest and cache it. There is no feedback to the node that sent the value, so in contrast to concepts building upon reliable networking, Hood only needs asymmetric links between nodes.

Looking at the programming part of the approach, a neighborhood becomes a programming primitive. To create a new neighborhood and allow its individual parameterization, a code generation tool has to be invoked by the developer. The system itself builds upon TinyOS [3], an operating system widely deployed on sensor nodes. Several interfaces offered by Hood provide handles to access neighborhood attributes and define sharing and updating strategies. Furthermore, values of neighbors stored locally on a node can be annotated with so called scribbles. These simply note extra information, for example the quality of the link to the mirrored node

Overall, a programmer who builds applications upon Hood is given the possibility to address and control functional parts of the network together, instead of issuing single nodes. Thus, this project alleviates effort for maintenance and low-level concerns, and takes the burden of dealing with distribution from the programmer.

3.2 Maté

Reprogramming of nodes usually involves flashing a complete binary image. If this is done when the network has already been deployed, the complete software stack, which includes firmware, network protocols, middleware and application, will need to be distributed on a multi-hop path to every single node. Dependent on the size of the image, this procedure consumes a tremendous amount of energy, a resource that most of the times cannot be re-charged after node deployment. To tackle this problem Lewis et al [5] have developed a virtual machine, Maté, and a specific byte-code it can interpret. This way, software updates may solely implicate transmission of the new application.

Using Maté on sensor nodes presumes that applications have to be expressed in special Maté instructions. The design of a suit-able language is therefore crucial for the ability to express applications the network is supposed to accomplish, and thus for the success of the virtual machine. Mandatory goals include the need for concise instructions and dense byte-code to save energy on transmission, and an expressive but simple language to enable the envisioned wide spectrum of possible applications. The authors decided on a stack-based architecture and an instruction set programming style. Since Maté is highly dependent on TinyOS and makes use of its messaging infrastructure, the complete system architecture is optimized for a symbiotic relationship. The sizes of Maté instructions are for example customized to perfectly fit into TinyOS packages. Programs can thus be segmented into equal sized chunks, so called capsules, which is advantageous for on-the-fly software installations.

The instruction set of Maté combines low and high-level instructions, and allows three possible operand types to be used: values, sensor readings and messages. Besides basic instructions for arithmetic computations, halting and branches, sensor network specific commands are available and offer a convenient abstraction for an application programmer. A build-in routing algorithm can e.g. be called by issuing a single instruction, which is in charge of sending the specified package to its destination. Also, packets can forward themselves to install new applications in the network within a single instruction call. Furthermore, the instruction set includes eight instructions that may be implemented by the application programmer, and is thus tailored to the needs of a special application scenario.

A safe execution environment as provided by a virtual machine hides the complexity of the hardware or, in this case, TinyOS's complex, asynchronous execution model, and prevents system crashes. The instruction set design is especially targeted to the sensor network domain.

3.3 Generic Role Assignment

An approach to offer a configuration environment for nodes in WSN is suggested by Römer et al. [9] with the General Role Assignment (GRA) project. The large scale of a wireless sensor network lets configuration issues get intricate after deployment of nodes. A direct interaction between users and single nodes is not viable, which leads to the fact that automation is necessary in this context. Nodes will have to evaluate their own status within the network, compare it to surrounding nodes and then tune their behavior themselves.

To accomplish such self-organizing networks, the nodes agree on specific functions or roles each of them will take within the network. GRA acts as a framework that organizes the distribution, communication and evaluation issues involved in role constitution. Four key elements have to be defined to support such actions: First of all, characteristic properties of individual nodes have to be accessible to determine the state of an individual node.

These attributes may include static hardware elements like available sensors and their resolution, as well as dynamic features like remaining battery charge or eventually physical location. Any element of interest is maintained in a property directory on each node in a name-value list, and can be questioned using a provided interface.
Based on this directory, a programmer can depict roles for nodes to implement the envisioned task of the network. A role specification states the behavior of a node within its network context with regard to its local attributes and network properties. Each role is associated with an identifier and a set of conditions or rules that have to be met to assign it to a node. This evaluation, or role assignment algorithm, usually not only depends on the attributes of one, but on a group of spatially neighboring nodes, and thus builds upon distributed, localized interaction. The authors presume a similar software state of all participating nodes. Any change of condition may result in re-evaluation of roles, and therefore starts a reassignment process.
The last element this approach requires is a set of basic services. They encapsulate diverse functions the role assignment process or nodes can call at runtime. Routing, time management or other common features will be part of this library.
Application programmers are offered a new way of specifying the task a network has to accomplish. Instead of writing one application, a set of roles can be implemented to grasp the behavior of the network. The role specification is a configuration tool, once again helping a user to abstract from distributed management concerns.

3.4 TinyDB

TinyDB [7], among other distributed database approaches [2], is a very popular suggestion to alleviate network programming from application programmers. Although it has been discussed in depth before [8], it will be presented here briefly since it incorporates many interesting features. The idea of a middleware using a database abstraction is to enable the utilization of a well-known, declarative programming language upon the distributed nodes without having to deal with network issues. Therefore, the network established by the sensor nodes is understood as a distributed database which can be queried using a subset of SQL. The authors added some essential key-words specific to the sensor network domain to enhance the language respectively.
In this concept, each node contributes one row to a single, virtual table, and each column represents one of the attributes that can be queried. A query processor is run on every node to handle and possibly aggregate the sensor data questioned by the query specification. Thus to obtain values from the network, a user issues a query, which is then automatically routed to all nodes. TinyDB maintains a spanning tree from the node where the request has been initialized, so that resulting data can be sent back in reverse direction. Queries may be marked to be evaluated periodically, or values to be aggregated, summed up or grouped on their way back through the network. Any maintenance concerning bootstrapping or failure of nodes or routing issues will be handled by TinyDB without any interaction with the programmer.
TinyDB contributes with its SQL-style programming manner an interface that is already widely accepted. Since distribution issues are transparent to the user, even unexperienced users are able to task the network appropriately.

3.5 Impala

The last middleware component introduced here is Impala [4], an architecture implemented within the ZebraNet project [6]. The primary design goal has been to build a modular, lightweight run-time environment for applications that manages both devices and occurring events. Hence, Impala splits the field of duty into two layers, one to encapsulate the application protocols and programs for ZebraNet, and an underlying layer that contains functions for application updates, adaptation and event filtering.
Application programming follows an event-based programming paradigm, thus any application deployed upon the nodes has to implement a set of event- and data-handlers to respond to different types of events, including timer, packet, data and device events. Besides supplying event filter mechanisms, Impala emphasizes the need for integrating adaptation and updates of applications at runtime within the system architecture.
Adaptation of an application or an application-level protocol can become necessary due to changes of the system, e.g. failure of certain sensors or low battery level, as well as application specific modifications, e.g. a sudden drop of successfully delivered packages. A middleware agent, the Application Adapter, checks the overall state of the system on a regular basis and selects the most suitable configuration according to the present circumstances. Dynamic software updates may be mandatory during execution, but since the devices are not re-programmable at the same time (ZebraNet equips wildlife animals with sensor nodes, which results in a highly dynamic topology, so software can be received in incomplete bundles of packets) the Application Updater serves as a management component for available versions and code bundles. In contrast to Maté, Impala does not use a special instruction set and byte-code, but supplies compiled binaries that are linked whenever a complete new version is available. The proposed approach can be interpreted as a combination of operating system functionalities, a resource manager, a configuration tool and an event filter. Programs depending on Impala have a clear interface to a set of events. The middleware offers a management framework for applications beyond operating system issues.

4. CLASSIFICATION

All of the approaches presented in the last section can be categorized on behalf of the focus they are taking. We identify three different conceptual groups of middleware implementations, with each contributing a solution to one substantial problem for application development in WSNs. The scope of this classification is certainly not restricted to the examples, but holds true for almost all middleware architectures present. The affiliation to one group is neither exclusive nor discrete, so an approach may feature some elements of one group and totally incorporate attributes of another. Table 1 provides a subsumption of the approaches discussed into the recommended classes.

4.1 Group Abstraction

Several research teams have created or suggested middleware implementations that organize nodes of a network into groups sharing certain characteristics. The most common attribute to qualify nodes for membership in a subset of the network is network proximity. Clusters may for example include all nodes reachable in a two-hop neighborhood. But although proximity is one essential criterion, it fails to aid an application in search of sensor data of a specific kind. In this case, it will be more

valuable to use the sensor hardware to come to a decision on a group infrastructure. Any other characteristics may be applied as well, leading to a very widely applicable tool for node organization.

The group abstraction tackles those problems that arise for tasks due to the scale of a network and the distribution of nodes. Sensor networks are envisioned to contain nodes in the magnitude of hundreds or maybe thousands, thus there is no efficient way to obtain control manually. Rather, the network itself has to be able to cope with common issues that arise from using small, cheap and thus error-prone hardware. Likewise, lifetime of a network depends on the behavior of single node, and cooperation strategies are certainly indispensable to maximize it. The keyword commonly used in this context is localized algorithms, meaning that the state of the network has to be controlled by using algorithms that do not span over the whole network, but are executed within a defined spread of nodes, a group.

Other scenarios the group abstraction provides powerful tools for are setups that feature heterogenic hardware. For example nodes may not all be equipped with the same sensors, the same amount of memory or energy. In the Scatterweb network [10] some nodes posses a solar cell, and in case of sunshine these nodes are preferably chosen to route information. A middleware should be able to react to such environmental changes and man-age to configure the system accordingly.

Hood and GRA are two projects that explicitly provide a group abstraction. While Hood servers as a management entity for data sharing and networking issues, GRA emphasizes upon configuration of the network and leaves basic services, e.g. management of software updates or neighborhood discovery, up to an underlying software layer. Groups are built by nodes acting similar when executing the same role. Opposed to this, Hood implements an interface for applications to use the specified attributes of different groups.

The only contribution to group abstraction Maté fosters is the incorporation of TinyOS's Active Messages and their capability to address logically separate sensor networks. Impala does not recognize the need for grouping of sensors and assumes these functionalities to be implemented within the application software.

TinyDB supports the SQL statement GROUP BY. This statement is misleading, if we assume TinyDB will build a somehow clustered infrastructure when sending a query that contains this keyword. Instead, TinyDB maintains a global spanning tree for queries, and floods the request to all nodes. The result of a query will be presented in a structured manner, grouped by the chosen attribute, but no optimization with regard to networking issues is involved. Grouping is therefore possible at application level, but not implemented at network level.

4.2 Virtual Machine Abstraction

Middleware approaches that employ a virtual machine (VM) upon sensor nodes aim at improving the ease of use for the application programmer. More generally speaking, a virtual machine addresses the complexity originating from the underlying software, which can be within the range of pure firmware up to a complete operating system. In both circumstances a VM serves as a safe execution environment for application code, thus providing means to prohibit an application from crashing a node completely.

The impact of the chosen language interpreted on each device is important: The more sophisticated the language is, the more experienced a user usually has to be to program a task. On the other hand, if the VM supports a script language, application development will be a lot easier for non-experts, but may also lack the ability to express essential functionality, and may therefore in the worst case become obsolete. Besides this tradeoff, it also has to be taken into account whether to support a language already established, which is advantageous for rapid application development, or to provide a domain-specific solution. Due to the encapsulation of basic functionalities within the language and definition of WSN specific data types, applications may be expressed in a more natural and concise way in the latter case.

The developers of Maté provide a language that already holds commands for sending and receiving packets and a default routing algorithm. These high-level commands take the responsibility from the application programmer to specify the correct header information and wrap the data being sent in packages. Furthermore, values can be typed as sensor readings, supporting an intuitive way to handle data. But while the Maté instruction set offers a programming tool that partially abstracts from network management issues, the language itself requires solid knowledge of assembler programming and stack architectures. Inexperienced users are probably not the intended audience for Maté.

TinyDB also incorporates a VM abstraction in its implementation. In contrast to Maté, the language has been chosen with regard to rapid development goals. SQL, enhanced by domain-specific keywords, provides a fair and rather easy to use programming abstraction to deal with data collection within WSNs. At the downside, any user-defined optimization that goes beyond this scope, for example querying only a selected part of the network, cannot be expressed within the language.

The concept of Impala explicitly rejects a VM abstraction in favor of a runtime environment that supports update and adaptation management. The authors motivate their choice with the expected infrequent software updates and argue that compilation and linking will fit their envisioned target application better.

To specify the conditions a node has to satisfy for choosing a role, the developers of GRA suggest a framework that evaluates rules, Boolean expressions, eventually containing predicates over nodes properties. This concept is rather easy to understand and, since functions can be called by a rule, an expressive means to program. Whether the rules are really interpreted upon the nodes, or precompiled and linked is left up to the implementation.

Hood itself does not provide a runtime environment of any kind, but serves as a functional layer in between the operating system and the application.

4.3 Modular Service Architecture

Classical middleware implementations like CORBA or DCOM share the goal to hide complexity due to heterogeneity, distribution or communication from applications. The common way to implement an infrastructure supporting this goal is to define a set of basic services, implement them and offer a standardized interface, applications are free to invoke. Furthermore, a modular design is advantageous: As long as interfaces are being kept the same, the implementation of services can be exchanged without touching any software relying on the middleware.

This design is favorable, since it allows easier adaptation of software components than a monolithic design. Other than the group and VM abstraction, approaches of this category do not provide a solution to an inherent problem of the WSN domain, but rather emphasize the software development process. If modularity

can be obtained in a way that even parts of the middleware are exchangeable at runtime, this will of course be beneficial with regard to energy-consumption and adaptation.

When discussing services and a middleware layer, the question what service should be incorporated within this layer arises, and whether a general middleware abstraction layer exists. The examples above demonstrate that this question is still a topic of active research, if not even superfluous due to their application driven nature.

TinyDB supplies a framework that implements a default routing strategy as well as node discovery, scheduling and power management support. The newest version allows the user to integrate new routing and aggregation algorithms, thus the design has been changed towards a modularized architecture since its first release.

The virtual machine Maté serves as a runtime environment for applications, but also implements basic routing and software update strategies that can be invoked. In case a programmer identifies the need for more basic services, Maté offers eight instructions that may encapsulate their implementation. Services are mapped onto the instruction set level.

GRA can be interpreted as a single service offered to the application layer: the control of network structure by assignment of roles, thus behavior of nodes, within their network context. Since routing and management issues are left to an underlying service layer, these have to be incorporated on demand.

Services in Impala include application scheduling, adaptation and linking of software updates and device management, classical operating systems functionalities but also middleware features with its event-handling notion. To enable on-the-fly reconfigurations and linking of code at runtime, the design has to be modular, otherwise changes at runtime would not be possible.

Hood allows a user to specify and alter any TinyOS components and modules generated. The basic service it provides for programmers is an infrastructure handling data sharing and caching issues with regard to membership of nodes to groups.

As can be derived from the examples above, there is no clear perception of the nature of middleware services. They may range from very high-level support to basic operating system functions.

Table 1: Middleware approaches and their affiliation to the three categories group abstraction, virtual machine abstraction and service-oriented middleware.

Approach	Group	VM	MSO
Hood	++	0	0
Maté	0	++	+
Impala	0	0	++
GRA	+	+/++	n/a
TinyDB	0/+	++	+

5. CONCLUSION

In this paper we discussed five middleware architectures for Wireless Sensor Networks. Although they are very different in terms of provided services and interface, programming paradigm or architecture, they all share some conceptual views upon WSNs. We believe, that approaches designed to tackle common problems in programming for this application domain usually make use of at least one of the following techniques: Abstraction by providing a way of grouping nodes, abstraction by providing a virtual machine or usage of a modular, service-oriented architecture. While the first two abstractions deal with the distribution and embedded nature of the sensor network devices, the third concept contributes to the software development process. Evaluation of possible means to implement features of all three concepts may help in the development process of future middleware approaches.

6. REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, Wireless Sensor Networks: A Survey. In *Computer Networks*, 38(4): 393-422, March 2002.

[2] P. Bonnet, J. E. Gehrke, and P. Seshadri. Querying the Physical World. In *IEEE Personal Communications*, 7(5):10–15, 2000.

[3] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Programming Language Design and Implementation (PLDI)*, June 2003.

[4] P. Juang, H. Oki, Y. Wang et al. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS-X*, San Jose, USA, October 2002.

[5] P. Lewis and D. Culler. Maté: A Tiny Virtual Machine for Sensor Networks. In *ASPLOS-X*, San Jose, USA, October 2002.

[6] T. Liu and M. Martonosi. Impala: A Middleware System for Managing Autonomic Parallel Sensor Systems. In *ACM SIGPLAN*, San Diego, USA, June 2003.

[7] S. R. Madden, M. J. Franklin, J. M. Hellerstein and W. Hong. TAG: a Tiny Aggregation Service for Ad-Hoc Sensor Networks. In *OSDI 2002*, Boston, USA, December 2002.

[8] K. Römer. Programming Paradigms and Middleware for Sensor Networks, *GI/ITG Workshop on Sensor Networks*, pp. 49-54, Karlsruhe, Germany, February 2004.

[9] K. Römer, C. Frank P. M. Marron and C. Becker. Generic Role Assignment for Wireless Sensor Networks. In *ACM SIGOPS European Workshop 2004*, Leuven, Belgium, 2004.

[10] J. Schiller, A. Liers, H. Ritter, R. Winter and T. Voigt. ScatterWeb - Low Power Sensor Nodes and Energy Aware Routing. In *HICSS 2005*, Big Island, January 2005.

[11] K. Whitehouse, C. Sharp, E. Brewer, and D. Culler. Hood: A Neighborhood Abstraction for Sensor Networks. In *ACM MobiSys 2004*, Boston, USA, June 2004.

[12] Y. Yu, B. Krishnamachari, and V.K. Prasanna. Issues in Designing Middleware for Wireless Sensor Networks. In *IEEE Network Magazin*, 2003.