Aus dem Institut für Informationssysteme
der Universität zu Lübeck
Direktor:
Prof. Dr. rer. nat. Volker Linnemann

Efficient XML Data Management and Query Evaluation in Wireless Sensor Networks

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck
- Aus der Sektion für Informatik/Technik und Naturwissenschaften -

vorgelegt von
Nils Höller
aus Zell am See, Österreich

Lübeck, Oktober 2010

Nils Höller
Institut für Informationssysteme
Universität zu Lübeck
Ratzeburger Allee 160
D-23538 Lübeck
E-Mail: hoeller@ifis.uni-luebeck.de

Dissertation zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften (Dr.-Ing.)
der Sektion für Informatik/Technik und Naturwissenschaften
der Universität zu Lübeck

Vorsitzender der Sektion:	Prof. Dr. Jürgen Prestin
Vorsitzender des Prüfungsausschusses:	Prof. Dr. Stefan Fischer
1. Berichterstatter:	Prof. Dr. Volker Linnemann
2. Berichterstatter:	Prof. Dr. Thilo Pionteck

Tag der mündlichen Prüfung: 07.12.2010
Zum Druck genehmigt. Lübeck, den 07.04.2011

Abstract

With the advancing development in the field of microprocessor technology tiny wireless computing devices for sensory tasks denoted as wireless sensor nodes have been introduced. Large scale networks of these devices, denoted as wireless sensor networks, are deployed for long term location monitoring and typically collect large sets of sensor data over months or years. Managing sensor data in networks consisting of hardware resource limited sensor node devices brings up many challenges for database research. Moreover, integrating wireless sensor networks in heterogeneous networks is a complex task. A reason is the absence of a standardized data exchange format that is supported in all participating sub networks.

In the last decade, XML has become the defacto standard for data exchange in the World Wide Web (WWW). The positive benefits of data exchangeability to support system and software heterogeneity on application level and easy WWW integration make XML an ideal data format for many other application and network scenarios like wireless sensor networks. Moreover, the usage of XML encourages using standardized techniques like SOAP to adapt the service oriented paradigm to sensor network engineering. Nevertheless, integrating XML usage in wireless sensor network data management is limited by the low hardware resources that require efficient XML data management strategies suitable to bridge the general resource gap.

This dissertation introduces approaches for integrating efficient XML usage in wireless sensor networks. This includes the integration of XML in the engineering process, energy and memory efficient data management strategies, efficient solutions for XML data acquisition and general optimization strategies for handling XML queries and result messages in large scale sensor networks.

In detail, this work introduces the programming framework $XOBE_{SensorNetwork}$ which provides the direct use of XML in a sensor network programming language while ensuring stable and space, time and energy efficient programs handling XML data. To allow flexible XML data management on sensor node devices with strict hardware resource limitations, $XOBE_{SensorNetwork}$ includes two separate strategies on integrating compressed XML data management in wireless sensor networks that both have been implemented and are running

on today's sensor node platforms. In this dissertation, it is shown how this compressed XML data can be further processed and how XPath queries can be evaluated dynamically. In an extended evaluation we compare the performance of both strategies concerning the memory and energy efficiency and show that both solutions have application domains and are fully applicable on today's sensor node products.

In summary, $XOBE_{SensorNetwork}$ offers a complete XML solution for wireless sensor networks from application engineering to in-field data management. As part of the DFG project AESOP's TALE, the presented XML data management solutions are the future basis for integrating SOAP support in wireless sensor networks.

While the previous aspects cover the field of data management and application engineering, this dissertation also includes further optimizations in the field of communication. Generally, saving energy in wireless sensor networks is essential to extend the lifetime of in-field deployments. Previous research has shown that communication is generally the most energy consuming task and needs to be reduced in order to build resource-efficient long-term applications. This dissertation therefore additionally introduces optimizations for processing high amounts of unique queries by using a **d**ynamic **a**pproximative **c**aching **s**cheme: **DACS**. In DACS, query results can be retrieved from caches that are placed nearer to the query source instead of sending queries deep into the network. The communication demand can be significantly reduced and the entire network lifetime is extended. To verify cache coherence in sensor networks with non-reliable communication channels, an approximative update policy is used. To localize the adequate cache adaptively, model-driven queries including a degree of demanded result quality can be defined. The entire logic is thereby processed by DACS and hidden to the user. The significant energy conservation is proven in evaluations that include real sensor node deployments.

Acknowledgements

I would like to thank the entire IFIS staff for their support over the last years of finishing this thesis and for the great exam celebration: Angela König, Jana Neumann, Nils Fussgänger, Sven Groppe, Christoph Reinke and Prof. Volker Linnemann. Thanks to Dirk Kukulenz for introducing me to the field of database systems research and showing me how to write interesting papers. I especially thank my advisor Prof. Volker Linnemann for giving me the opportunity of working and researching at the Institute of Information Systems, helping me with his research experience, supporting all of my plans, conference trips and ideas and giving me advise for my future career. Also special thanks go to my colleague and friend Christoph for working as an unbeatable team over nearly 10 years in university and sharing so many unforgettable memories. I thank the exam committee Prof. Thilo Pionteck and Prof. Stefan Fischer for reviewing my work and chairing my exam.
During the last years I worked on the project AESOP's TALE of the Institute of Information Systems and the Institute of Telematics. Many fruitful discussions have influenced my work and gave me new ideas. I therefore thank all ITM members of the AESOP's TALE team for their cooperation: Prof. Stefan Fischer, Nils Glombitza, Martin Lipphardt and Christian Werner. I also greatly appreciate the help of many student workers that worked for me and the AESOP's TALE team over the last years. I hereby especially thank (in alphabetic order): Daniel Böckmann, Julia Bulygina, Florian Frischat, David Gregorczyk, Dirk and Lutz Hansen, Fabian Kausche, Claudia Möller, Björn Schütt and Stefan Werner.

Lastly, I deeply thank my family for their support. I thank my mother Ilka and my father Friedrich for bringing me up with love and care, for always teaching me that education is a precious good and not to forget for supporting me financially and mentally over all of the years. I thank my sister Ann-Kathrin for being the best sister I can imagine. I thank my German and Austrian grandparents for always supporting me, being proud of me and providing me a peaceful childhood. Finally, I thank Karo for being always beside me and guiding me how to be successful.

Lübeck, March 2011 Nils Höller

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

With the rapidly advancing development in the field of microprocessor technology, which results into smaller and more powerful microprocessors, networked sensing systems became an increasing research topic throughout many fields of computer science in the last years. Large scale wireless sensor networks consist of single sensor nodes that combine regular computing devices with different sensors for monitoring environmental conditions and events. Unlike traditional sensor deployments, the new kind of resource constrained tiny sensor nodes/motes are deployed to not only sample and analyze real world processes but also further process the sensor data in-network, e.g. filter, analyze, combine and share sensor data. Typical deployment scenarios range from geological environment monitoring to ubiquitous applications [108, 157, 246]. The general hardware limitations, e.g. low energy and memory capacity and limited computational power, open up new challenges for data management research in order to find an efficient solution for sensor network data management and retrieval. The following section gives a detailed motivation for and description of the application scenario and strategies of this thesis.

1.1 Motivation

The evolving techniques of microprocessor and communication technology led to a new form of highly distributed, wireless large scale networks consisting of tiny sensory processing units denoted as wireless sensor networks (WSNs). These networks further coin the terms of *Ubiquitous and Pervasive Computing* which have evolved to autonomous research fields in computer science. The sensor nodes combine the traditional sensory tasks and further process the data in-network. Hereby, they are limited by the hardware resources of their components, e.g. the microcontroller, memory and radio unit.
In the recent years, the processing power of integrated circuits like central processing units (CPUs) could be significantly increased. The transistor density was hereby nearly doubled every two years following the prediction of Gordon E. Moore from 1965 [167]. On the other hand, the latest microcontrollers used for platforms like sensor nodes have been de-

veloped with the main goal of significantly reducing the size and the energy consumption to allow long term autonomous deployments. As a result, the processing capabilities of today's sensor nodes are not comparable to existing workstation CPUs or graphic processors. The memory configuration of the sensor nodes is also highly limited in comparison to workstations and other mobile devices, e.g. smartphones, due to the designated small form-factor of the nodes, which leaves no space for high memory configurations.
Beside the processing and memory infrastructure, the most important hardware limitation is the energy supply of the sensor nodes. Typically, sensor nodes are equiped with non-rechargeable energy supplies that limit the lifetime of in-field deployments like described in [108, 157, 246]. The lifetime of each single sensor node is determined by its battery. The network's application lifetime can be even shorter due to failing communication bridges and network separation. Hence, a main research goal in the field of sensor network research is efficient energy conservation. The most energy consuming task and hence most serious limitation in sensor networks is communication. Early results have shown that the communication energy demand is significantly higher than the energy demand for processing tasks, e.g. sending 1 byte via radio consumes the same energy as up to 1000 processing cycles [156]. Reducing the in-network communication demand and pushing data management processing steps deep into the network is therefore one of the main research goals in many fields of sensor network data management.

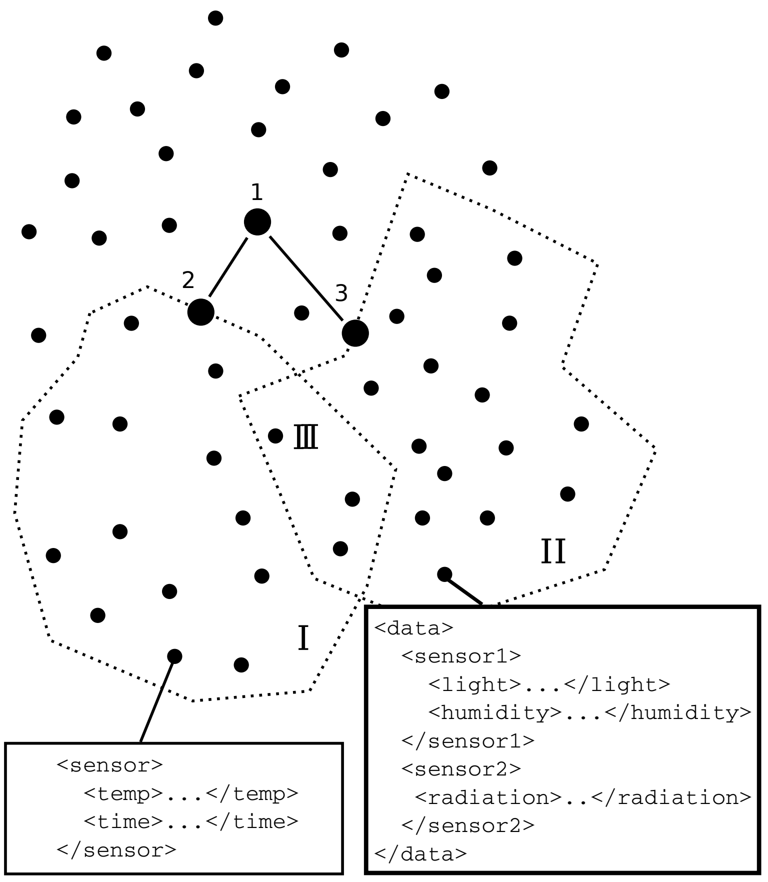
In summary, the mentioned general hardware restrictions of today's sensor node platforms bring up new challenges especially for complex data management in WSNs, e.g. processing large sets of structured and unstructured sensor data in the network and providing simple data acquisition solutions to acquire this information from outside the network. Today's complex data management techniques are not directly applicable without further adaptation to the constraints. Moreover, sensor network programming is a highly complex and error-prone task. The absence of supporting development frameworks like for traditional platforms and a development abstraction layer that requires working close to the hardware layer result in a tedious engineering process, especially when complex data has to be handled during application engineering. Finding simplified development solutions that also take care of data integration into the engineering process is crucial to improve the usability of sensor networks for non expert users, e.g. researchers from other domains than computer science.

Previous work in WSN data management has been focused on supporting simple relational data models like one table per network [156, 253]. In this thesis we discuss the need and dynamical usage of complex data structures like XML in WSNs. While the simple one table per network approach or other simple data models can reduce the exchangeability, complex data structures are required for handling large heterogeneous data sets efficiently. Generally, complex data models like XML enable using heterogeneous data and network components by providing an independent, well structured data model in the application

layer of the OSI Reference Model [257]. Hence, different types of sensors and systems can exchange their data and deployed networks can be extended continuously in-field. While managing more complex data structures on the sensor nodes is a challenge concerning the hardware and energy limitations, we present strategies that are especially designed to bridge these resource gaps. The results of this thesis show that complex data management is possible even on highly constrained sensor node platforms.
Furthermore, using XML is a key feature to support standardized protocols in the field of service oriented architectures. In detail, the project AESOP's TALE has been introduced to adapt the service oriented paradigm to sensor network application engineering to simplify the tedious and error prone task of traditional application development and open up application development to non experts. Additionally, to support the service developers, handling XML during the engineering process has been simplified by integrating native and statically type checked XML usage in the programming phase [96]. The following subsections clarify the motivation of this thesis by giving examples for the introduced topics.

1.1.1 XML Usage in Wireless Sensor Networks

As described before, in this thesis we suggest the usage of XML as a central data format in WSNs. From the data management perspective, the usage of XML supports interconnecting heterogeneous sensor networks and encourages the interchangeability of different types of WSNs with other systems by using a standardized data exchange format, e.g. making it easy to interconnect a sensor network to the WWW.
In Figure 1.1 we give a simple example for using XML on the application layer of the WSN to enhance the network heterogeneity. In detail, the example network includes three sub networks I, II, III, that differ in their provided data. This scenario is typical for sub networks that are deployed autonomously and do overlap because often a precise placement can not be guaranteed, e.g. deploying out of air over the target area. We assume a scenario where a WSN I has been deployed in the past. Now new sensor nodes with different sensor configurations need to be deployed, forming the sub network II and hence extending the entire WSN. As mentioned before, the positional placement of single nodes during deployment is often not precise. Hence, the two sub networks can overlap. While the overlapping sub networks may exist independently from each other, forming a third sub network III, that includes both data structures and thus has to support heterogeneous data on the application layer, can be an efficient option. By sharing and exchanging data between sub networks WSNs can be optimized e.g. concerning failure robustness and load balancing. Using XML enables the data exchange between these sub networks by supporting standardized transformation languages like XSLT [231] and standardized data query languages like XPath [234] and XQuery [235]. Furthermore, heterogeneous networks can be combined on higher level (e.g. sensor network interconnection) following the approach of the global sensor network (GSN) [2].



```
<sensor>
  <temp>...</temp>
  <time>...</time>
</sensor>
```

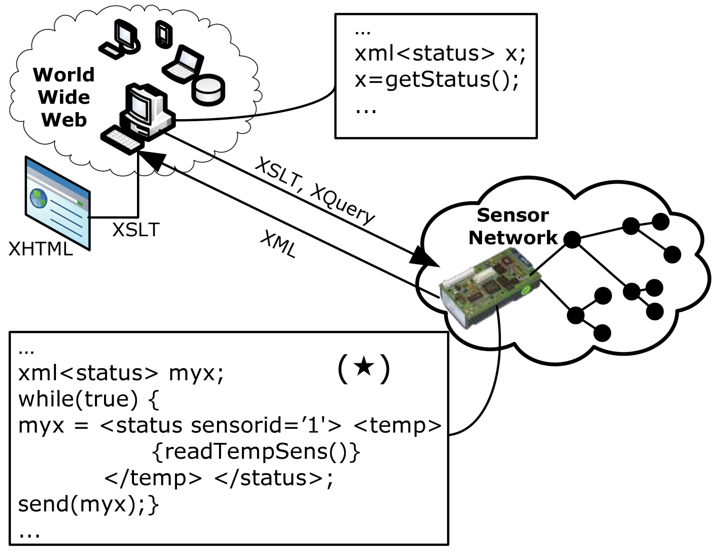
```
<data>
  <sensor1>
    <light>...</light>
    <humidity>...</humidity>
  </sensor1>
  <sensor2>
   <radiation>..</radiation>
  </sensor2>
</data>
```

Figure 1.1: Heterogeneous Data in Wireless Sensor Networks

The positive benefits of XML usage in WSNs for interconnecting the WSN to other networks like the WWW are shown in Figure 1.2. In this example, a sensor network can directly be queried using XML query languages like XPath [234] and XQuery [235] by any client that is able to process XML. The XML result can be further processed, e.g. in order to present it on a webpage using XSLT [231]. No explicit transformation step is needed which enhances the usability of the network and simplifies data acquisition for non expert users. The example further shows one of the main contributions of this thesis, as the major application features are programmed by directly using XML data within the target sensor network programming language which simplifies sensor network application engineering for the developers.

1.1.2 Adapting the Service Oriented Paradigm to Sensor Network Application Engineering

While the previous subsection introduced examples from the data management perspective, there are positive benefits from the engineering perspective. Using XML is a key feature to support standardized protocols in the field of service oriented architectures. The project



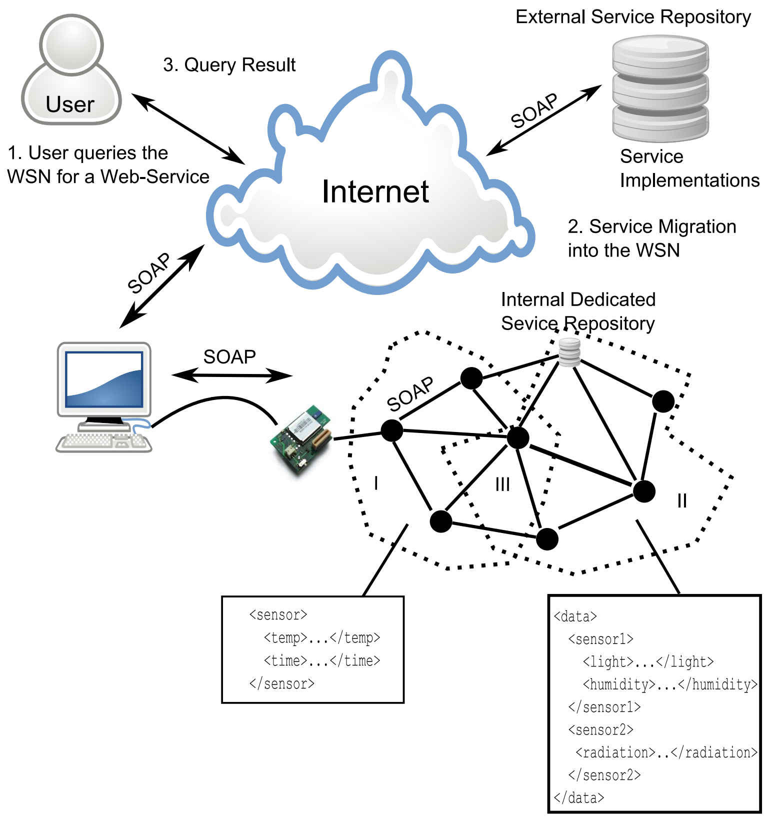
```
...
xml<status> x;
x=getStatus();
...
```

```
...
xml<status> myx;
while(true) {
myx = <status sensorid='1'> <temp>
            {readTempSens()}
        </temp> </status>;
send(myx);}
...
```

Figure 1.2: Sensor Network $< - >$ WWW Integration

AESOP's TALE focusses on adapting the service oriented paradigm to sensor network application engineering [149]. The complex, traditional application development process is simplified by introducing services that encapsulate sensor network functionalities and can be combined to create more powerful applications. This engineering process opens up application development to non-expert engineers. Additionally, to support the service developers, handling XML during the engineering process can be simplified by integrating native and statically type checked XML usage in the programming phase, as described in the previous example in Figure 1.2. In Figure 1.3 we give an overview on the project's system architecture. On the one hand we show the combination of service oriented aspects and on the other hand we present how XML simplifies the handling of heterogeneous data and enhances the interoperability of heterogeneous networks.

The key target of AESOP's TALE's system architecture is the user/developer that should be supported in developing applications by providing a standardized method of combining existing services to high level applications. Each service in the sensor network provides a basic functionality and is described by a unique service description. Additionally, external applications can consist of heterogeneous services, e.g. traditional web services outside the sensor network and services that are provided by the wireless sensor networks. To verify the exchangeability the project relies on standardized protocols, e.g. SOAP [232]. The support of such a protocol is a significant benefit of the XML support, which is the major contribution of this thesis.


External Service Repository
3. Query Result
User
SOAP
Service Implementations
1. User queries the WSN for a Web-Service
Internet
2. Service Migration into the WSN
SOAP
Internal Dedicated Sevice Repository
SOAP
SOAP
I
III
II

```
<sensor>
  <temp>...</temp>
  <time>...</time>
</sensor>
```

```
<data>
  <sensor1>
    <light>...</light>
    <humidity>...</humidity>
  </sensor1>
  <sensor2>
    <radiation>..</radiation>
  </sensor2>
</data>
```

Figure 1.3: AESOP's TALE System Architecture and Network Example

1.2 Goals and Organization of this Work

The main goal of this thesis is to integrate XML data management in the engineering process of wireless sensor network applications. Therefore, we discuss an extension of the sensor node programming language C that enables native XML usage within the program code. The extension is denoted as $XOBE_{SensorNetworks}$ and acts as a precompiler that transforms plain, type checked XML data [122] to data models that are compilable with the standard sensor node C compilers, e.g. gcc , ba-elf-gcc and avr-gcc [75].

To meet the general hardware restrictions of the nodes, e.g. low memory capacity, we introduce different approaches for compressed XML data models that can be processed dynamically, e.g. for evaluating XML queries in deployed sensor networks.

Beside the limitations of memory and processing power the main limitation of sensor nodes is the energy capacity. Communication has been shown to be one of the main energy consumers limiting the lifetime of the entire network. Using and transmitting complex data models like XML hence requires strategies for data delivery in the network. We therefore present approaches for optimizing the transmission of queries and results using XML data caches.

We summarize the goals and contributions of this work in particular:

- Integration of plain XML data in the Embedded C programming language and development of a precompiler to process the XML data.
- Development of XML data binding strategies for resource restricted sensor node devices and transformation of the embedded XML data in compressed data models and structures.
- Development of a sensor node runtime framework to process compressed XML data and XML queries on the compressed data dynamically.
- Introduction of communication strategies and query optimizations for XML sensor networks to extend the lifetime of deployed wireless sensor networks.

The contributions presented in this thesis have also been published in the following refereed publications [84, 85, 95, 96, 97, 98, 99, 100, 101, 102, 132, 133, 154, 171, 172, 198, 199, 200].

Structure of this Work

The remainder of this thesis is structured as follows:

In the next chapter, we introduce the basics that are required to understand the strategies and approaches of this thesis. In detail, the chapter gives a technical introduction into the field of wireless sensor networks and introduces the concepts of XML, XML query languages, XML data binding solutions and existing XML compression techniques. Furthermore, we describe fundamentals of service oriented architectures and the AESOP's Tale project as an application scenario for XML usage in wireless sensor networks.

In the following chapters, we discuss the contributions of this thesis in detail. Chapter 3 introduces the $XOBE_{SensorNetworks}$ precompiler that enables the integration of XML data in the WSN engineering process. We give a detailed overview on the precompiling cycle from parsing the embedded XML data to transforming the embedded XML data in compressed data models that are processable by the standard C compiler.

In Chapter 4, we discuss the transformation and XML data binding process in detail. We introduce the XML Template Compression Scheme that allows to significantly reduce the memory usage of XML data in order to enable dynamic XML data management on the sensor nodes. We hereby present two separate implementations of the XML Template Compression Scheme that are optimized for a high compression ratio (XTS) and a fast processing speed (XTO) respectively. We evaluate these implementations and compare them to existing applicable XML data binding solutions.

Data acquisition is one of the main tasks of wireless sensor networks. Hence, in Chapter 5 we give a detailed introduction on how to evaluate XML queries on compressed XML structures in wireless sensor networks dynamically. XPath evaluation algorithms for both XML Template Compression Scheme implementations are presented and the memory and energy efficiency of the evaluation process is compared for both solutions. We further discuss the dissemination and optimizations of XML queries.

After giving an overview on query optimizations in wireless sensor networks, we introduce a dynamic caching scheme for extending the lifetime of XML sensor networks in Chapter 6. We hereby discuss cache placement, cache coherence and cache localization in highly dynamic ad-hoc sensor networks. The caching approach is evaluated to show the significant energy efficiency when used for data acquisition. We finalize this thesis by summarizing the contributions and show possible fields of future work in Chapter 7.

Chapter 2

Basics and Related Work

In this chapter, we summarize related and previous work in the area of wireless sensor networks and especially in the area of integrating complex data management in wireless sensor networks by using XML. While the complex programming of sensor nodes requires efficient XML data binding solutions and the programming language integration of XML data in the sensor network engineering process, the general hardware constraints require optimized compression and processing techniques. Nevertheless, wireless sensor networks are deployed for data acquisition. Hence, the need of processing, updating and querying sensor network XML data dynamically is essential. We finally give references to possible application scenarios like data storage and communication in conjunction with service-oriented architectures in wireless sensor networks.

2.1 Wireless Sensor Networks

In the recent decades the computer science and technology has significantly evolved. In the early years, computers were mostly used for scientific, military and financial computation, controlled by a limited number of expert users following the usage paradigm of one computer for many users. With the development of the Home Personal Computer in the beginning of the 1980s which was favored by the advances in electric circuitry resulting into size reduced and affordable hardware the new paradigm of one computer per user became true. Still following Moore's Law, with the miniaturization and integration process in the recent years many computing devices like mobile phones and notebooks became part of the daily life, which finally initiated the era of many computers per user.

In his work *The Computer of the 21st Century* from 1991 [243], Marc Weiser forecasted this significant move from the one computer per user to the many computer per user era. In his work he coined the term of Ubiquitous Computing, denoting the integration of information processing devices and functionalities in everyday objects and activities. The process is also often denoted as Pervasive Computing. The integration process reaches to the point where the ubiquitous technologies disappear and are not longer distinguishable from the everyday objects [243]. This transition is currently in process and wireless sensor networks that are discussed in this thesis are one important trend in ubiquitous computing.

In detail, the significant miniaturization of electronic circuitry and new techniques in wireless communication and efficient, economic power management enable wireless, tiny, autonomous mobile devices denoted as sensor nodes / motes. These sensor nodes are typically equipped with diverse sensors to analyze the environment and build up autonomous ad-hoc wireless sensor networks. They contain computational hardware (e.g. microcontrollers), wireless communication hardware (e.g. radio controllers) and sensing devices (e.g. temperature sensors). As denoted in the introduction of this thesis, sensor networks are deployed for long term applications. Hence, the limitations in energy supply and memory configuration of the nodes bring up new challenges for different research areas, e.g. communication techniques, electrical engineering and information management. This thesis hereby covers strategies on integrating energy efficient XML data management in wireless sensor networks and optimizing the tedious task of sensor network application engineering.

For better understanding of the target application platform, we summarize the history of sensor node technology, research and development in this section. We further introduce the latest technologies and products in Section 2.1.3. We finally show the general hardware and development limitations of the platforms and define challenges for database research.

2.1.1 Historical Overview

The origin of today's wireless sensor networks is the military surveillance. One of first named projects is the Sound Surveillance System (SOSUS) [77], which has been introduced and used by the United States Navy during the cold war for tracking soviet submarines at different strategic locations around the world. SOSUS is hereby historically part of the United States Navy's Integrated Undersea Surveillance Systems (IUSS) network. The development started in 1949 with the first research phase at the MIT and the beginning of the installation of the first prototype in mid 1950. Unlike today's wireless sensor network approaches that are reviewed in this thesis, SOSUS consisted of bottom mounted hydrophone arrays that were connected by undersea communication cables to base stations at facilities on shore. The only wireless connection was between different base stations via satellite. The sensor nodes itself were deployed as dedicated nodes with only one function: sample the hydrophone data and send it to the base station in order to process the tracking algorithms. To improve the detection rate, the deployment was done deterministically, e.g. placing the hydrophone nodes on continental slopes and seamounts to improve the signal quality. In contrast, today's large scale networks are often deployed out of air, whereby the exact position of a node can only be estimated. In Figure 2.1, we show an example overview on the SOSUS deployment, illustrating the satellite interconnection of the base stations that retrieve the sensor sample data from the hydrophone sensors.

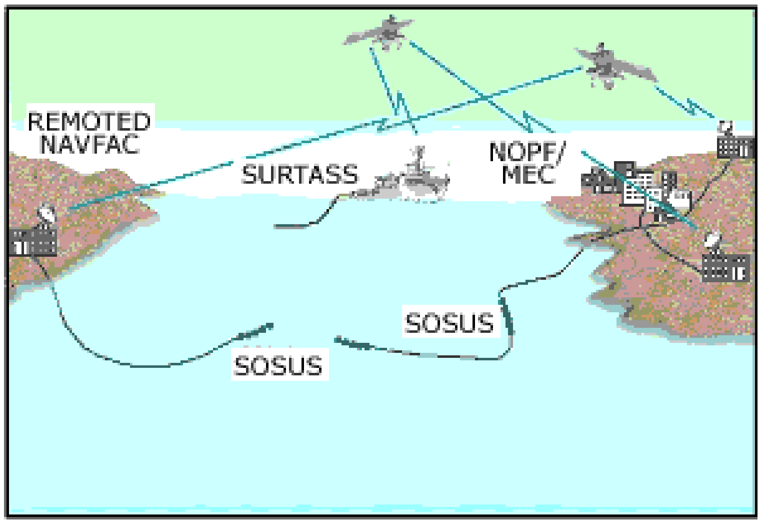


Figure 2.1: SOSUS Deployment Overview [174]

Until the 1960s, SOSUS was significantly extended to a world wide interconnected deployment, which can be denoted as one of the first very large scale sensor network multi-hop networks. In 1961, SOSUS tracked the USS George Washington from CONUS to the UK. During the next decades SOSUS was mainly used for tracking soviet strategically movement in field. Nevertheless, there were several documented rescue and science operations during that time [77], e.g. earthquake measurements [174]. With the end of the cold war in the beginning of the 1990s the SOSUS project was officially declassified, only parts of the deployment remain operational and former secret information got into public which can be used to improve future civil projects. The system is currently used by the National Oceanographic and Atmospheric Administration (NOAA) for detecting events in the ocean concerning seismic and animal activity.

Beside SOSUS, a milestone in the evolution of modern wireless sensor networks was the initiation of the first Distributed Sensor Nets (DSN) Workshop (CMU) in 1978 which was sponsored by Defence Advanced Research Projects Agency (DARPA). The research at that time was focussed on large scale networks of low-cost sensor nodes with the technical focus on sensing, communication and distributed data processing.

With the technical development of Micro Electro-Mechanical Systems (MEMS) and integrated circuits in conjunction with the predictions of Moore's law, the technical requirements for today's wireless sensor networks have been reached. It enabled very compact, autonomous and mobile nodes, each containing one or more sensors, computation and communication capabalities as demanded in the previous projects [115]. In 1999, Deborah Estrin et al. [66] and Joe Kahn et al. [115] published the vision of very large scale wireless sensor networks and the new challenges when operating these systems. These vision papers can be seen as the initial work for the type of sensor networks that are currently researched and are reviewed in this thesis. The corresponding development of the wireless sensor nodes started in 1998 with the Smart Dust project [115] and in 1997 with the NASA Sensor Webs project [23]. Both projects coin the terms of *mote* and *pods* respectively, denoting tiny wireless sensor node units, which are used in many areas of wireless sensor network research. The Smart Dust project is often referred to as the initial movement of the today's sensor network research community. Since then, various hardware platforms and software solutions in the field of wireless sensor networks have been developed and been influenced by this initial work. Hence, we give a more detailed introduction to the Smart Dust project in this paragraph. For extended information related to the NASA Sensor Webs project, we refer to [23] and [40].

The main research goal of the Smart Dust project was the development of autonomous sensing and communication devices in a cubic millimeter form factor. The milimeter-scale nodes are denoted as Smart Dust, forsaying the future possibility of mobile nodes remaining suspended in the air like flying dust, sensing and communicating for very long time [115]. Figure 2.2 and 2.3 show the future demanded scale and the currently reached scale of smart dust nodes.

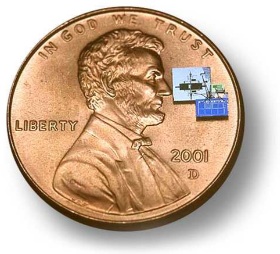


Figure 2.2: Smart Dust Desired Form Factor [115]

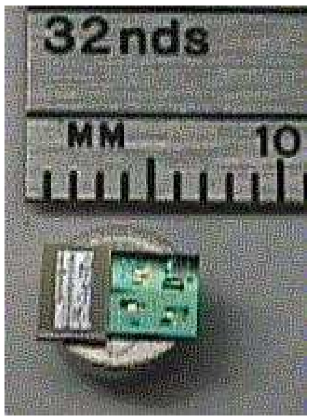


Figure 2.3: Smart Dust Currently Reached 5mm Form Factor [115]

The basic technology of Smart Dust motes corresponds to today available sensor node platforms. Consisting of signal-processing and control circuitry, optical receivers, thick-film power sources with limited energy supply and various sensors, the Smart Dust motes define the challenges of wireless sensor nodes that have been researched in the recent years. The major challenge of extending the lifetime of the energy limited sensor networks hereby has been the key motivation. The Smart Dust project particularly defines challenges for various research domains:

- The demanded form factor requires new techniques in designing batteries for long life energy supply.

- Due to the size limitations the processing capabilities are relatively small.
- The energy demand of the communication interface is the most significant aspect for the node's lifetime and hence the key goal for optimizations. The project defined further problems like the communication challenges concerning the line-of-sight and link directionality.
- The energy efficient interaction of a a huge variety of sensor devices is one design challenge.

The official Smart Dust Project ended early. However, it was the birth of the sensor network research, like the TinyDB data acquisition framework [43, 156] and TinyOS [141], that also influenced this thesis.

2.1.2 Application Scenarios

Wireless sensor networks are deployed for a wide range application fields ranging from military and civil surveillance, monitoring and automation areas. They are deployed for application tasks where former fixed wired sensor systems are too costly or not even applicable due to the application range. While there have been several deployments reported in the past, we introduce some of the most significant deployments and applications in this section.

Great Duck Island Habitat Monitoring

One of the most important and most referenced wireless sensor network deployments is the Great Duck Island habitat monitoring [33, 157, 222, 223]. This project has been done by the UC Berkeley starting in 2002 with the distribution of about 150 sensor nodes on the Great Duck Island in order to observe the nesting behaviour of local birds. The goal of the project was to develop a habitat monitoring kit that enables researchers worldwide to engage in the non-intrusive and non-disruptive monitoring of sensitive wildlife and habitats. The deployed sensor nodes were equipped with a microcontroller, low-power radio, memory and batteries. They collected sensor readings from sensors for temperature, humidity, barometric pressure and mid-range infrared. From the data management point of view, the system was deployed as a push system, sending the data periodically to a base station on the island where it was forwarded to the research lab via satellite. The data was made available as real-time environmental data in the WWW. This process has been introduced in the introduction of this thesis and is one of the main key features of the XML support for wireless sensor networks.
Other deployments for monitoring and tracking animals have been reported in [6, 113, 256]. Comparable studies on tracking animals like birds are discussed in [38, 112]. The ZebraNet deployment [113, 256] hereby differs from the static wireless sensor network deployments

as the sensor nodes are attached to a collar around the zebras necks and hence are mobile, which opens up new research domains and design issues [6].

Structural Health Monitoring of the Golden Gate Bridge

Another application scenario for wireless sensor network deployments is the structural health monitoring of buildings and other physical infrastructure. Wireless sensor networks are especially useful in scenarios where wiring is not possible or too costly, e.g. monitoring large buildings over many floors. One of the largest wireless sensor networks deployments for structural health monitoring has been reported in [124, 125, 180, 181]. In a project cooperation between the University of Berkeley and the sensor node vendor Crossbow, a wireless sensor network has been designed, implemented, deployed and tested on the 4200ft long main span and the south tower of the Golden Gate Bridge [125]. As shown in Figure 2.4, sensor nodes have been distributed over the main span and the tower to collect ambient vibrations without interfering with the operation of the bridge. The sampled data is collected reliably over a 46-hop network and compared to theoretical models and previous studies of the bridge [125]. Deployments like this are especially important in areas with high probability of natural disasters, e.g. earthquakes, to prevent buildings of structural damage.

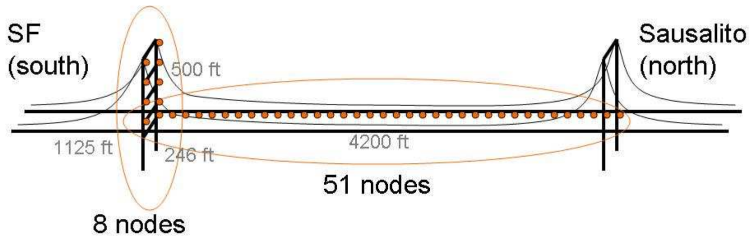


Figure 2.4: Structural Health Monitoring of the Golden Gate Bridge [125]

Beside the Golden Gate Bridge Project, extended descriptions on other sensor network deployments for building and process control and automation can be found in [249]. Deployment and application reports on building security are described in [68, 69, 94]. Reports on intelligent building sensing and control can be found in [203, 205, 211]. Building sensing and control enables pervasive homes which are often defined as *Smart Spaces* and *Smart Homes* [90, 177, 221]

Other Deployments Scenarios

As denoted previously, in the recent years there have been several deployments in civil and military application domains. Most of these deployments have been summarized in recent survey papers [5, 9, 17, 41, 119, 254]. As far as military application reports, e.g. on vehicle tracking and shooter localization, are published, different tracking and surveillance reports can be found in [92, 163, 214]. Sensor network deployments for monitoring the environment are described in [52, 53, 62, 81, 219]. A recent discussion of integrating wireless sensor networks in enterprise systems is given in [105, 161, 162]

2.1.3 Existing Technical Solutions and Resource Constraints

In this section, we discuss the general architecture of sensor nodes and give an overview on the specific parts of the technical configurations. We further discuss the limitations of today's sensor node platforms and the corresponding challenges and solutions for each layer of the OSI reference model [257].

In Figure 2.5, we show the general technical architecture of a sensor node. A sensor node basically consists of four components: a microcontroller, a power source, a transceiver and a sensor controller that acts as an interface to various sensor devices.

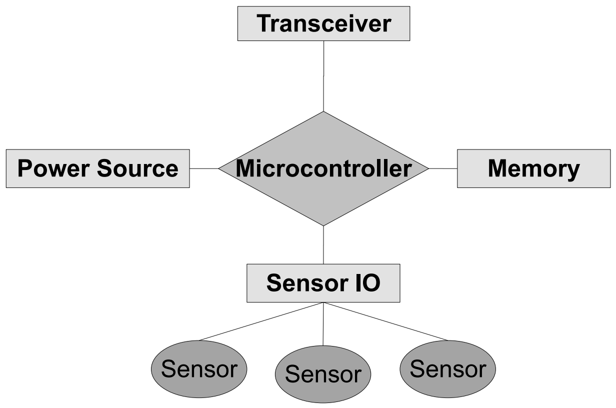


Figure 2.5: General Hardware Architecture of Wireless Sensor Nodes

The microcontroller is the processing unit of the sensor node that processes the data and controls the communication and other components of the device. In the recent years, sensor node platforms have been equipped with several types of microcontrollers and digital signal processors (DSPs) of various manufacturers. Hereby, the state-of-the-art microcontrollers provide energy efficient processing power while remaining small sized and hence suffice the tiny form factor demand. Typical sensor node microcontroller manufacturers are Atmel (e.g. Atmega128L), Phillips (e.g. LPC 2136) and Texas Instruments (e.g. MSP430). The microcontroller is directly attached to external memory. Following the Harvard Architecture most RISC-based sensor nodes hereby are equipped with physically separate storage and signal pathways for instructions and data [92].
Sensor nodes that are build as Application Specific Integrated Circuits (ASICSs) can be the most energy efficient platforms [189]. However, due to the limited flexibility of these nodes after deployment, they have been only discussed for very specific application scenarios. The usage of Field Programmable Gate Arrays (FPGAs) [160] as dynamic coprocessors on the sensor node has been discussed in the recent years [13, 176, 228].

They provide energy efficient reconfigurable circuitry to enhance the lifetime of the sensor node when processing large amounts of sensor data continuously. However, due to the miniaturization demand that conflicts with the size of existing FPGAs the integration process is still in early stages and today's sensor node platforms absent from using FPGAs.

Due to the general large scalability of wireless sensor networks, sensor node transceivers mostly use the Industrial, Scientific and Medical (ISM) Band which is globally available. In the recent years various transmission concepts like infrared and optical communication or using general radio frequencies have been discussed. The state-of-the-art communication is radio frequency (RF) based communication working on frequencies from 433 MHz to 2.4 GHz. For long term deployments energy efficient transceivers and protocols have been investigated. They include energy efficient wake-up-sleep cycles and energy conservation on higher layers of the corresponding protocol stack.
Typical radio hardware manufacturers are Chipcon AF (e.g. CC1000 an CC2420), RFM (e.g. TR1001868) and Semtech (e.g. XE1205). A widespread energy efficient communication protocol stack is hereby the ZigBee standard [12, 126] which includes the IEEE 802.15.4 standard. Besides, the usage of Bluetooth technology has also been discussed in [17, 140, 169]. Other radio technologies like Wifi (IEEE 802.11) are not used frequently due to the lower energy efficiency.

Wireless sensor nodes are typically equipped with batteries providing a limited power supply. Active power supply like solar panels has been also discussed in [27, 47, 242]. Recharging the batteries by using environmental conditions has been discussed in [202]. Nevertheless, typical sensor node platforms and deployments work with limited power supplies and require additional concept for energy conservation to extend the network lifetime.
The sensor controller attaches various sensor devices to the sensor node, ranging from temperature sensors to complex chemical sensors. Based on the complexity of the sensor, frequently sensing the environment leads to a high energy demand and significantly reduces the network lifetime. While the sensors themselves do not provide processing logic, additional concepts over all system layers are required to support energy efficient sensing, e.g. adapting the sensing rate to data requirements [156].

In the following part of this section, we give a more detailed introduction on typical sensor node platforms and their technical constraints and design space.

Sensor Node Platforms

In the recent years, several sensor node platforms have been introduced by university project groups and electronics-specialized companies. One of the first sensor node designs following the Smart Dust target specifications was the UC Berkeley sensor node platform

MICA [92]. The sensor nodes have been denoted as *Motes*. This platform has been commercialized by the company Crossbow Inc. (http://www.xbow.com/) and various types of MICA motes have been presented. Today, Crossbow is one of the leading manufacturers of wireless sensor nodes. Beside the MICA mote platform, other sensor node platforms have been designed by university research groups like Eyes [88], Tmote [246], Pacemates [153], Telos [187], BTnodes [17] and iSense nodes [45]. In Table 2.1, we give an overview on existing sensor node platforms and their technical specifications.
We show the most important of these sensor nodes in Figure 2.6 and 2.7. A complete historical overview on existing sensor node platforms can be found in The Sensor Network Museum at http://www.snm.ethz.ch. The approaches that are presented in this thesis have been developed and implemented and evaluated for the BTnodes, Pacemate and iSense Core platforms. Nevertheless, they are fully adaptable and applicable on all presented sensor node platforms. In the following paragraphs, we discuss a selection of the most important sensor node platforms.

	BTnode v3	**MICA2**	**MICA2dot**	**MICAz**	**telos A**	**tmote sky**	**EYES**	**Pacemate**	**iSense Core**
Manufacturer	Art of Technology	Crossbow			Imote iv		Univ. of Twente	Univ. of Luebeck	Coalesenses
Microcontroller	Atmel Atmega 128L				Texas Instruments MSP430			Phillips LPC 2136	Jennic JN5139
Architecture	8 Bit				16 Bit			32 Bit RISC	
Speed	7,37 Mhz		4 Mhz	7,37 Mhz	8 Mhz		5 Mhz	8 Mhz	16 Mhz
Program Memory (ROM)	128 kB	128 kB	128 kB	128 kB	60 kB	48 kB	60 kB	256 kB	80 kB
Data Memory (RAM)	64 kB	4 kB	4 kB	4 kB	2 kB	10 kB	2 kB	32 kB	16 kB
Storage Memory (Flash)	180 kB	512 kB	512 kB	512 kB	256 kB	1024 kB	4 kB	0 kB	128 kB
Radio	Chipcon CC1000			Chipcon CC2420			RFM TR1001868	Semtech XE1205	IEEE 802.15.4 compliant radio
Outdoor Range (m)	150-300			75-100					
Size (mm2)	1890	1856	492	1856	2080	2621		3304	1350

Table 2.1: Sensor Node Platform Comparison of Technical Specifications

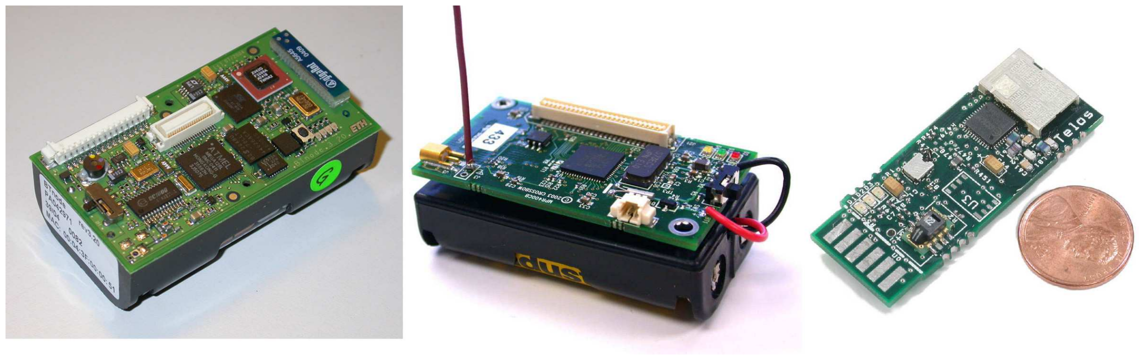


Figure 2.6: Wireless Sensor Nodes (from left): BTnode v3, CrossBow MICA2, Telos

The MICA platform, TinyOS and BTnodes
The Berkeley MICA mote platform is often introduced as the de facto standard sensor node platform. Today, they are commercially produced and sold by the company Crossbow Technology Inc. There exist different types of MICA motes with various hardware settings and form factors. In Figure 2.6, we show the popular MICA2 sensor node. Other follow up products include the MICAz platform, which operates on the 2.4GHz band, and the Intel-designed IMOTE2 sensor node. Berkeley-style MICA nodes implement the TinyOS soft-

ware framework [141], a component-based programming paradigm and application programming interface (API) that has been particularly designed for wireless sensor networks and is operated by an international consortium, the TinyOS Alliance.
The TinyOS API provides an efficient hardware abstraction layer to simplify the complexity of sensor node platforms. TinyOS applications are written in the nesC (network embedded systems C) programming language [74] which introduces component-based, event-driven applications for the TinyOS platform. The nesC language is an extension to the C programming language. Accordingly, all contributions of this thesis that have been evaluated on Embedded C programmed sensor nodes are also applicable on nesC programmed sensor nodes.

The BTnodes hardware platform has been introduced by the ETH Zuerich Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems [17]. In Figure 2.6, we show the BTnodes v3 sensor node that has been used for developing and evaluating the concepts of this thesis. The BTnodes architecture shares many technical specifications with the Berkeley-style MICA platform as shown in Table 2.1. The processing is done by a 8bit Atmel Atmega128l microcontroller. While the program memory equipment is equal, the BTnode sensor nodes are equipped with a larger amount of data memory, representing the latest progress in the miniaturization of memory cells. The communication specifications (baseband, MAC and link-layer protocols) is also common for most of the presented architectures. However, one of the main interests in developing the BTnode platform was the integration into other networks. Therefore, unlike the MICA nodes, the BTnodes are equipped with a Bluetooth controller, letting them interact with any Bluetooth-enable devices without the need to integrate further hardware or software [17].
This possibility of hardware interoperability was one reason for the selection of BTnodes as a prototyping platform for the concepts of this thesis, e.g. supporting heterogeneous sensor networks and integrating wireless sensor networks in the WWW. BTnodes are programmed in standard Embedded C using the BTnut API that provides a hardware abstraction layer [16, 60]. Besides, the BTnodes hardware is also compatible to the TinyOS API.

The iSense OS platforms (Pacemates and iSense Core nodes)
Other sensor node platforms that have been extensively used for implementing and evaluating the concepts of this thesis are the Pacemate sensor nodes and iSense Core modules that run the *iSense OS*. Like TinyOS and BTnut, iSense OS is a modular, flexible and convenient API that provides abstracted access to the node hardware and a broad variety of networking protocols . Sensor nodes that run iSense OS are programmed in C++ and Embedded C. The iSense OS framework also provides a built-in support for simulating iSense applications using the simulator *Shawn* [131].
In Figure 2.7 (left), we show the Pacemate sensor node. The Pacemate sensor node platform has been designed and developed for the MarathonNet project of the Institute of Telematics

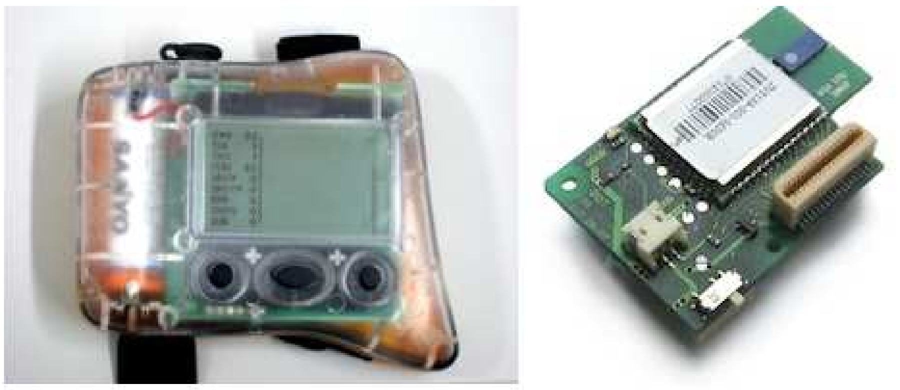


Figure 2.7: Wireless Sensor Nodes II (from left): Pacemate, Coalesenses iSense Core

of the University of Luebeck [153]. The project goal was to design an application to track marathon runners paths and vital values over a mobile ad-hoc network. Due to usability reasons, the Pacemates have a larger form factor consisting of an ergonomically designed case and a user interface that lets the runners access their data during the race. While the technical specifications of the Pacemate nodes are comparable and representative to other sensor node platforms like MICA motes, the ergonomic case, input interface and display make them an ideal platform for fast debug and prototyping. As denoted previously, the Pacemate nodes are programmed in C++ and Embedded C using the iSense OS API. Besides, they have also been used for developing the service oriented operating system SurferOS that is also part of the AESOP's TALE project [152].
In Figure 2.7 (right), we show the iSense Core module. The iSense Core modules are a higly flexible sensor node platform designed and developed by Coalesenses. Coalesenses is a research spin-off of the Institute of Telematics of the University of Luebeck that officially supports and develops the iSense OS API. The iSense Core modules are state-of-the-art sensor nodes, equipped with the latest processing, communication and sensor technology. While the actual iSense Core module only consists of a Jennic 32bit microcontroller and a 802.15.4 compliant radio device, it can be extended with various types of sensor devices, ranging from temperature sensors to security modules. The iSense sensor node platform is the reference platform for the iSense OS API. Furthermore they are supported by the iShell programming, operating and analysis tool that provides a variety of functions for operating and debugging iSense wireless sensor networks. The technical specifications and the form factor of the nodes make the iSense Core platform the target reference platform for the concepts of this thesis. Nevertheless, as denoted previously the results of this thesis can be adapted for all sensor node platforms that support the C programming language and are within the technical specifications that are discussed in the next paragraphs.

Design Challenges

Wireless sensor networks are distributed ad-hoc networks / systems [247]. Accordingly, they share many design and implementation concepts with previously existing mobile ad-hoc networks (MANETs) [119]. Nevertheless, in [119] Karl and Willig introduce the most important reasons that make wireless sensor networks different to existing MANETs. In [201], Roemer and Mattern discuss the design space of sensor network applications and show that applications deviate in various dimensions under various constraints. In the following paragraph, we summarize the differences and key challenges for engineering applications for wireless sensor networks. We hereby discuss technical limitations and clarify the design space of sensor network applications.

Wireless sensor networks are deployed for a wide variety of application scenarios consisting of various hardware and software configurations. The design of the sensor network has to fulfil various requirements regarding hardware issues and software support [201] which opens up a very large design space. In detail, a formal definition of the application requirements and the network conditions is often not possible due to the variety of issues in wireless sensor networks, e.g. network scale, communication technique, ad-hoc configuration and hardware configuration. Many research papers in the past worked with assumptions of the network model. However, in [201] Roemer and Mattern discuss that these assumptions might often not be correct due to the various dimensions of the extensive sensor network design space.
In Table 2.2, we show the various fields of the design space. Wireless sensor network deployments can differ in their deployment type and network size, the usage of mobile vs. immobile nodes, the hardware resource configurations that includes homogeneous and heterogeneous configurations, the type of network infrastructure and network topology, the coverage and the connectivity. Moreover, sensor networks are deployed for a dedicated lifetime. This lifetime is most critical to be achieved and has deep impact on the selection of network maintenance and communication algorithms.
The dimensions described in Table 2.2 can be applied to describe existing projects like the Great Duck Island project that has been introduced in Section 2.1.2. The Great Duck Island deployment was an one-time deployment whereby immobile nodes were manually placed inside the bird burrows. The sensors where equipped with various sensors to measure the humidity, light and temperature. The presence of the birds was detected with infrared sensors [201]. According to Roemer and Mattern the hardware resources of the used nodes can be classified as *matchbox* [201] based on the physical dimensions. The technical specifications are comparable to the ones described in Table 2.1. While there are more dimensions of the Great Duck Island deployment described in [201], these listed dimensions show that there are several issues that need to be reviewed during engineering the sensor network application.
As described in [15], for building generic applications one has to assume a worst case sce-

nario, e.g. assuming minimum node capabilities and highly mobile nodes. Nevertheless, this approach does not explore the real design space of the application. A variety of hardware and software solutions is indeed essential to explore even large portions of the design space. The actual application design hence becomes a very complex and error prone task that requires new concepts of software engineering and a critical discussion of the technical constraints and metrics in sensor networks. This application scenario dependent adaptation of the design space [15] is one key feature of the adaptation of the service oriented paradigm to sensor network engineering as discussed in the introduction of this thesis.

Dimensions	**Property Classes**
Deployment	randomly vs. manually deployed; one-time vs. iterative deployments
Mobility	active vs. passive mobility, occasional vs. continuously mobility of all or only selected nodes
Cost, Size, Resources and Energy	unlimited resources vs. very restricted resources
Heterogeneity	homogeneous vs. heterogeneous sensor node platforms
Communication Modality	radio vs. light vs. inductive vs. capacitive vs. sound
Infrastructure	fixed infrastructure vs. ad-hoc networks
Network Topology	single-hop vs. star vs. networked stars vs. tree vs. graph
Coverage	sparse vs. dense vs. redundant
Connectivity	connected vs. intermittent vs. sporadic
Network Size	few nodes vs. thousands of nodes vs. even more
Lifetime	hours vs. days vs. years
Other QoS Requirements	real-time demands, robustness and others

Table 2.2: Dimensions of the Sensor Network Design Space according to [201]

The challenges in designing software for sensor networks have been also discussed in the early sensor networks projects. In their early work in the field of wireless sensor networks *Next Century Challenges: Mobile Networking for Smart Dust*, Kahn et al. [115] were predicting these challenges. The predicted key challenges can be summarized as:

- **limited hardware resources** of the highly integrated sensor nodes
- a **large amount of devices** in the network leading to a complex collaboration task
- a **tight coupling of the application scenario** and the sensor nodes
- a **broad usage profile** ranging from system-experts to non-expert users, e.g. biologists.

Many of the mentioned challenges also apply for general mobile ad-hoc networks (MANETs). However, as Karl and Wittig describe in [119], there are significant differences in many fields of the application design. The most important differences are:

- the previously described **wider application design space**, e.g. very different network densities
- the **environmental interaction**, e.g. non-continuous expected data delivery rates including unpredictable data bursts in event detection
- a **higher scalability**, e.g. up to hundred of thousands of distributed nodes

- a **scarce energy supply** making the energy consumption the primary metric to be considered
- the requirement for new energy efficient and robust **self configuration** concept due to extended lifetime demand
- very different concepts of **dependability and quality of service**, e.g. a large variance in delivery rate requirements
- **data-centric network maintenance**, e.g. the single node is not important, only the data is important
- the limitation in hardware resources require much **simpler operating and networking software**.

All of the presented surveys on design challenges in wireless sensor networks show the complex task of application engineering. This is especially relevant when only algorithms and concepts are designed that should be part of complex applications. Therefore in the past, many concepts for wireless sensor networks, e.g. routing or data management algorithms, have been designed using generic sensor network models that should describe a large design space. The algorithms for these concepts have often been tested in simulators that represent the network model. Kotz et al. and Kurkowski et al. claim in their work [129, 134] the results of this simulations can often not been repeated or applied either in simulation or especially in real sensor network deployments. To produce more representative results, the concepts of this thesis have therefore been implemented and tested on real sensor node products considering all of the described challenges. Hereby, we noticed that the largest design challenge are the technical constraints of sensor node platforms, e.g. the limitations in the hardware resources. We therefore discuss these constraints in more detail in the next paragraph.

Technical Constraints

Beside our own experiences in implementing and testing applications on sensor nodes, almost all sensor network research publications claim that the actual greatest challenge in wireless sensor network are the limited hardware resources. Most of the applications and concepts of conventional (distributed) computing or even other mobile ad-hoc networks, e.g. cell phones, are not applicable for wireless sensor networks due to the large resource gap between these platforms. To explore the resource configuration space Beutel defines metrics for wireless sensor networks in [15]. These metrics compare existing sensor node platforms to find the actual limitations in all parts of hardware configurations. In summary, wireless sensor networks are limited in the following hardware metrics:

- System Core

  - CPU architecture
  - CPU speed (clock frequency)
  - Program memory
  - Data memory
  - External storage
  - Energy conservation (idle, processing)
  - Node size
  - Amount of on-board sensors
  - Amount of external IO channels

- Radio System
  - Band frequency
  - Channels
  - Data rate
  - Setup time
  - Energy conservation (transmitting, receiving)
  - Sensitivity
  - Outdoor / Indoor range

All of the described metrics have different impact on design decisions for wireless sensor applications. However, it has been shown that the power consumption and the limited energy supply are the most challenging hardware metrics [15]. Different sensor node platforms can vary in there general power consumption. Nevertheless, the power consumption of today's sensor nodes is significantly higher for communication. Therefore, sensor network application should avoid extensive communication as far as possible, e.g. in Chapter 6 we discuss a technique to significantly reduce the communication demand. For the system core the processing power limitations, e.g. using energy efficient, but slow microprocessors, and the memory capabilities, e.g. data memory of only up to 64 Kb, are the most challenging metrics that require efficient programming concept.

While the early work of Beutel in [15] compares a limited number of sensor nodes and various metrics that are of lower significance for the concepts of this work, we present an updated comparison in Figure 2.8. We hereby include the three sensor node platforms that have been used for evaluating the concepts of this thesis: BTnode v3, Pacemate and iSense Core module. The comparison shows that all three platforms have been improved in various dimensions compared to the older Mica platforms. Nevertheless, the hardware configurations are highly limited compared to today's personal computers or other mobile

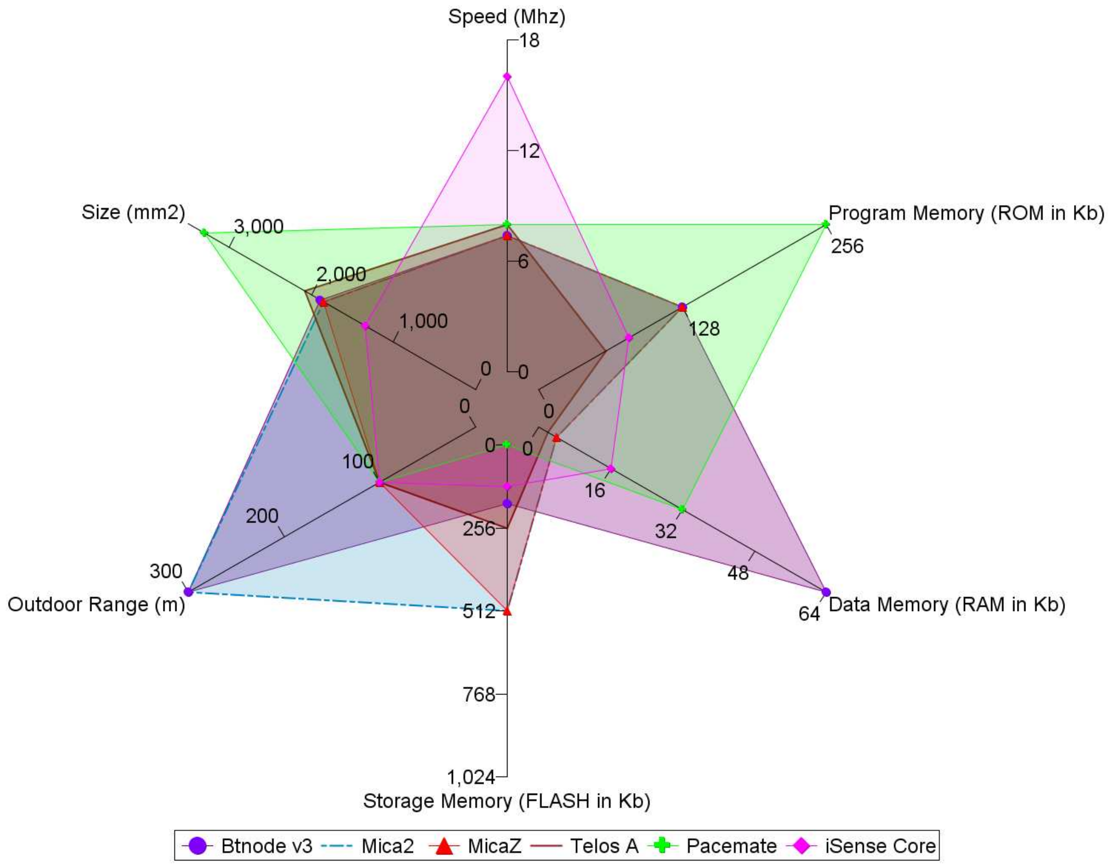


Figure 2.8: WSN Platform Comparison Technical Specification

devices. The CPU architectures range from 8bit microcontrollers as used on Atmel Atmega based Mica nodes to 32bit systems, e.g. Jennic microcontrollers on iSense core modules, with microprocessors clocked from 8 to 16 Mhz. Accordingly, the processing speed is significantly lower than on existing mobile devices, e.g. smartphones with microprocessors which are clocked up to 1 Ghz [194]. As denoted previously, other significant limitations are the memory capabilities. This is even more important for data management and in-network data storage concepts as presented in this thesis. The overall program memory is limited by up to 96 Kb leaving a dynamic heap data memory of up to 64 Kb. Some nodes are equipped with external flash memory of up to 512 Kb. Nevertheless, this memory has limitations for random access write operations which can require additional paging concepts for placing and updating sectors in the heap memory. Research results on the efficiency of these operations on sensor nodes are yet not available.

To manifest challenges of the hardware limitations, we describe a generic state-of-the-art sensor node hardware configuration as follows:

- CPU architecture: 8bit, 16bit, 32bit
- CPU speed (clock frequency): 8 Mhz

- Program memory: 96 Kb
- Data memory: 32 Kb
- External storage: 256 Kb (optional)

These sensor node hardware configurations are the recommended system requirements for the concepts presented in this thesis. Nevertheless, as we will show in Chapter 5, the minimum system requirements for the program frameworks of this thesis are even lower.

2.1.4 Research Domains

The recently gained publicity and importance of wireless sensor networks has motivated researchers of several domains in working on these networks. Due to the deviated design space many research areas have been opened up. Thereby most research is concentrated on finding long life, energy efficient and robust networks concepts. In-network data acquisition has been also discussed frequently. Recent approaches target usability issues for sensor network developers and users.
The research areas in wireless sensor networks span across all layers of the OSI reference model. Many approaches and concepts are thereby defined as cross layer approaches due to the not clear separation for optimization issues. In this section we introduce some research projects for the physical, network and transport layer of the OSI reference model. Due to the wide range of research in sensor networks we can not cover all projects in this introduction. Especially the application layer includes a huge variety of research work. We therefore refer to recent survey papers published in [5, 6, 9, 12, 54, 119, 228, 254].
The research concepts of this thesis belong to the field of data acquisition, data management and in-network storage. We discuss the research in the field of data management and query processing in wireless sensor networks in detail in Section 2.2.

Physical Layer and Medium Access Control

Due to the large scalability of wireless sensor networks one of the crucial tasks is to choose an adequate RF band which should be licence exempt. As a result, low data rate but energy efficient wireless communication standards such as IEEE 802.15.4 [165] have been designated including the definition for the PHY and MAC layer [12]. A significant energy efficiency can only be reached by optimizing across all layers as described in [66]. The wide spread Zigbee standards hence extends the IEEE 802.15.4 standard on higher layers [12] [126].
Wireless sensor network protocols are mostly designed to let the sensor node be left in a sleep / doze mode. Staying in sleep mode significantly extends the life time as current sensor nodes consume up to ten times the power in idle mode in relation to sleep mode. As shown in the previous sections, the energy consumption for data transmission and communication is even worse. This is the reason why across all layers protocols and applications try to send the less data as possible but instead tend to push pre-processing tasks deep in the

network. MAC protocols as described in [54, 82, 91, 126] concentrate on sparse wakeup schedules to maximize power saving. Nevertheless, this can result into latency that might be unacceptable for time critical alert and event detection application scenarios which require adapted MAC protocols [110].

Networking and Transport

Wireless sensor networks are ad-hoc networks. Hence, achieving a network connection is a distributed task on the network layer without additional aiding infrastructure [64, 92]. The task is even more complex as wireless sensor networks are very dynamic networks. Single nodes tend to fail, e.g. caused by energy depletion, which can result into separation of the networks at communication bottlenecks. While single hop networks limit the network size and are not likely for real deployments, multi hop networks have been researched in the past. Moreover, the routing approaches can be divided in those that treat every node equally and those that route via cluster heads that are representative for separate node groups.
Routing protocols need to be adapted to the scarce energy resources. As an example, energy efficient unicast routing needs to use short paths in the network. Nevertheless, nodes on the route that are already low in battery should be avoided, resulting in a longer but more robust routing path [119]. One of the well known early approaches is the clustering approach LEACH [89], other approaches are described in [3, 11, 36, 210]. Energy efficient multicast routing algorithms are introduced in [32, 48].
The route selection protocols often impose a hierarchical topology that needs to be maintained by a topology control. Survey papers on this topic are published in [204, 241]. Wide spread topology concepts are routing trees and geographic routing. Routing trees are often used by data acquisition applications like TinyDB [156]. A well known concept is TAG (Tiny AGgregation) [155] that uses a minimum spanning tree to route aggregates to a data sink. However, other approaches like Synopsis Diffusion [170] claim that fixed tree structures can not be maintained in real deployment scenarios where nodes tend to fail at high probability. Instead, these approaches suggest a ring-oriented multicast routing approach that has been used in this work as described in Section 6. Another approach that has gained a lot of attention is *Directed Diffusion*, where the routing topology is a directed acyclic graph (DAG) rooted at the sink used for multipath data delivery. Geographic routing strategies, also denoted as greedy routing, have been introduced in [130, 255]. The idea is to forward packets based on geographic information, e.g. minimizing the absolute remaining distance. Sensor nodes need to know their location, e.g. coordinates. In the past, this approach has often been used for GPS equipped sensor nodes making it more applicable for open air deployments. Nevertheless, the routing can fail resulting in endless packet forwarding. Approaches e.g. described in [209] try to solve this problem.
Designing an efficient, robust transport protocol for wireless sensor networks is difficult to the exterior influences, e.g. radio interference, and energy efficiency requirements. Communication protocols therefore absent from setting up special transmission channels or data

paths. As a result, the transport is comparable to the User Datagram Protocol (UDP) [188]. Open questions are still which level of dependability and quality of service should or can be reached in non robust wireless sensor networks. Some concepts have been discussed in [4, 71, 218, 240].

Other Research Domains

Beside the described layers, there has been a lot of work in higher layers. One important topic is security which has been discussed in [34, 185, 186, 215]. Another topic is time synchronization. Time synchronization is of significant importance to synchronize sensor measurements and sleeping cycles. Important work in the field of time synchronization can be found in [63, 65]. Beside the data acquisition task, localization and event detection are the most common applications. As denoted in the previous paragraph, localization is i.a. important for geographic routing. For a detailed introduction we refer to [106, 137, 159]. Event detection has been discussed i.a. in [30, 37, 142, 143].

2.2 Data Management and Query Processing in Wireless Sensor Networks

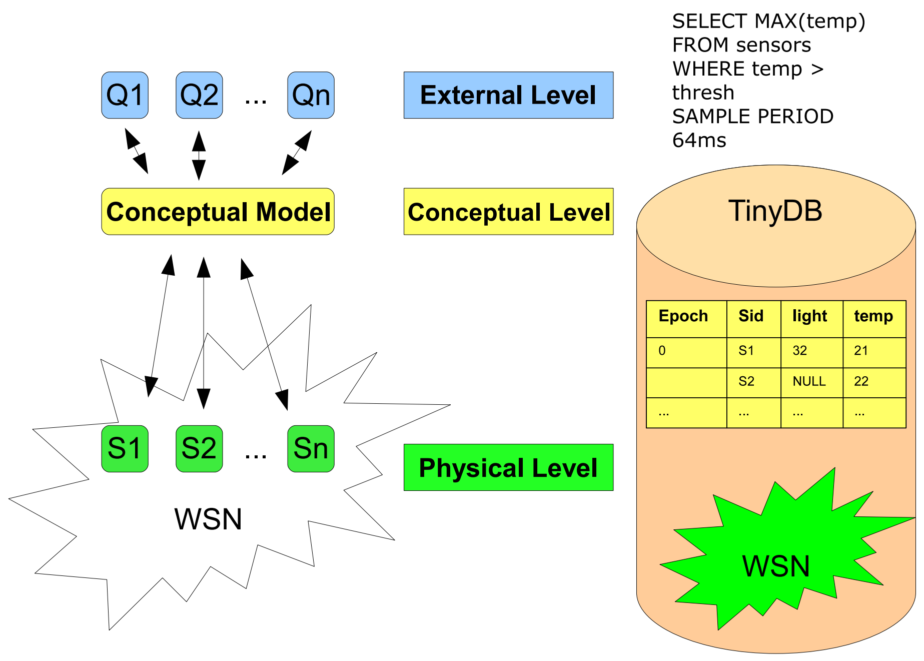
Wireless sensor networks are mainly deployed for data acquisition of measured environmental data. Thereby, there exists a variety of concepts from in-network data storage and analyzation to traditional push and pull applications. Data acquisition in wireless sensor networks often follows the concept of *data-centric* routing. Data-centric routing has been described as the core abstraction of wireless sensor networks [119]. Applications should access data instead of addressing individual nodes using an in-network processing framework. The actual routing process is up to the corresponding routing protocol as described in the previous section. The data access is supported by specialized sensor network query languages that combine a traditional query interface with energy efficient in-network query evaluation as shown in the next subsection.

2.2.1 Query Languages and Data Models

Since data acquisition is a central aspect in sensor network data management there exists significant work on energy efficient query evaluation in wireless sensor networks [97, 101, 102, 156, 253]. Data-centric query languages have been introduced and in-network aggregation is used to support energy efficient result processing [43, 156, 253]. Surveys of the initial work in data management in wireless sensor networks are given in [6, 31]. In the next paragraphs we discuss the idea of representing the sensor network as a database and the idea of in-network storage. We further introduce the initial approaches for data acquisition frameworks *Cougar* and *TinyDB*. The later represents the state-of-the-art data management solutions for many of today's general application deployments. We give a detailed overview on the query language and the corresponding data model to show the limitations of these approaches in the next subsection.

The Sensor Network as a Database

As denoted for the data-centric query concept, the idea of effectively using wireless sensor networks includes an abstraction of the actual network topology. The user is more interested in having simple data acquisition methods. Where the actual data is located, how the query gets to the corresponding nodes and how results flow to the data sink is not of high interest. The only important factor is that the required data is delivered to the query issuer in an energy efficient way to maximize the network lifetime.



```
SELECT MAX(temp)
FROM sensors
WHERE temp >
thresh
SAMPLE PERIOD
64ms
```

Epoch	Sid	light	temp
0	S1	32	21
	S2	NULL	22
...	...	...	...

Figure 2.9: The Sensor Network as a Database

Following this theory, the wireless sensor network can be seen as a database that somehow stores data in a distributed way and provides an efficient query interface and abstract views on the data. This concept follows the ANSI-SPARC architecture [10, 111] of traditional database systems as proposed in Figure 2.9. The external level (query layer) consists of the queries and query results of the users. The user thereby not only queries for data, he actually programs the sensor network through queries using a query-like declarative language, as the data is only collected when queries are active in the network due to energy efficiency [31].
The conceptual level includes a global representation of the distributed sensor data. This is the actual abstraction as a global database hiding the real distribution. In the past, several data models have been used for logically representing the distributed data in the net-

work. The TinyDB approach uses a simple global table to represent the distributed nodes [156]. The global sensor network approach as described by Aberer et al. [1] tries to embed the structure and hierarchy of distributed sensor networks in a structured global XML document. While this approach represents the data outside the network, in this thesis we introduce concepts for managing this representation in the network. This results into significant improvements compared to simple data models like TinyDB as described in the introduction of this thesis and in the following Section 2.2.3.
While many of the initial data management solutions view sensor network databases as a continuous stream, in recent years, motivated by the evolution in flash memory technology, storage-centric sensor networks have been discussed [58]. In these sensor networks a high amount of data is stored on physical level over time to allow in-network analyzation of long-term data. In detail, the physical level is responsible for storing the distributed data persistently including the choice of the storage medium, e.g. flash memory or heap memory, compression concepts and other physical optimizations. In this work, we discuss a compressed distributed XML in-network storage scheme that can be used for heap and flash memory storage. We therefore discuss the concept of in-network data storage in more detail in the next paragraph.

In this paragraph, we showed that wireless sensor network database concepts and traditional database systems have a great deal in common. Nevertheless, according to Carreras et al. [31] there are three main differences that require new concepts in processing data:

- Streamed result data: the query results are often sent as a stream; queries are often continuous.
- Communication errors: communication links are unreliable and a not delayed, reliable data delivery can not be guaranteed.
- In-network processing: while transmission energy costs are several orders of magnitude higher than in-network processing, processing query results, e.g. calculating the average temperature, is pushed deep in the network.

In-Network Storage

According to Diao et al. [58], wireless sensor network applications can be differentiated into live data and historical data applications. Live data applications like event detection only require a conceptual scheme to access the data. The actual data is immediately processed after sensing, either throwing an event or not. On the other side, there has been a lot of long running environmental observations that require continuous data, e.g. calculating the average temperature of the week. These application scenarios are the main areas for the data acquisition frameworks Cougar and TinyDB that will be discussed in the following paragraphs, although both approaches were initially designed for immediately processing data continuously [58]. A recent approach that has been designed exclusively for in-network storage is StoneDB [58] that includes the energy efficient use of flash memory and provides a query processing interface.

Delivering single sensor values continuously to the data sink is costly due to the high energy demand of data transmission. The idea of in-network processing includes pushing algorithms that work on large windows of sensor data deep in the network to only send global, final results back to the sink. These concepts require long-term in-network data storage. The technology trends in miniaturization and capabilities of flash memory cells have opened up new opportunities not only for just storing data in the network [72] but also for other concepts like redundancy control, replication and data archiving.
In detail, data migration and redundant data storage in the network are discussed in [118, 226]. The approach discussed by Lin et al. [148] includes an optimization of in-network persistence by using XOR-combined data representations. The idea is to use a minimal spanning tree, that is denoted *interval tree*, that covers all sensor nodes in the network. Now the nodes perform a bitwise XOR operation in the tree from bottom to top. Each node in the interval tree thereby XORs all the backup data sent from its direct children along with its own data and stores this results as backup information and forwards it to its parent. After recovering a failure node, the neighbour nodes restore the data out of its XOR backup information. The advantage of this approach is that XOR operations can be processed in an efficient way. Other approaches discussed in [76, 147, 197] also describe replication concepts. In [197] a concept for replicating relevant data to cluster heads for event detection is discussed. The approach described in [76] partitions the network in geographic zones where each sensor node is responsible for a specific task, e.g. storing replicated data or managing the replication process. In [147] two algorithms for placing replicas via XOR combination and fragmentation in a tree topology are discussed. Finally, in [252] the network scheduling issue for an energy efficient data archiving task is discussed.

The concepts of this thesis include in-network storage concepts. The sensor nodes store their data over long periods as XML documents, whereby the XML data is structured chronologically ordering the sensor measurements. While the actual XML documents on each sensor node represent a conceptual schema, multiple sensor nodes can be embedded in a global hierarchical XML document following the global sensor network concept of Aberer et al. [1]. The data can be accessed using standardized query languages like XPath. We further discuss sensor network specific optimizations for the query evaluation. Nevertheless, many strategies of the early Cougar and TinyDB approaches are also applicable for XML query evaluation. Hence, we discuss these approaches in the next paragraphs.

Cougar

The Cougar project [253] is a solution for efficient query processing in wireless sensor network that has been designed by the Cornell University database group. It is often referred to as the initial work on efficient data acquisition in wireless sensor networks. Cougar is a clustering based evaluation mechanism that differentiates three types of sensor node roles: Data is produced at *source nodes*. The source nodes route their data to *intermediate nodes* which can be defined as cluster heads of a cluster of source nodes. These cluster leader nodes route the data finally to the *sink nodes* that are often referred to as gateway nodes. Cougar uses Directed Diffusion as a main routing protocol on network layer. Queries are expressed in a high level declarative language that is based on SQL. Query optimization is done on a PC that is connected to the sensor network. The resulting query execution plan is translated to command calls of receiver functions on the nodes. The single sensor node therefore is not able to completely process a given user query alone. Cougar provides optimizations for evaluating aggregation queries. Therefore, intermediate nodes are not only responsible for forwarding results of the source nodes but also for processing aggregate functions early deep in the network. The actual size of data that needs to be transmitted is significantly reduced.

TinyDB

TinyDB is often referred to as the state-of-the-art data acquisition technique for wireless sensor networks. It has been designed by Samuel Madden et al. and been described in [156] for TinyOS-based sensor nodes. On the conceptual level the TinyDB approach supports a simple relational data model as shown on the right side of Figure 2.9. The whole sensor network is represented as one unstructured table, denoted *sensors*. The relational schema consists of a sensor node id and an unlimited number of optional sensors. A sensor node is represented as a row consisting of it's id and all of it's actual sensor values. If the sensor node does not operate a sensor that is defined by the schema it includes a NULL value. We give another example motivated by the Great Duck Island deployment in Table 2.3. The table contains the information of the luminance in the bird nests. The epochs define a chronological series of measurements.

Epoch	**NodeID**	**NestNo**	**Light**
0	1	17	1455
0	2	27	1389
0	3	17	1422
0	4	25	1405
1	1	17	5
1	2	27	1389
1	3	17	8
1	4	25	1405

Table 2.3: Example TinyDB *sensors* Table for Bird Observation

The sensor data can be accessed using the TinySQL language that is designed in a SQLstyle. TinySQL is designed as a high level data acquisition language to allow "programming" applications in a data-centric way without the knowledge of real sensor network programming, e.g. programming in Embedded C. We present the TinySQL grammar in Listing 2.1.

Listing 2.1: TinyDB Query Grammar

```
SELECT <aggregates>, <attributes>
[FROM {sensors | <buffer>}]
[WHERE <predicates>]
[GROUP BY <exprs>]
[SAMPLE PERIOD <const> | ONCE]
[INTO <buffer>]
[TRIGGER ACTION <command>]
```

Given this grammar, the user can issue queries to the TinyDB system that are forwarded on the minimal spanning network tree (TAG tree [155]) to all nodes.

Listing 2.2: TinyDB Example Query

```
SELECT nodeID, nestNo, light
FROM sensors
WHERE light < 1300
EPOCH DURATION 1s
```

In Listing 2.2, we show an example query to get the bird nests that have a luminance under 1300lx and hence are suspected to be protected by the mother bird at the moment. The epoch durations is given in seconds. If the sampling rate is one hertz, we can denote this query a non continuous query. In this example the sensor network sends the tuples of sensor nodes 1 and 3 of epoch 1 to the data sink. Accordingly, the nest 17 should be protected at the moment of the query. For other more complex examples we refer to Madden et al. [156].
Comparing TinySQL to SQL, there are significant differences in the power of the query languages, i.a.:

- TinySQL only allows to access only the table *sensors*.
- Arithmetic operations are limited to basic operations.
- Subqueries are not supported.
- The relational scheme can not be updated (no ALTER-Clause).
- No boolean operators or string comparison in *HAVING- and WHERE-Clauses* are supported.
- TinySQL provides a *SAMPLE-PERIOD-Clause* as a basic continuous query concept, that allows triggering sampling times.

- TinySQL provides a *TRIGGER-Clause* to trigger tasks on events and after receiving results respectively.
- TinySQL provides materilization using the *INTO-Clause*.

As denoted previously, the TinyDB framework is known for efficiently evaluating aggregation queries. The aggregation is done early in the network using the minimum spanning tree of the TAG topology [155]. The approach is in general similar to the Cougar approach but works on a tree topology rather than on clusters in the network. In the example query in Listing 2.2 we absent from using aggregation queries for simplification. We discuss the aggregation process in Section 2.2.2.

Other Data Acquisition Approaches

Beside the Cougar and TinyDB approach there has been other query interfaces or data-centric programming concepts for wireless sensor networks presented. A complete overview is out of the scope of this work. We therefore refer to the data management survey papers published in [6, 31]. In this paragraph, we shortly introduce two well known concepts SwissQM [168] and DSN [43].
SwissQM [168] has been introduced to allow using existing query languages for data acquisition in wireless sensor networks. Moreover, it is a data-centric programming concept that extends the capabilities of TinyDB by allowing data cleaning and virtualization. Therefore, queries are translated into compact, optimized virtual queries. These are further translated in *network queries* that are pushed into the network as byte code. This translation approach has similarities with the Cougar translation approach as introduced previously. SwissQM uses a tree topology like TinyDB to disseminate the queries in the network. The sensor nodes are implemented as stack-based virtual machines. The instruction set is a subset of the Java virtual machine that has been extended with more powerful operations to reduce the actual program size. It contains 59 instructions/operations. Because of the independence between programming and query language SwissQM does not define a real data model. E.g. if a user uses XQuery to issue a query to the SwissQM deployment this does not mean that real XML is accessed in the network. SwissQM only provides the capability of using a standard query language and then translates the query according the instruction and operation set of the virtual machine. As an entire framework, SwissQM is more a programming concept than a query interface. Nevertheless, SwissQM also includes the TinyDB aggregation capabilities as extensively introduced in [168].
Another approach that is related to data acquisition is the Declarative Sensor Network (DSN) System [43]. Like SwissQM, this approach can be more classified as data-centric programming concept rather than a traditional query language. DSN includes a programming language, compiler and runtime system to support declarative specification of wireless sensor network applications. The programmers should be encouraged to focus on program outcomes (what a program should achieve) rather than implementation (how the

program works) [43].

Both approaches show that in the past the concepts for programming sensor networks and accessing sensor network data can not be exactly separated. There exist many cross layer approaches and the boundaries between data-centric data acquisition and programming are fluid. The XML data management concepts of this work are focussed on real data management and query evaluation. By using the frameworks of this work, we provide a query layer and an in-network data storage layer that supports the idea of the sensor network as a database. Hence, our approach is more related to TinyDB, Cougar and StoneDB. Nevertheless, as motivated in the introduction of this work, this thesis is part of the project AESOP's TALE that tries to apply the service oriented paradigm to sensor networks application engineering. This idea, that is discussed in more detail in Section 2.4.4, targets a simplification of the engineering process like the data-centric programming frameworks SwissQM and DSN. Nevertheless, a complete evaluation of these concepts is not the topic of this work. For an overview on the actual evaluation of these diverse programming concepts we refer to [151].

2.2.2 Query Optimization

As denoted previously, evaluating queries in wireless sensor networks has to fulfil strict efficiency rules to support a lifetime enhancement of the entire network. In the recent years, several query optimization strategies to allow energy efficient evaluation have been presented. Since the energy demand for communication is significantly higher than the energy demand of a processing microcontroller, most concepts focus on avoiding extensive communication. The general strategy is to push in-network data processing deep in the network to reduce the actual size of result data that has to be transmitted to the data sink. Besides, general optimization strategies known from traditional information systems are applied, e.g. pushing high selective sub queries and projections in the network to reduce the temporary result set. Traditional concepts on optimizing query evaluation are i.a. discussed in [46, 83, 251]. New concepts that have been especially designed for wireless sensor networks include i.a. acquisitional query processing, multi-query optimization, in-network aggregation and model-driven queries. An overview on energy efficient query processing in wireless sensor networks is hereby given i.a. in [5, 119]. In this section, we shortly introduce the most important approaches. A recent approach is using data caches in wireless sensor networks. In this thesis, we introduce a complete data caching framework in Chapter 6. For better understandability, we discuss related work on data caching in that chapter.

The term of *acquisitional query processing* has been coined by the TinyDB project [156]. The basic idea of acquisitional query processing is to control when and where and with what frequency data is collected, as well as which data is delivered. This approach basically tries to optimize the energy demand for sensing. While this energy demand varies for different types of sensors, e.g. complex chemical analyzations consume more energy than simple temperature sensing, acquisitional query processing can result into a significant energy conservation. This idea is in contrast to traditional database systems where data is mainly provided a priori. Acquisitional query processing includes two main challenges: How should the query be processed and the sampling rate be adapted and which samples should be transmitted? In their approach, Madden et al. adapt the sampling rate by a given sampling operator in the query [156]. Queries are moreover ordered in a power-optimal way, e.g. a sampling frequency can determined by a query with the highest frequency. To reduce the data that needs to be delivered to the data sink, the approach tries to exploit spatio-temporal correlations between sensor samples. If two samples are highly correlated only one needs to be sent.

A topic that is related to acquisitional query processing is *model-driven query* evaluation. Model-driven queries embed stochastical assumptions and models that are used to estimate or filter query results in order to reduce sampling for data or sending unnecessary query results to the data sink. An initial approach for model-driven queries has been discussed in [55]. A more complex approach on efficiently evaluating continuous queries in general information systems and sensor networks has been introduced in our work published in [97, 132]. We therefore give an extended overview on this approach in Section 5.3.4.

Many query approaches assume a dedicated gateway node that is used for inserting queries into the sensor network. Multiple queries that are inserted into the network are managed from outside with a given query *id*. For larger scale networks it is desirable to also accept multiple gateway nodes where queries are inserted at different local positions. While the problem of having duplicate *id*s can be solved by attaching the gateway *id*, a high amount of active queries can significantly reduce the lifetime of the entire network. Since many queries are correlated, e.g. multiple queries for retrieving the temperature over various time spans, optimizing the evaluation by merging queries and query results can be an efficient option. A critical discussion on this topic can be found in [225].

In-network aggregation can be denoted as the main research topic in sensor network query optimization. According to the previously mentioned idea of optimizing the communication demand, in-network aggregation is used to significantly reduce the result set that needs to be transmitted to the data sink by merging separate tuples deep in the network instead of processing them outside at the data sink. The origin of the idea of in-network aggregation can not be determined identically. However, it has been included in the next century challenges published by Estrin et al. in [66] early in 1999. The most important publications in this area are Cougar [253] and TinyDB [156] and TAG [155] respectively. Beside these approaches, till now there have been a multitude of publications on aggregation in wireless sensor networks. A complete overview is therefore out of the scope of this paper. Recent approaches can be found i.a. in [116, 117, 193, 213, 250]. For the rest of this section, we will however shortly introduce the in-network aggregation concept in more detail due to its importance in the field of query processing in wireless sensor networks.

The basic problem of in-network aggregation is that preliminary results need to be aggregated only once. Given a general network with broadcast communication, a preliminary result, e.g. a temperature value, will be received by all neighbours. The neighbours are now responsible to aggregate the value to reduce the amount of data that is sent to the sink. They hereby need to coordinate in order to make sure that the temperature value is not aggregated more than once. Besides, the aggregator needs to make sure that they keep track of the number of participating nodes in a preliminary aggregation result to ensure the correctness of the global aggregation result. Madden et al. denote these aggregation functions as duplicate sensitive [156]. Examples for a duplicate sensitive function are AVG (average) or COUNT.

As denoted previously, TAG uses a minimum spanning tree as a logical routing topology. The positive benefit of this topology is that on the way from the leafs to the sink (root) every child has only one father. This fact implicitly avoids duplicate aggregation. On the other side having only one father node and hence only one route to the data sink produces communication bottlenecks. To reduce the deviation of results in case of breaking communication links Madden et al. therefore introduce the TAGk concept where each node has k fathers and the preliminary results are split in k parts. The aggregation functions are then adapted to the number of fathers as described in [155].

In Figure 2.10, we show a simple example to visualize the efficiency of TAG in optimizing the communication demand. For a given count query that demands the number of nodes in the network every node sends a beacon (or one value) to participate. On the left side the query is evaluated in a traditional way letting the gateway count the actual number of nodes. Every node hereby just forwards the messages of the child nodes. The value that a node itself sends is within the circle. This results in a high message demand of 14 messages. On the right side the query is evaluated using in-network aggregation. As a result, the message demand is significantly lower as preliminary aggregation results are sent to the father nodes without forwarding child messages.

Nath et al. [170] showed that TinyDB and TAG respectively do not perform well in real deployments with non robust communication links as the overhead for recreating the tree topology is very high. Active aggregation queries furthermore are evaluated including a high error in the result due to the bottlenecks in case of a link error. Therefore, Nath et al. introduce a ring oriented communication scheme that does not assume a fixed topology and hence is robust against communication errors. This approach includes an aggregation technique denoted as Synopsis Diffusion. A Synopsis is an alternative representation of

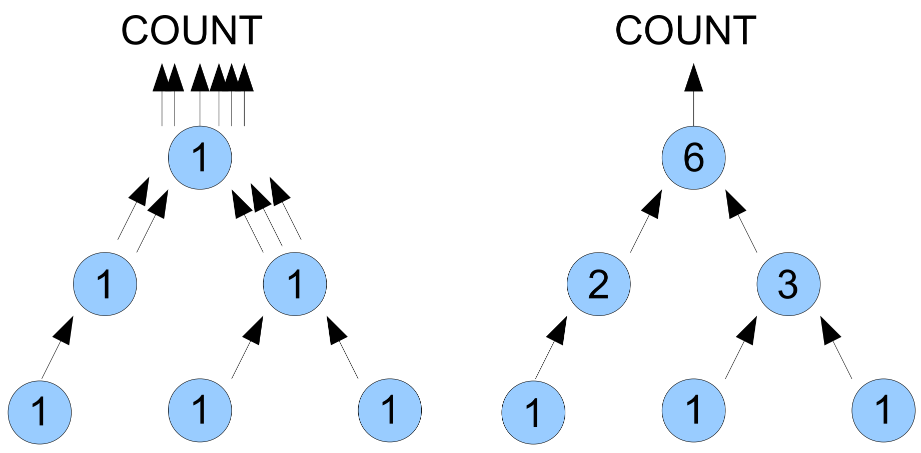


Figure 2.10: TAG Aggregation Process

a query result that is processed in a pseudo duplicate insensitive way. This approach is not really duplicate insensitive in every case as the final result might include a smaller error that can be estimated before. Nevertheless, the error is significantly smaller than the error of TAG results in case of communication errors. Since the routing scheme of Synopsis Diffusion provides a flexible and robust environment, we use this approach for our concepts in Chapter 6.

The concepts on evaluating XML queries on the XML data in the sensor network that are presented in Chapter 5 also include in-network aggregation. Beside the traditional in-network aggregation like described for TAG [155], XML data management allows to have multiple, hierarchical aggregation functions working on subsets of the entire XML documents. In our work, we implemented these functions that provide a more complex form of aggregation queries. The actual evaluation is thereby done using the concepts of TAG [155] and Synopsis Diffusion [170]. Which evaluation protocol is processed can be adapted to the expected robustness of the deployed network, e.g. Synopsis Diffusion for non robust outdoor deployments.

2.2.3 Limitations of Existing Data Management Approaches

In the last sections, we introduced the most important existing concepts in wireless sensor data management. However, as we denoted in the introduction of this thesis these concepts have significant limitations when it comes to support heterogeneity and standardization on application level, e.g. the TinyDB framework only supports a simple relational scheme and a non standardized query interface.
We summarize the most important limitations of the existing data management approaches as follows:

- *Simple data models*, e.g. one table per network or unstructured byte code.

- *Limited heterogeneity*, e.g. different networks with different schemes need further assistance to operate together.
- *Limited interconnection* to existing networks, e.g. direct embedding wireless sensor network in the World Wide Web using standardized protocols.
- *Non-standardized query language*, e.g. new languages need to be learned, existing program interfaces can not be used.
- *Tedious programming*, e.g. simple changes in the application often require complex updates in the program of the sensor node.

According to these limitations and following the extended motivation in the introduction of this work we suggest the usage of complex data models in wireless sensor networks. Due to it's distribution in existing applications and networks we suggest XML as a standardized data format/model in wireless sensor networks. XML is the de-facto standard format for data exchange in the World Wide Web and in many business applications. It has gained awareness over the last decade. While we give a detailed introduction into XML in the next section, we summarize the benefits of XML motivated by [217] as follows:

- *Standardization*: XML is a W3C standard.
- *Self-description*: XML documents are self descriptive and no additional schema needs to be stored.
- *Extensibility*: XML Documents can be easily extended to new environments and changes in the data scheme.
- *Structured Nature*: XML Documents are hierarchically structured and can embed and represent the structure of a sensor network.
- *Provides a 'one-server view' for distributed data*: Represent the sensor network data following the global sensor network approach [2].
- *Simplicity and Machine Readability*: XML Documents can be easily understood by both humans and machines. They can be directly processed in machine work flows using machine-readable context information.
- *Facilitates the comparison and aggregation of data*: The tree structure of XML documents allows documents to be compared and enables the adaptation of aggregation concepts element by element.
- *Support of multilingual documents and Unicode*: More heterogeneity by allowing internationalization of applications.

- *Standardized Query Languages*: E.g. XPath is already supported by many programming frameworks and known to even non-expert users.
- *SOAP is XML*: XML is the basis for the Simple Object Access Protocol (SOAP) to enable standardized service oriented architectures.

As a result, we believe that XML is the ideal standard data exchange format for general wireless sensor networks. Nevertheless, there are limitations and problems that need to be resolved when integrating XML data management in sensor networks that have scarce resources. We discuss these limitations in detail in Section 2.3.5. This thesis provides concepts and frameworks to meet these limitations and furthermore enhance the possibilities of dynamic data acquisition in wireless sensor networks.

2.3 Complex Data Models and Data Management

As motivated in the introduction of this thesis, using complex data models in wireless sensor networks optimizes the heterogeneity, exchangeability and integration between and into other (sensor) networks. In the last section we shortly introduced XML as the de-facto standard data exchange format that includes many benefits for wireless sensor networks. To understand the concepts of this thesis for efficiently integrating XML data management in wireless sensor networks, we discuss several aspects of XML in the subsequent sections.

2.3.1 XML: History and Concept of the eXtensible Markup Language

In this section, we give a historical overview and a brief technical introduction to the e**X**tensible **M**arkup **L**anguage (XML). In the following sections, we further introduce the concepts of XML schema definitions, XML query processing and XML data binding.

History of XML

The history of XML reaches back to the end of the 1960s when the first concepts of a descriptive markup language were introduced. In these years, companies started more and more using workstations for processing large scale documents. While the first applications used binary storage formats that could only be interpreted by the computer itself, there was a need for a data representation format that was self-descriptive and human-legible.
This data format was introduced as the Generalized Markup Language (GML) that has been developed by Charles **G**oldfarb, Edward **M**osher and Raymond **L**orie in the beginning of the 1970's at the IBM research labs.
The general concepts of a markup language are the annotation of textual information to express meta-information about the semantic and structure of the text itself. The origin of the term markup thereby goes back to traditional press where editors where *marking*

up manuscripts in order to highlight errors and remarks on the text and to give typing and layout instructions for the print phase.
In the following years, the Generalized Markup Language was adopted by IBM to be included in the corporation own publishing systems to manage and produce technical documentations. The first standardization process of GML started in 1978 by the American National Standards Institute (ANSI) to find a nationwide information exchange data format. Charles Goldfarb worked on this group and the first industry standard denoted as GCA 101-1983 was adopted in 1983. This standard was further improved by the International Organization for Standardization (ISO) that joined the working group in 1984. In 1985, the first standardization reference draft was published and in 1986 the newly denoted Standardized Generalized Markup Language (SGML) became an ISO standard (ISO 8879) under the full name of "ISO 8879:1986 Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML)". The naming of SGML is often referred to be confusing, as SGML is not itself a markup language but a specification to design markup languages. Well know markup languages are i.a. TeX (1977) [127] and LaTeX (1980) [136].
With the development of the World Wide Web (WWW) and the idea of hypertext SGML's possibly most known application was introduced: HTML (Hypertext Markup Language) [14]. HTML defines a set of markups / tags that are used for defining the representation of web pages. The original idea was hereby to separate the content from the presentation. The presentation tags need to be interpreted and rendered by the web browser. The problem is and was that there is a large space for interpretation, so that the presentation of a HTML document could look totally different using different web browsers. Web designers however did not like that their web pages are represented in different ways and therefore HTML got extended with functions to force representations, e.g. the <font> tag. This progress however conflicts with the initial idea of separating content and presentation. Furthermore, HTML was started to be used in a loose way, e.g. leaving closing tags out, as the web browsers were anyway accepting the HTML documents. The unofficial introductions of more functions, e.g. for multimedia and animation, which was often supported by the two main competitors for web browsers Netscape and Microsoft finally broke the initial concept of a simpler SGML version for web presentation.
These problems showed that HTML is too limited for many application fields and moreover not strict enough. SGML however was always denoted to be too complex for non expert users. Accordingly, in 1996 Jon Bosak from Sun, Tim Brag and Michael Sperberg-McQueen and others introduced the eXtensible Markup Language that is analogously to SGML rather a specification to define markup languages without being too complex. The first specification (working draft) of XML was only 26 pages while the SGML specification was more than 500 pages long. Finally, in 1998 XML was approved in Version 1.0 by the World Wide Web Consortium (W3C) and became a standard. One of the first applications of XML was the representation of HTML denoted as XHTML [184]. XML is today the de-facto standard in many domains of data exchange and data management. It comes with

many benefits for exchangeability and readability. The working group itself had announced the goals of internet usability, SGML compatibility, stability, formality, conciseness, legibility, ease of authoring and minimization of optional features that were reached in the version 1.0 specification [230]. With all the further benefits we introduced in Section 2.2.3 it is also the first choice of data format for wireless sensor networks. We therefore give an introduction into the XML specification in the next paragraph.

Basic Concepts of XML

Due to the space limitations of this thesis, in this paragraph we only give a brief introduction into XML. A formal extensive introduction can be found in the W3C's XML specification [230]. Besides, there exist many publications on XML, e.g. online tutorials [49] and books [239].

The basic concept of eXtensible Markup Language (XML) is a standardized syntax to enable data exchange between different applications and systems. XML documents are represented in a textual form to support human readability. Furthermore, the data formats of database systems and computer systems may be often incompatible. With its textual representation, using XML enables storing data independently of the actual used software and hardware. Exchanging data between different applications or incompatible systems is simplified. XML documents can be used by their simple text format and their self-describing appearance even when exchanging platforms where other data formats need to be firstly converted. XML usage thus increases the availability of data since any hardware, software and applications supports this language.

An XML document is structured starting with the document root followed by a variable nesting of elements that themselves may include attributes. Thereby each element is always enclosed by tags. The start of an element is marked with a start tag, e.g. <sensor>, and the end of an element is represented with a closing tag, e.g. </sensor>. Unlike HTML, every starting tag needs to have a corresponding closing tag. The spelling of the tag (element) names is case sensitive. Elements can contain simple text or other elements (sub elements) or a mixture of them. Correct nesting is mandatory, e.g. a closing tag of a sub element is not allowed to appear after the closing tag of the surrounding element. Elements that do not contain contents are empty elements that can be represented by an abbreviation, e.g. <sensor/>. The name and the content of an element define its type. The type of elements of the document is defined by the user using i.a. type definitions as described in Section 2.3.2.

As denoted previously, beside the textual and sub element content of an element, elements can embed attributes that are defined within the starting tag after the element name. Attributes are used for specifying additional information about an element. They appear

within the starting tag of an element, e.g. <sensor brand = "Pacemate">. An element may have arbitrarily many attributes in arbitrary order. However, attributes of an element may not have the same name. Attributes can always be replaced by elements, e.g. <sensor><brand>Pacemate</brand></sensor>. This often makes it difficult to know when to choose attributes because attributes do not really extend the expressiveness of XML documents. Nevertheless, they are a compact form of representation and can be compressed in an even more compact form as we will discuss in the next chapters.
Finally, XML documents may include other content types, e.g. processing instructions (<?Target-Name Parameter ?>) and comments (<!– comment text –>). Due to their lower importance for representing the actual information, we refer to [230] for a complete introduction.

In the following example in Listing 2.3 we present an example XML document that summarizes all of the introduced concepts. The document represents a sensor network formed by Btnode sensor nodes. Following the global sensor node approach every deployed Btnode embeds itself with its current battery and sensor status.
While being a simple example, this document was actually used and produced in the evaluation of this work by a deployed Btnodes sensor network. It represents the result of the common task of status delivery as described in Section 4.4.3. The XML document consists of the root element *<btnodes>*. *<btnodes>* embeds the actual status notifications of the Btnodes. Each Btnode is hereby represented by the tag *<btsysinfo>* including its id as an attribute, e.g. <btsysinfo id="0"> identifies the status information of the Btnode with *id 0*. The actual status information is given by the tags *<bat>* and *<sens>*, e.g. the Btnode with id=0 has a battery level of 3 and a sensor measurement value of 21. While representing information, an XML document does not include any rules what to do with it. An application to read the values and interpret them is therefore needed.

Listing 2.3: Example XML Document

```
<btnodes>
  <btsysinfo id="0">
    <bat>3</bat>
    <sens>21</sens>
  </btsysinfo>
  <btsysinfo id="1">
    <bat>5</bat>
    <sens>18</sens>
  </btsysinfo>
</btnodes>
```

Given the hierarchical structure of an XML document, we can define hierarchical conditions for elements. If an element *a* is a sub element of element *b*, we call it *child of b*. In this case *b* is the *parent of a*. In our example in Listing 2.3 *<btsysinfo id="0">* is the *child* of *<btnodes>* and analogously *<btnodes>* is the *parent* of *<btsysinfo id="0">*. Elements with the same parent are denoted as *siblings*, e.g. *<btsysinfo id="0">* and *<btsysinfo id="1">* are siblings. A recursive child is defined a *descendant*, e.g. *<bat>* is a descendant of *<btnodes>*. Vice versa, *<btnodes>* is a *ancestor* of *<bat>*. With this given hierarchical structure every XML document can be represented as a tree. We show the resulting tree representation of Listing 2.3 in Figure 2.11. The hierarchical ordering is very important for processing the document, e.g. for evaluating XPath queries as described in Section 2.3.3.

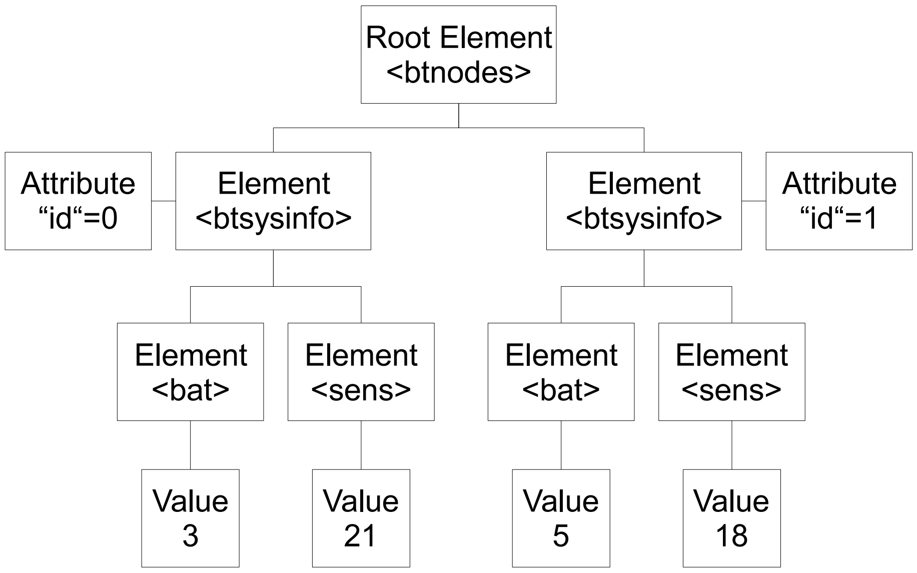


Figure 2.11: Tree Represenation of the XML Document of Example Listing 2.3

For any given XML document there are conditions that need to be reviewed. An XML document first may start with a correct prolog that was left out in Listing 2.3 for better readability. A prolog is an optional component that appears before the root element and consists of the XML declaration and a schema declaration which we will discuss in the next section. The XML declaration embeds information about i.a. the character set and the version regarding the XML specification, e.g. *<?xml version=“1.0“ encoding=“iso-8859-1“?>*. The defined character set needs to be used and no special syntax characters, e.g. "<,>, &, ...", are allowed except in the markup-delineation roles. Following the prolog only one unique root element is allowed. As introduced previously, a root element and every other element then may contain sub or child elements as well as text data in an arbitrary but correct nesting. Attributes of an element have to be named unambiguously and their values have to be textual / flat. Elements as attributes are not allowed. If all of the conditions are fulfilled, the XML document is defined to be *well-formed*. The well-formedness rules define the syntax of XML documents and exclude all violating text documents as being not XML. The rules are the firsts to be checked by any XML processor, as used by the programming framework $XOBE_{SensorNetworks}$ which we introduce in the next chapter.

2.3.2 XML Schema Definition

In the last subsection we introduced the well-formedness as a weak condition for syntactical correct XML documents. However, application and networks that exchange XML documents need more information about the structure and semantics of the processed XML documents. Schema languages have therefore been introduced to define rules on the actual semantic contents of XML documents. If an XML document is well-formed and follows the rules and conditions defined by a certain schema, we denote this XML document to be *valid*. In the following paragraphs we introduce the two most common schema languages Document Type Definitions (DTDs) and XML Schema. While there exist other schema languages, e.g. RelaxNG [44], for a complete overview we refer to the XML literature, e.g. [138], due to their lower importance in the field of XML data management.

Document Type Definition (DTD)

A schema language that has been already introduced for SGML are Document Type Definitions (DTDs). DTDs allow to define a semantical structure of valid XML documents. This includes the elements that are usable and the states and relation in what they may be applied. While the original SGML DTDs are more complex, a reduced and simplified version has been included in the official W3C XML specification [230]. With a given XML document and the corresponding DTD a parser is able to check the semantic and structural conditions to proof if a document is valid or not. DTDs can hereby be embedded within an XML document or stored outside the XML document by using a reference in the document prolog. A DTD consists of declarations of element types, attribute list, entities, notations, comments and processing instructions, that reflect the general structure of XML documents as introduced in the last subsection. Entities are hereby used for abbreviation purposes and notations may be used to reference non-XML data in an XML document, e.g. associate images with a system renderer. In Table 2.4, we summarize the key features of a DTD.

Markup Declaration	Function
<!ELEMENT sensor ...>	element type declaration
<!ATTLIST sensor ...>	attribute list declaration
<!ENTITY % brand "Pacemate">	entity declaration
<!NOTATION exampleSVG SYSTEM "sensor1.svg">	notation declaration
<!-...->	comments
<? ... ?>	processing instructions

Table 2.4: DTD Markup Declarations

In Listing 2.4, we show an example DTD which describes valid XML documents like the one in Listing 2.3.

Listing 2.4: Example DTD

```
1 <!ELEMENT btnodes  (btsysinfo)*>
2 <!ELEMENT btsysinfo  (bat, sens)>
3 <!ATTLIST btsysinfo id ID #REQUIRED>
4 <!ELEMENT bat  (#PCDATA)>
5 <!ELEMENT sens  (#PCDATA)>
```

If the DTD should be embedded within the XML document, it has to be placed in a DOCTYPE environment as shown in Listing 2.5.

Listing 2.5: Example XML Document with embedded DTD

```
<?xml version="1.0"?>
<!DOCTYPE btnodes [
<!ELEMENT btnodes  (btsysinfo)*>
<!ELEMENT btsysinfo  (bat, sens)>
<!ATTLIST btsysinfo id ID #REQUIRED>
<!ELEMENT bat  (#PCDATA)>
<!ELEMENT sens  (#PCDATA)>
]>
<btnodes>
  <btsysinfo id="0">
    <bat>3</bat>
    <sens>21</sens>
  </btsysinfo>
  <btsysinfo id="1">
    <bat>5</bat>
    <sens>18</sens>
  </btsysinfo>
</btnodes>
```

Both examples include the key features of DTDs. The content models of elements can be defined as shown in Listing 2.4 Line 1 using regular expressions. We here define the element type <btnodes> and determine which and how many child elements are allowed for the element type. The star operator * determines that an arbitrary number of children of type <btsysinfo> are allowed. If at least one child element has to appear in a valid document, we could signal this using the plus operator + instead. Using the question mark operator *?* would signal that non or exactly one child has to appear in the valid document. If no additional operator is used like in Line 2, the child elements have to appear exactly once. Furthermore this line shows how to define lists of child elements. If two child elements are separated using | either the first or the second element has to appear in a valid document. Additionally the content model definition of DTDs allows to define empty content models, e.g. *<!ELEMENT btnodes EMPTY>*, and arbitrary content,

e.g. *<!ELEMENT btnodes ANY>*. Like described before, elements can embed textual information instead of child elements. This content model is defined using the *#PCDATA* operator (ParsedCharacterData) like shown in Line 6. Parsed character data is processed by any given parser which is important when using entity references. The opposite of *#PCDATA* is *#CDATA* which signals that the textual information should be processed as defined without any interpretation. Definition of textual content and child elements can be furthermore mixed, e.g. *<!ELEMENT btsysinfo #PCDATA | bat | sens)*>*.
In Line 5, we introduce the definition of attributes. E.g. the attribute list of *<btsysinfo>* is defined by naming the element followed by a list of possible attributes. Each attribute in the list is defined by three parts: the name, the type and a standard value. We present possible attribute types in Table 2.5. The standard value of attributes can be #REQUIRED, signaling the duty to use the attribute in an element, #IMPLIED, signaling that the attribute is optional and no standard value exists, or a fixed non-optional value starting with the key operator #FIXED. Following our example in Listing 2.4, we define that the element *<btsysinfo>* has to include a required ID.

Attribute Types	**Function**
$CDATA$	character data
$(v1\|v2\|..)$	logical OR of attribute types v1, v2, ...
ID	unique ID
$IDREF$	a reference ID of another element
$IDREFS$	a list of IDs of other elements
$NMTOKEN$	a valid XML name
$NMTOKENS$	a list of valid XML names
$ENTITY$	an entity
$ENTITIES$	a list of entities
$NOTATION$	a notation
$xml:$	a pre-defined xml value

Table 2.5: Attribute Types

As shown in this paragraph, DTDs easily allow to define languages for valid XML documents. In this thesis, DTDs are supported and used by the $XOBE_{SensorNetwork}$ precompiler to analyze the expressiveness of given XML documents and XML document generators as discussed in the following chapters. By knowing the language of a given DTD and hence the structure of possibly generated XML documents the actual task of XML data compression can be significantly optimized. This task can be even more optimized when further information, e.g. types of values, is provided by the schema [244], e.g. knowing if a parsed character data represents a string or just an integer number. While being a simple and efficient schema language, DTDs show their drawbacks when it comes to complex type systems and other features. As introduced before, DTDs only support two basic textual types: #PCDATA for elements and #CDATA for attributes. Another drawback is that DTDs do not support scoped declarations, e.g. all definitions are global. We therefore introduce a more powerful schema language *XML Schema* in the next paragraph to give a complete background on the discussed optimizations.

XML Schema

XML Schema is an XML-based alternative to DTDs to describe the structure and semantic contents of XML documents. XML Schema was introduced as a working draft by the W3C in the year 2000. It was approved as a W3C Recommendation in May 2001 and a second edition incorporating many errata was published in late 2004 [233]. The strengths of XML Schema are firstly that it is valid XML and secondly that it is more complex than DTDs as discussed in the last paragraph. An XML Schema is also often referred to as an XML Schema Definition (XSD). We will discuss some of these extensions in the following part of this paragraph.
One of the biggest advantages of XSDs in contrast to DTDs is the more complex type system. In global, there is a differentiation between *ComplexTypes* and *SimpleTypes*. ComplexTypes include the types Attribute and Element as known for DTDs. While in general SimpleTypes and ComplexTypes are classes for new user defined types they can be compared to the types of DTDs. SimpleTypes are the analogon to #PCDATA and #CDATA including elements with values, attributes and restrictions. They further include a detailed differentiation of the actual type of the content. XSD supports the following types of atomic values: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time. Elements can be defined using the *xs:element* markup followed by a name definition and the simple type definition, e.g. *<xs:element name="bat" type="xs:integer"/>*. As denoted in the end of the last paragraph, this extended type system i.a. allows to optimize data compression, e.g. a 16 digit number needs 128 bits in textual representation but only up to 64 bits in integer representation. Like for DTDs, XSDs allow to define standard values and fixed values by using the key words *default* and *fixed* in the *xs:element* markup. Attributes can be defined analogously to elements, e.g. *<xs:attribute name="id" type="xs:integer"/>*.
The values of element can be restricted using the restriction rules of Table 2.6. In Listing 2.6 we give an example. We define the element *<bat>* that embeds an integer value that is restricted from 0 to 5. Using the restriction rules can hereby optimize the compression of this element.

Listing 2.6: Example XSD

```
<xs:element name="bat">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="5"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restriction Rules	Function
enumeration	define an enumeration of allowed values
fractionDigits	number ($>= 0$) of post-decimal positions
length	number ($>= 0$) of allowed characters
maxExclusive	maximum exclusive value
maxInclusive	maximum inclusive value
maxLength	maximum number ($>= 0$) of characters
minExclusive	minimum exclusive value
minInclusive	minimum inclusive value
minLength	minimum number ($>= 0$) of characters
pattern	define pattern of allowed characters
totalDigits	exact number ($>= 0$) of digits
whiteSpace	space handling

Table 2.6: Restriction Rules

After discussing the basic concepts of XML and introducing schema languages to define valid XML documents concerning semantic and structure, in the next section we discuss how data can be accessed. We therefore summarize available query languages and discuss the XML query language XPath in detail.

2.3.3 XML Query Processing and XPath

Wireless sensor networks are deployed for data acquisition. Providing a flexible query layer is therefore crucial for XML data management in wireless sensor networks. In the past, many concepts and languages for XML query processing have been introduced. The most important languages that have been recommended by the W3C are XPath [234], XQuery [235] and XSLT [231]. While there exists a large set of publication on XML query languages and query evaluation, a complete introduction is out of scope of this thesis. We therefore refer to surveys [19, 220] and books [26, 120, 158]. Besides, existing XPath and XQuery engines like Galax, MonetDB/XQuery, Oracle, Saxon, Tamino XML Server and Microsoft's SQL Server 2005 Express [237] are limited in their usability in wireless sensor networks. Either the program or runtime memory demand exceeds the capabilities of today's sensor nodes or they make use of non-supported object models like described in [98]. Theoretical studies on the complexity of XPath can be found in [79].
XQuery 1.0 and XSLT 2.0 use XPath 2.0 for navigating and accessing the XML document. The term navigation is hereby related to moving through the tree representation of the XML document and selecting nodes that represent the navigation (query) result. XQuery 1.0 is a functional programming language. In contrast to XPath, XQuery allows user defined functions, the transformation of nodes, the creation of new structures using XML templates, libraries of functions and a way to bind values to variables. XQuery was designed to provide for XML databases and document collections what SQL does for relational data stores. It includes a FLWOR (for-let-where-order-by-return) clause that can be compared

to the SELECT-FROM-WHERE clause of SQL. XQuery is a superset of XPath implying that every valid XPath expression is also a valid XQuery expression.
On the other side, XSLT 2.0 has been introduced for transforming XML documents from one format to another, e.g. to present XML data as an HTML document in a web browser. As denoted previously, also XSLT 2.0 uses XPath to navigate the input document. XSLT queries / programs are denoted stylesheet documents. In contrast to XQuery, XSLT stylesheets are valid XML on their own. An XSLT processor processes a given XML document and a stylesheet and produces a new XML document as the transformation result. How the input and the stylesheets are specified depends on the processor implementation.
XQuery and XSLT share a lot of functionalities. Approaches to transform XQuery to XSLT and vice versa are discussed in [85]. A comparative analyzation is discussed in [19]. Generally, XSLT is preferable when most of the input is processed from one document and when the output is generated for human consumption, e.g. presentation of data on a web page. XQuery is preferable when processing multiple documents to collect data in one document, which can then again be processed by XSLT for presentation.
Both languages are more complex than XPath but presume the correct and efficient evaluation of XPath queries. In this thesis, we focus on XPath queries because firstly it is powerful enough to express common data acquisition queries and secondly can be easily extended to both languages. In the next paragraph, we give a more detailed introduction to XPath and the document navigation.

The Concepts of XPath

XPath is a specification for an expression language for navigation in XML documents recommended by the W3C [234]. In detail, XPath is used for expressing navigational steps and conditions for selecting nodes in a tree representation of an XML document. XPath 1.0 was introduced in 1999 with a definition for the syntax and the semantics. The newer version XPath 2.0 has been recommended in 2007 including optimizations and an extended functionality. Nevertheless, the basic concepts for data acquisition are the same for both versions. In this paragraph, we introduce these basic concepts and refer to [121] for a complete overview on the extensions of XPath 1.0 and 2.0.
As denoted previously, XPath provides the ability to navigate through a tree representation and selecting tree nodes of different types based on given criteria. The basic concept of XPath includes a differentiation of node types as shown in Table 2.7.
An XPath *expression* is a syntactical construct that represents the following basic types:

- a node set (determined by a location path)
- a boolean value
- a number
- a string

The location path is the most important expression in XPath. It consists of a sequence of *location steps* that traverse through the tree. Each location step consists of three compo-

Node Type	Explanation
root node	the whole XML document (document root)
element node	represent an element
attribute node	represent an attribute
text node	represent the textual element / attribute content (atomic values)
namespace node	represent a namespace
processing instruction node	represent a processing instruction
comment node	represent a comment node

Table 2.7: XPath Node Types

nents: an *axis*, a *node test* and zero or more *predicates*. The general syntax of a location step is *Axis :: Node [Predicate]*. The axis determines the direction of navigation. The node test specifies the node type and the name of nodes selected by the location step. The arbitrary predicates embed XPath expressions to filter the set of result nodes of the current location step.
Location paths can recursively contain expressions to filter the nodes. A relative location path / step is always evaluated in relation to an actual selected node, which is defined as the *context node*. The result of a location path / step is a list of selected nodes, which is defined as *context list*. A context list is then iterated for processing the next location step by testing all nodes of the context list as context nodes. An absolute location path starting with a / is always evaluated using the root node of the document.
The term *axis* denotes the relation of two nodes in the tree hierarchy. They are used to determine the way which should be followed to traverse the tree representation. We summarize the available axes in Table 2.8. The XPath syntax includes an abbreviated syntax and a full syntax for XPath axis which is also summarized in Table 2.8.

Axis	Selected Nodes	Abbreviated Syntax
self	context node	.
child	direct child elements	(default, can be left out)
parent	direct ancestor	..
descendant	all descendant nodes (children and grandchildren etc.)	
descendant-or-self	context node and all descendant nodes	//
ancestor	ancestor nodes (parent and grandparents etc.)	
ancestor-or-self	context node and all ancestor nodes	
following	all nodes that follow the context node in document order	
following-sibling	all following nodes that are siblings of the context node	
preceding	all nodes that precede the context node in document order	
preceding-sibling	all preceding nodes that are siblings of the context node	
attribute	the attributes of the context node	@
namespace	the namespace of the context node	

Table 2.8: XPath Axes

In Figure 2.12, we visualize the axes by using a graphical tree representation. The grey node is the context node. Accordingly, it is also selected when following the self axis. The direct predecessor is selected when following the parent axis. All nodes on the direct path to the document root are selected when following the ancestor axes and so on.

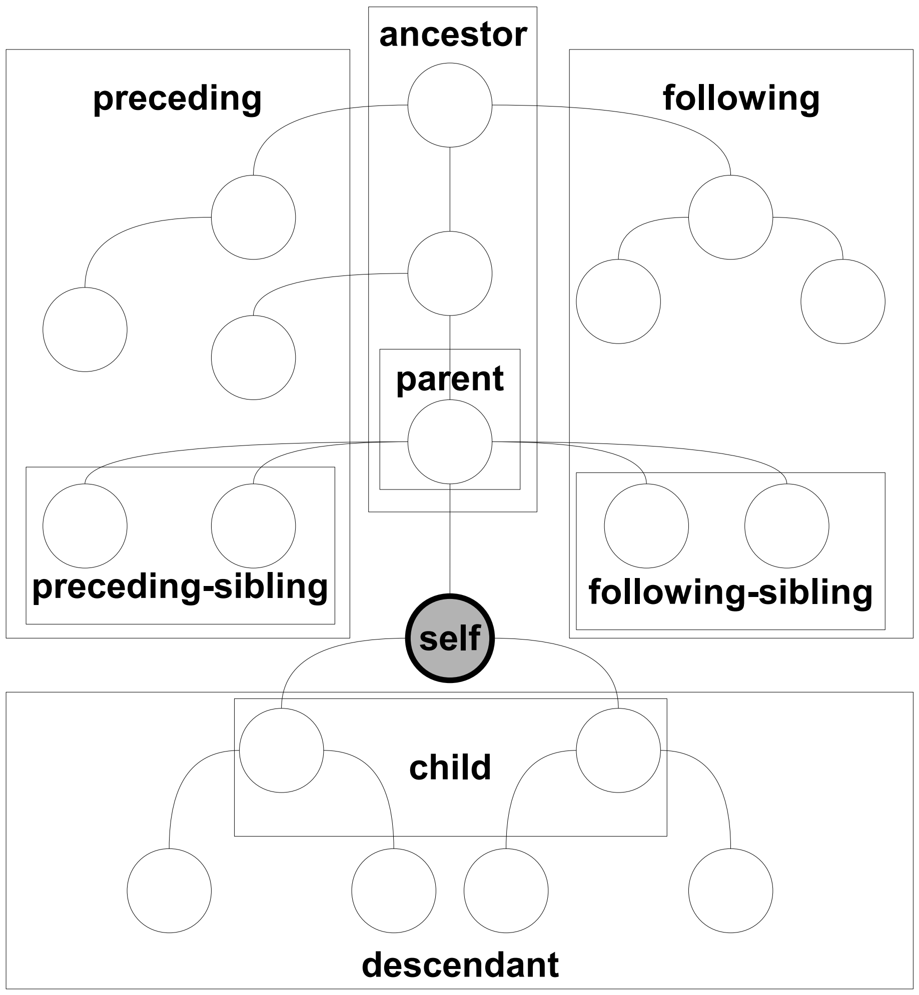


Figure 2.12: Overview on XPath Axes

After following the axis and selecting the corresponding nodes, they are filtered regarding the node type and name by the node test. XPath provides six basic node test functions. If a function results in *true*, the currently reviewed context node stays in the context list. Otherwise it is dismissed. In Table 2.9, we summarize the basic node test functions.

Node Test Function	Explanation
comment()	filters comment nodes
text()	filters text nodes
processing-instruction()	filters processing instruction nodes
node()	demands an arbitrary node type e.g. element or attribute
*	wildcard operator
name	filters all context nodes for a give name

Table 2.9: Node Test Functions

The last part of an XPath expression is a predicate that can embed any XPath expression to filter the current context node list. If the expression in the predicate evaluates true for a given context node it is kept in the context node list, otherwise it is dropped. In the application field of wireless sensor networks predicate expressions can be used to filter results for measurement or attribute requirements, e.g. //sensornodes[@id>4] will return all sensornodes with an id greater four. To summarize the XPath language, we define a simplified grammar for XPath in Table 2.10. For a complete grammar in EBNF we refer to the official W3C specification in [234].

location_path	$\rightarrow$	*relative_location_path*
relative_location_path	$\rightarrow$	*step*|*relative_location_path* **'/'** *step*
step	$\rightarrow$	*axis_specifier node_test predicate*$*$
axis_specifier	$\rightarrow$	*axis_name* **'::'**
axis_name	$\rightarrow$	**ancestor** | **ancestor-or-self** | **attribute**
		| **child** | **descendant** | **descendant-or-self**
		| **following** | **following-sibling** | **parent**
		| **preceding** | **preceding-sibling** | **self**
node_test	$\rightarrow$	*name_test*|*node_type* **'('')'**
predicate	$\rightarrow$	**'['** *predicate_expression* **']'**
predicate_expression	$\rightarrow$	*expression*
name_test	$\rightarrow$	**'$*$'** | *name*
node_type	$\rightarrow$	**comment** | **text** | **node**

Table 2.10: Simplified XPath Grammar

We end this paragraph by showing examples for the presented XPath concepts. We therefore take the XPathMark XML document shown in Listing 2.7 as a basis for the XPath evaluation. XPathMark is a wide spread benchmark for XPath engines which was used in the evaluation of the concepts of this work in Chapter 5. In Table 2.11, we present example XPath queries, the meaning and the corresponding results.

Listing 2.7: XPathMark Evaluation Document

```
<A id="n1" post="26" pre="1" xml:lang="en">
  <B id="n2" post="3" pre="2">
    <C id="n3" post="1" pre="3">clergywoman</C>
    <D id="n4" post="2" pre="4">decadent</D>
  </B>
  <E id="n5" post="22" pre="5">
    <F id="n6" post="6" pre="6">
      <G id="n7" post="4" pre="7">gentility</G>
      <H id="n8" idrefs="n17 n26" post="5" pre="8">happy-go-lucky man
        </H>
    </F>
    <I id="n9" post="9" pre="9">
      <J id="n10" post="7" pre="10">jigsaw</J>
      <K id="n11" post="8" pre="11">kerchief</K>
    </I>
```

```
15     <L id="n12" post="15" pre="12">
16       <!--L is the twelve-th letter of the English alphabet-->
17       The letter L is followed by the letter:
18       <M id="n13" post="10" pre="13"></M>
19       which is followed by the letter:
20       <N id="n14" post="13" pre="14">
21         <O id="n15" post="11" pre="15">ovenware</O>
22         <P id="n16" post="12" pre="16">plentiful</P>
23       </N>
24       <?myPI value="XPath is nice"?>
25       <Q id="n17" idrefs="n8 n26" post="14" pre="17">quarrelsome</Q>
26     </L>
27     <R id="n18" post="18" pre="18">
28       <S id="n19" post="16" pre="19">sage</S>
29       <T id="n20" post="17" pre="20">tattered</T>
30     </R>
31     <U id="n21" post="21" pre="21">
32       <V id="n22" post="19" pre="22">voluptuary</V>
33       <W id="n23" post="20" pre="23">wriggle</W>
34     </U>
35   </E>
36   <X id="n24" post="25" pre="24">
37     <Y id="n25" post="23" pre="25">yawn</Y>
38     <Z id="n26" idrefs="n8 n17" post="24" pre="26" xml:lang="it">
         zuzzurellone</Z>
39   </X>
40 </A>
```

XPath Query	Meaning	Result (Line)
//L/*	All children of all descending-or-self L elements	18, 20, 25
//L/parent::*	All parent nodes of all descending-or-self L elements	6
//*[parent::L]	All nodes that have a parent L	18, 20, 25
//*[@id]	All nodes with an id	All lines with starting tags
//L/comment()	All children of L nodes that of type comment node	16
//*[child::* and preceding::Q]	All nodes that have at least a child and precede Q	27, 31, 36
//*[@pre > 12 and @post < 15]	All nodes with an attribute pre greater 12 and post lower 15	18, 20, 21, 22, 25

Table 2.11: XPath Expression Examples

The Power of XPath for Sensor Network Data Management

After introducing the basic concepts of XPath, we show that the XML data model and the XPath language are mighty enough to represent existing sensor network data management approaches. We therefore give an example based on the previous TinyDB introduction as it is often referred to be the state-of-the-art data management solution. In detail, in Table 2.3 we showed an extension of the one table per network approach of TinyDB. Any given TinySQL query is evaluated on this table. In Listing 2.8, we show the corresponding XML document of this example. While we see that we can translate any TinyDB example to XML, we point out that XML provides a more flexible data management solution. Accordingly, XML documents can not be generally transferred into the TinyDB data model.

Listing 2.8: Transforming TinyDB to XML

```
<nodelist>
  <node epoch="0" nodeID="1" nestNo="17">
    <light>1455</light>
  </node>
  <node epoch="0" nodeID="2" nestNo="27">
    <light>1389</light>
  </node>
  <node epoch="0" nodeID="3" nestNo="17">
    <light>1422</light>
  </node>
  <node epoch="0" nodeID="4" nestNo="25">
    <light>1405</light>
  </node>
  <node epoch="1" nodeID="1" nestNo="17">
    <light>5</light>
  </node>
  <node epoch="1" nodeID="2" nestNo="27">
    <light>1389</light>
  </node>
  <node epoch="1" nodeID="3" nestNo="17">
    <light>8</light>
  </node>
  <node epoch="1" nodeID="4" nestNo="25">
    <light>1405</light>
  </node>
</nodelist>
```

In Listing 2.2, we gave an example for a TinySQL query selecting the bird nests that have a luminance under 1300lx over the next second epoch. This query can be transformed into XPath as shown in Listing 2.9.

Listing 2.9: Transforming TinySQL to XPath

```
//light[.<1300]/parent::node/*
```

While the actual data selection is the same for both representations, the TinySQL query includes an operator for evaluating the query continuously, e.g. over epoch duration 1s. Processing continuous queries in wireless sensor networks is crucial to avoid sending a query over and over again. The existing XML query languages including XPath have no support for continuous queries. However, extensions have been discussed in [22]. Furthermore, any XPath engine on the sensor nodes can repeat the query by attaching the epoch operator to the XPath query. An energy efficient solution for wireless sensor networks has been discussed in our previous work in [97] which we summarize in Chapter 5. Other acquisitional optimizations concerning the sampling of sensor nodes based on the inserted queries [156] and query decomposition based on the localization of data are issues that are currently addressed in our future research and hence out of scope of this thesis.

2.3.4 Benefits of XML Integration in Wireless Sensor Networks

There are three major application areas in the field of wireless sensor networks which may benefit from the use of XML data management: heterogeneous networks, WWW integration and service oriented architectures.

As mentioned in the introduction of this thesis and shown in Figure 1.1, when wireless sensor networks get larger, it is likely that different sensor nodes with different capabilities and different properties are deployed to fulfil a common task. To enable the seamless communication between different types of sensor nodes on application layer, XML is a predestined data format because the developers do not have to think about proprietary data formats or other technical issues like big endian or little endian representations anymore.

Using an XML data management in wireless sensor networks also facilitates the integration of these networks with the WWW because it allows the usage of standard query and transformation languages like XQuery or XSLT to represent sensor data on a webpage. We implemented an application to demonstrate the easy integration as described in [96].

The most sophisticated application of XML in wireless sensor networks is the implementation of a service oriented architecture (SOA) in a wireless sensor network. While there exist several approaches claiming the implementation of a service oriented sensor network [86, 139, 192], these do not use the same techniques known from common service oriented architectures, e.g. SOAP. We discuss service oriented sensor networks in the subsequent sections.

2.3.5 Limitations and Goals of XML Integration in Wireless Sensor Networks

In the introduction of this thesis and in the last section, we motivated the usage of XML data in wireless sensor networks. However, all the advantages of XML like support for heterogeneity, standardization and others come at a price: the textual XML representation of information can be up to 400% of the size of its binary representation [98]. This is especially a problem in wireless sensor networks where scarce resources demand for small data sizes as described in the previous Section 2.1.3. Additionally, it has to be considered that transmitting data is the most energy consuming task in wireless sensor networks. Transmitting one bit can be 1000 times more expensive than computing one operation [155]. This may be one of the main reasons why XML programming and processing has not been integrated into wireless sensor networks, yet.

Another development limitation is the complex task of programming sensor nodes. This is even more demanding for integrating XML in sensor network programming. While recent sensor network applications use only simple data structures, e.g. only lists of sensor values or simple table structures [156], using XML will require consolidated knowledge on how to represent XML with the language constructs of sensor node programming languages like Embedded C. Hence, besides solving the verbosity problem of XML, the XML programming language integration should be transparent and usable without any consolidated knowledge.
The goal of this thesis is to overcome these limitations by compressing the XML data in a way that it is still dynamically processable and to facilitate the XML programming of sensor nodes by employing a programming framework which can guarantee stable programs.

2.3.6 Related Work to XML Data Binding and XML Compression

In this thesis, we describe the $XOBE_{SensorNetwork}$ framework that enables a seamless integration of XML data in the engineering process. Integrating XML within the programming phase and transforming it to the target programming language is often referred to as XML Data Binding. In this section, we summarize the related work in the area of XML Data Binding and name the solutions that are applicable for integrating XML in sensor network programming languages. Due to the space limitations of this thesis, we can not cover all frameworks and solutions. Other solutions that are not applicable for integrating XML in sensor network programming languages are introduced by Bourret in [24].

Existing XML Data Binding Solutions

Many XML data binding techniques to represent and handle XML data within programming languages have been presented in the last decades. Most of these frameworks are developed for existing object-oriented programming languages like Java and C++ [8, 67, 179]. These frameworks are not applicable for sensor network programming, mainly because of the choice of language, the compression ratio and the memory demand itself that is not sufficient to overcome the hardware resource limitations in wireless sensor networks. For the programming language C, as the main choice of language for sensor network engineering, the XML data binding framework autoXML [123] has been presented. However, our previous results in [98] have shown that autoXML is also not applicable for sensor network applications because of a steeply increasing memory demand and unstable runtime behaviour. We will again discuss this behaviour in Chapter 4.

Existing XML Compression Frameworks

Compressing XML data in wireless sensor networks is essential because of its verbosity. The memory limitation on todays sensor node products ranging from 16 kb to 128 kb make an effective uncompressed usage impossible. Standardized textual compression techniques

like zip [87] are not applicable because they require entire XML documents before compressing which can not be guaranteed for dynamically growing XML documents in active wireless sensor networks. A more serious problem is that the documents need to remain dynamically processable, e.g. for evaluating queries on them, which may not be possible without a decompression step in case of regular textual compressors. These problems are reasons why previous data management approaches absent from using complex data formats in wireless sensor networks and remain using simple solutions like the one table per network approach [43, 156, 168, 253]. In detail, in TinyDB [156], data is represented as a table with one column for each attribute which is available in the network.
On the other hand, there exist special compressors for XML data that show the possibilities of adapting complex data formats to resource constraints. They can be divided in the ones that produce compressed XML, that can be queried and dynamically processed [29, 166, 173], and the simple compressors that produce non processable representations of the XML documents [146, 244]. Besides, XML compression has been further discussed in [109, 146, 236]. However as previous investigations in [102] have shown, these compressors are not applicable to sensor network data management because of their significant footprint that easily exceeds the available program memory on sensor nodes and the processing power constraints. The *Xenia* approach [245] is an exception. It reduces the problem of XML compression to the execution of a deterministic pushdown automaton which is generated from an XML Schema definition at compile time. Currently, there is no possibility to evaluate XML queries directly on the Xenia-compressed XML document representations. This makes Xenia mainly useful for an in-network data representation only. However, the results that are presented in this paper can be adapted so that query processing on Xenia-compressed documents will be investigated in the future.
While the presented work on XML compression was not targeted for sensor network usage, the usage of more structured complex data formats in wireless sensor networks was firstly discussed in our work published in [96] and [98]. As a result, using complex XML data and supporting further XML-based application scenarios and protocols, e.g. SOAP [232], is now possible. In Chapter 4, we describe all of our contributions in the field of XML compression that are to the best of our knowledge the only approaches that are applicable for wireless sensor networks. We further discuss the dynamic processability of the compressed XML data and the evaluation of XPath queries on the compressed XML data in Chapter 5.

2.4 Service Oriented Architectures and AESOP's TALE

While previous work has been concentrated on finding energy efficient algorithms for data transmission, data routing and data acquisition in wireless sensor networks, the engineering process has been only slightly researched. Developing sensor network applications remains a tedious and error prone task which can be one of the reasons why wireless sensor networks are often not used by non expert research teams, e.g. biologists.

In detail, programming sensor network applications remains to expert users, e.g. computer and network scientists, due to the specific conditions in sensor networks, e.g. resource limitations, high communication and node failure probability as described in the previous Section 2.1, which require special programming concepts and algorithms. Moreover, the abstraction layer of the programming phase is often close to the hardware of the nodes.

A strategy for simplifying the engineering process can be the adaptation of the concept of service oriented architectures (SOAs) for sensor networks applications which is described in this section. Service oriented architectures have been introduced as a design principle for the system development process of mostly distributed application scenarios within business domains. They simplify and organize the development process by allowing a stepwise composition of atomic, self-describing services into more complex applications, which themselves can be defined as services.
By creating an abstract view on the system functionality and hiding complex implementations under the context of services, non expert users can create sensor network applications without comprehensive knowledge of the target platform. Nevertheless the resource limitations require an adaptation of the existing SOA concepts and standards. This adaptation is the goal of the project AESOP's TALE and this thesis. Hence, in this section we give an extended introduction into service oriented sensor networks and present the contributions of this thesis to the goals of AESOP's TALE.

2.4.1 Introduction to Service Oriented Architecture (SOA) and Web Services

With the introduction of Sun-RPC 20 years ago and the publishing of the RPC (remote procedure call) reference specification [164], middleware technologies to abstract and simplify the engineering process of distributed systems have been widely used.
In detail, middleware describes a software solution to interconnect software components and applications in a mostly distributed application scenario. Hereby, the communication processes between the connected software components are organized and encased in an abstraction layer between the operating system and the actual application. As a result, complex distributed applications that make use of various software components and services in large scale networks can be efficiently developed. The engineering process is further simplified by deriving application specific parts of the middleware from a given interface description which is defined by an Interface Definition Language (IDL) [175]. One of the widespread middleware technologies that includes the described concept is CORBA, which exists for various programming languages [229].
The communication aspects of the remote procedure call that is provided by the middleware are further hidden to the developer. The encapsulation and abstraction of the middleware provides better failure detection and error debugging. As a result, using a middleware can

abstract from the complexity of a distributed system by providing an abstraction level that makes the development process the same as for a non distributed application scenario.
While the ideas and strategies of the middleware technology exist and are used for over 20 years, the recent trends in distributed systems, software technologies and the evolving world wide connection through the World Wide Web (WWW) introduces a new middleware technology denoted as Web Services.
In detail, the Internet is the world largest application network and hence distributed system. The internet describes the world wide interconnection of computers using the standardized Internet Protocol Suite (TCP/IP) which serves billions of participants and interconnects millions of sub-networks. The most common application of the Internet is the World Wide Web (WWW) which describes the interconnection of Hypertext Documents using the standardized Hypertext Transfer Protocol (HTTP).
The large scale network size of the internet which exceeds all previous networks and the loose coupling of client and server participants make existing middleware technologies like CORBA not or only limited applicable for many domains and therefore require alternatives. Hence, the new middleware solutions denoted as Web Services has been introduced in [42] by the W3C. While there is a lack of a precise definition of the Web Service paradigm itself, the difference in the Web Service concept to previous middleware technologies can be summarized by high exchangeability, interoperability using Web document and protocol standards and programming language independence.

In detail, the early middleware concepts and the concept of Web Services both allow implementations of the Service Oriented paradigm also denoted as Service Oriented Architectures (SOAs). Hereby, a service comprises units of functionality and provides a description of how to use it defined by metadata. Services are generally made accessible in order to compose them to high level applications which are defined as consumer or clients of services. This task is also referred to as orchestration. The communication between the consumer and the service provider is defined by standardized protocols as denoted previously.
Previous middleware technologies like CORBA and Java-RMI [59] that allow implementing SOAs use an object oriented data model with a strict coupling to dedicated object oriented programming languages, e.g. C++ and Java. The life-cycle of a service is defined as follows. A service is defined as interfaces in the Interface Definition Language (IDL). In a next step the IDL is compiled to generate client stubs and server skeletons that represent the communication ends between the service provider and the service consumer. The service is implemented on server side, associated with the generated skeletons and published with a naming or trading service to make it available for the client. The client contacts the naming service for the desired service and retrieves the appropriate object reference which can be used transparently by the developer as if it would be a local object reference. The presented life cycle shows the disadvantage of the early middleware technologies. The strict relation between the data model (object model) and the target pro-

gramming language (object oriented programming language) and the location transparency (object references) make the previous approaches inflexible and reduce the exchangeability.

Web Services in contrast, as introduced by the W3C, are designed to support interoperable machine-to-machine interaction over a network. A Web Service has an interface described in a machine-processable format (specifically WSDL (Web Service Description Language)). Other systems interact with the Web Service in a manner prescribed by its description using Simple Object Access Protocol (SOAP) messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards. Services are used by sending and receiving messages following the standardized SOAP message exchange model. In contrast to the previous approaches, a user does not need to know anything about the implementation (object model, programming languages, etc.). Only the rules of SOAP need to be taken care of.

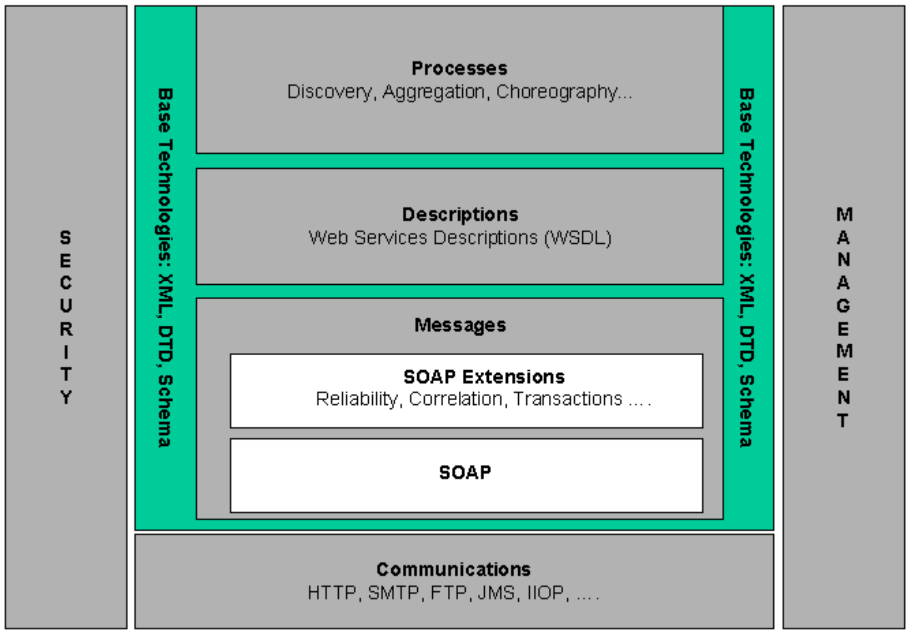

SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography...
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions ....
SOAP
Base Technologies: XML, DTD, Schema
MANAGEMENT
Communications
HTTP, SMTP, FTP, JMS, IIOP, ...

Figure 2.13: Web Service Stack Architecture [21]

In Figure 2.13, we show the general Web Service Network Stack. While a complete introduction is given in [21], we here discuss the general aspects and key features of the Web Service technology. The core of the Web Service technology are SOAP messages that are used for interacting with Web Services. A SOAP message uses the XML data format. It consists of an envelope, an optional header and a body containing the data. Web Services communication includes three types of SOAP messages: SOAP requests, SOAP responses and fault messages. Information, e.g. complex objects and data, can be embedded in SOAP messages in a serialized way, which is determined by the serialization rules for data exchange. Beside these contributions, the W3C specification also includes conventions for representing RPCs.

According to [78], the SOAP message exchange model can be defined as follows. In any case of receiving a SOAP message, the local application must:

1. Identify the parts of the message intended for it.
2. Verify that these parts are supported by the local application and process them accordingly.
3. If the application is not the final destination, the parts of step 1 have to be removed and the remaining message has to be forwarded to its final destination.

Like in the previous approaches, e.g. CORBA, Web Services need to be specified in order to make them public. The Web Service stack includes the Web Service Description Language (WSDL) that is an XML Schema language for describing the interface of a Web Service instance [42]. An interface description i.a. includes the provided methods and parameter types including both response messages and request messages. WSDL furthermore let you describe where a service is located (address) and which protocols on lower layer are used. WSDL hereby does not mandate a specific protocol as shown in the Web Service stack in Figure 2.13. Nevertheless, it is common to use bindings like SOAP and HTTP.
Web Services are not only used in closed environments where every available Web Service is known by all participants. In public networks public directories have to be available to register and lookup services. Therefore UDDI (Universal Description, Discovery and Integration) has been introduced as a registry service, that is actually itself a Web Service, for supporting the publication, search and binding of Web Service instances [238]. UDDI provides mechanisms for service providers to advertise for a service in a standardized form and for service consumers to search for services of interest. We visualize these mechanisms in Figure 2.14.
If UDDI finds a suitable Web Service, it is passed as a reference to the query issuer. The user binds the Web Service instance, e.g. using the WSDL description. The information is exchanged using SOAP messages. While our concepts of this thesis are firstly motivated on integrating plain XML data management in wireless sensor networks, we also support the integration of SOAP messages as a data exchange model. We therefore discuss the SOAP message exchange briefly in the next section.

2.4.2 Programming Web Services Using SOAP

As denoted in the previous section, using Web Services requires a standardized protocol for accessing the Web Services and transmitting the corresponding data. For a detailed introduction into programming Web Services using SOAP we refer to [216].
We give a short example to show how SOAP is used to use Web Services. The example is derived from [49], for a complete introduction we refer to this tutorial and to the book [216]. To use a Web Service, a client sends a SOAP request message as shown in Listing

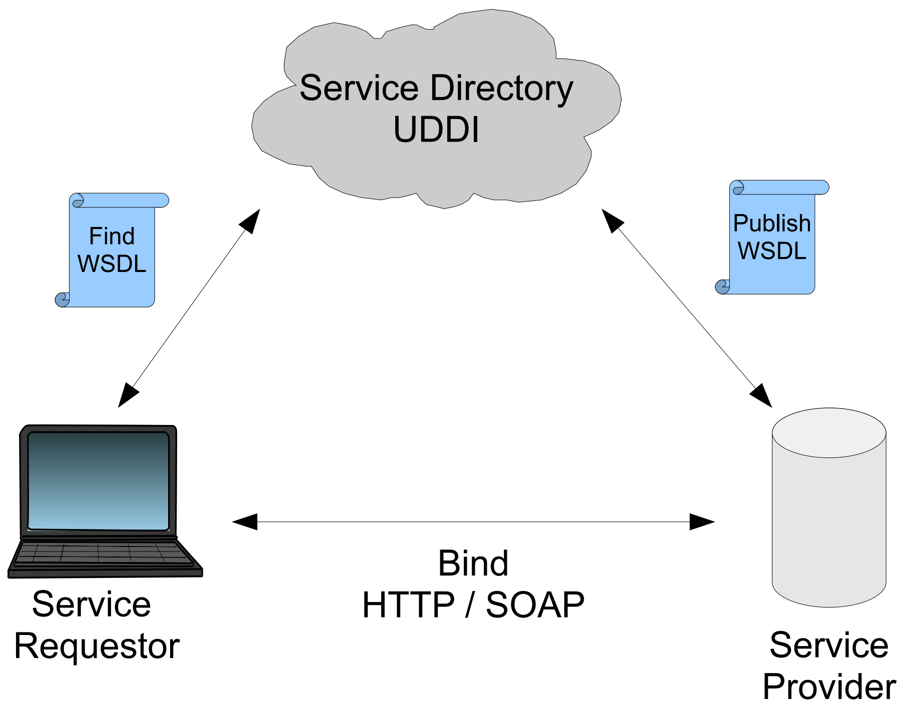


Figure 2.14: Web Service Architecture Triangle

2.10. Beside the header and envelope, the message contains the body including the Stock-Name parameter and a Price parameter that will be returned in the SOAP response. The namespace for this function is defined in "http://www.example.org/stock".

Listing 2.10: A SOAP Request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPrice>
    <m:StockName>CORN</m:StockName>
  </m:GetStockPrice>
</soap:Body>

</soap:Envelope>
```

After receiving the SOAP request the service provider calculates the current stock price for corn market stocks. The price is then embedded in the body of the SOAP response message as shown in Listing 2.11. This very simple example shows that SOAP messages are mostly XML and gives a hint how XML data management in wireless sensor networks can help to support SOAP. We discuss this aspect in the following sections.

Listing 2.11: A SOAP Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

2.4.3 Related Work to Middleware and SOA for Wireless Sensor Networks

To simplify the complex engineering process of distributed sensor network applications, some concepts have been introduced in the past. However, the majority of these concepts are mainly targeted to simplify the data acquisition by providing a data-centric query language as described in Section 2.2 and can be categorized as query interfaces rather than programming interfaces [43, 156, 168, 253]. Other concepts like the module based engineering as proposed for NesC [74] help to organize the engineering process and / or provide abstraction layers from the actual used hardware, e.g. sensing and communication APIs or concurrency control. However, these concepts are also mainly targeted for experienced developers and the usage for non experts remains questionable. The terms of service oriented architecture or service centric applications have been used in [18, 50, 51, 104, 135, 162]. The goal of these works is the simplification of the engineering process by providing sensor node functions as services in the network. Blumenthal et al.[18] discuss an approach that is related to classic tight coupling of the participants of the distributed system using a distributed middleware architecture. The sensor node application thereby consists of the middleware, the operating system (node firmware) and device drivers. The operating system controls the devices. The actual device driver controls sensory tasks. The middleware organizes the interaction of the distributed sensor nodes in the network and manages the operation of local and globally shared (cooperative) services. The service oriented framework OASiS [135] supports the encapsulation of program functionalities represented as services on an abstraction layer. Services can be connected, forming a service graph that represents the data flow. If a sensor node fails, the corresponding services are deleted from the service graph if no response to a service request is received. In [51], Delicato et al. discuss a reflective middleware for wireless sensor nodes. This middleware provides global services for accessing and delivering the sensor data in the network. Like in the Cougar approach [253], there exist dedicated sensor nodes (like cluster heads) that are responsible for receiving, processing and answering service requests. These service directory nodes also include the descriptions of the existing services in the network. The approaches discussed in [104, 162] also discuss how service directories can be managed and used for finding and binding services. All of the presented approaches discuss interesting ideas. However, they are still theoretical concepts that have not been implemented for real sensor network hardware and deployments. Moreover, the concepts are often discussed on network models that differ from the real metrics of wireless sensor networks as discussed in Section 2.1.3. A recent development in sensor network middleware research is the integration of sensor networks in Cloud environments, e.g. [196]. While this is a promising approach to optimize the integration of sensor networks in other networks and to simplify the usage of sensor networks, these approaches are also still in their early phase and have not been implemented yet.

2.4.4 Adapting the Service Oriented Paradigm to Sensor Network Engineering

In the last section, we gave an overview on related work in the field of middleware for distributed wireless sensor networks. We pointed out, that though there exist several approaches on integrating the idea of service oriented architectures in wireless sensor network, XML usage is the key feature for using standardized Web Service techniques. Furthermore, the extension of the service oriented paradigm to sensor networks will also provide better connectivity to sensor networks and better integration of sensor network services in web applications, e.g. data mining in sensor networks. For these reasons in recent years, combining service-oriented architectures with wireless sensor networks has been an important research area. The resulting approaches can be classified into two categories. In the first category, the sensor network acts as a whole as service provider by querying, processing and delivering data from sensor nodes [50, 191, 212]. In the second category, single sensor nodes provide services which can be composed to more complex Web Services using gateways for communication only [18, 135, 162]. Nearly all of them make use of standard Web Service technologies SOAP and WSDL, based on XML, for exchanging messages between the internet and the sensor network gateways or interfaces. Within the sensor network other special message formats are used for communication [135, 162]. However, Delicato et al. [18] propose an XML based communication within the sensor network, but this approach has not been realized on sensor nodes, yet. However, XML support remains crucial for integrating standardized and transparent service integration in wireless sensor network application engineering.

Applying and Extending the Service Oriented Paradigm to Sensor Network Application Engineering (AESOP's TALE)

As presented in the introduction of this thesis in Section 1.1.2, the adaptation of the service oriented paradigm to sensor network engineering is the goal of the DFG project AESOP's TALE [149]. This thesis contributes significantly to the project's goals as described in the next paragraph.
The main idea of AESOP's TALE is to integrate the standard web technologies for Web Services in the sensor network application engineering cycle. The sensor network itself should provide Web Services to outside clients but also be a service oriented architecture itself, e.g. sensor network programs are composed of in-network services. This idea has been visualized in the introduction in Figure 1.3.
From the technical point of view the AESOP's TALE project defines a general architecture for a sensor node that supports standardized Web Services as shown in Figure 2.15.

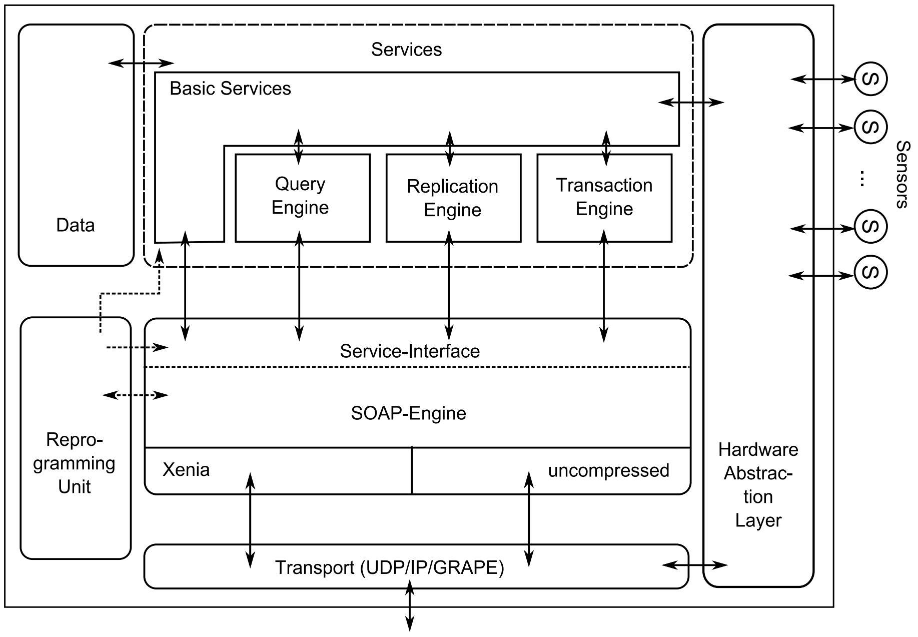


Figure 2.15: AESOP's TALE Sensor Node Architecture

The architecture summarizes the main research goals of the project as follows:

- An energy efficient transport layer, e.g. GRAPE
- A compression scheme for SOAP and XML transport and data management
- A SOAP engine

- A service interface for scheduling the service calls to the corresponding functionalities
- Service migration and dynamic programming of migrated service program code
- An XML query engine for processing queries on compressed XML data
- Service replication
- Transaction support for atomic and isolated service migration
- A basic service oriented operating system

We discuss the impact of this thesis on these goals in the next paragraph.

Impact on AESOP's TALE Project Goals

The work presented in this thesis provides the following contributions to the AESOP's TALE project goals [149]:

- Introducing $XOBE_{SensorNetwork}$ as a design framework to support developers using XML data (Chapter 3).
- Providing an efficient XML data representation on the sensor nodes (Chapter 4).
- Providing an XML query layer to access dynamic XML data within the sensor network using XPath (Chapter 5).
- Enabling the support of the Standardized Simple Object Access Protocol (Chapter 5).
- Providing various energy efficient optimizations for evaluating queries in the sensor network to enhance the network lifetime, e.g. XML data caching (Chapter 6) and Bounded Continous XML Queries (Chapter 5).

These contributions cover the following working packages of the project application (titles extended for understandability):

- Description of Resources within the Sensor Network
- XML Processing on Sensor Nodes
- XML In-network Storage and Data Management (Shared XML Data Repository)
- SOAP on Sensor Nodes
- Query Language and Query Evaluation for Data Resources

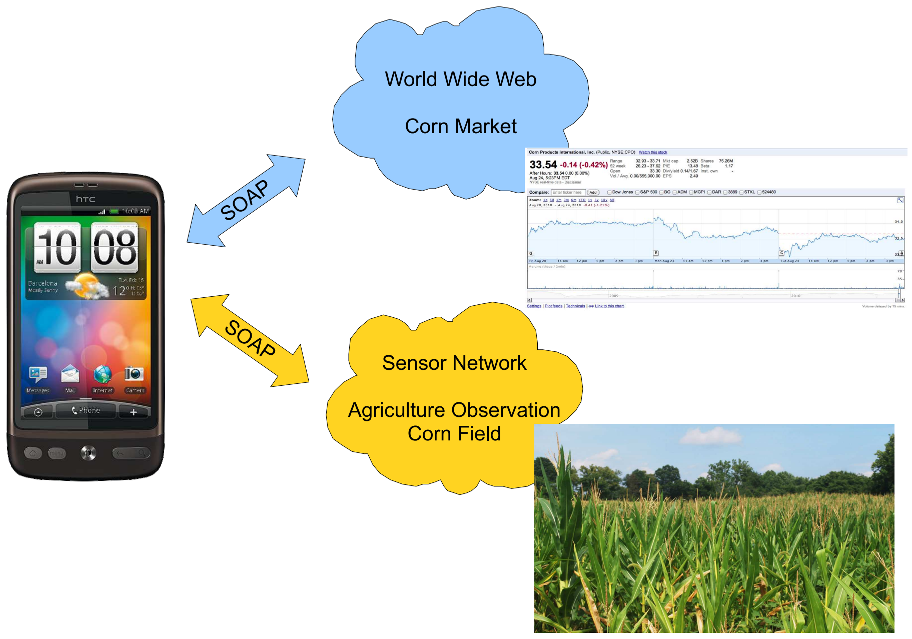


Figure 2.16: SOAP Usage in Wireless Sensor Networks Corn Farming Application Scenario

- Sensor Network Metrics (Estimating Resources)
- Demonstrator

Providing an efficient XML data representation is one of the key challenges of the AESOP's TALE project as it is the key feature for supporting the standardized Simple Object Access Protocol (SOAP) that is based on XML as shown in Section 2.4.1. One possible scenario is given in Figure 2.16. In this example, a mobile client is using an application that makes use of a heterogeneous Web Service, e.g. located in either the WWW or provided by the near sensor network. The communication should use standardized protocols for interoperability and compatibility reasons. Therefore, the mobile client is using the described Web Service technologies, e.g. SOAP messages, to bind service from the WWW and the sensor network. An application can be the mobile client of a farmer who is observing his corn field. The services from the sensor network provide information about the environmental conditions, e.g. humidity, solar radiation and nutritive value. Given this analyzation, the farmer can estimate the corn market price progress and use this information for trading, which could be also done using Web Services on the mobile client. On the other side, he can manipulate the field condition given a condition on the corn market that is presented on the mobile client, e.g. accelerate the corn growth using high speed fertilizer in case of an estimation of high corn prices in the next days. This pervasive usage of the sensor network and the WWW is the key feature of the AESOP's TALE technology and the concepts of this thesis.

Chapter 3

$XOBE_{SensorNetworks}$

In this chapter, we introduce the $XOBE_{SensorNetworks}$ programming framework to enable seamless XML data management support during the sensor network engineering phase and the sensor network runtime phase.

3.1 The $XOBE_{SensorNetworks}$ Programming Framework

As mentioned in the introduction of this thesis, handling data formats in the programming phase of wireless sensor network applications is a tedious and error prone task. This holds especially for handling complex data formats like XML as proposed in Section 2.3.4. Firstly, the structured form of XML requires more complex concepts for representing the structured XML data in the sensor node programming language, e.g. errors can be easily overseen. Secondly, the scarce hardware resources demand of energy and memory efficient representations that even aggravate the first challenge.
To support the developers of sensor network applications and services we introduce the $XOBE_{SensorNetworks}$ (XML Objects for Sensor Networks, $XOBE_{SN}$) programming framework for integrating XML in sensor node applications and the development process. $XOBE_{SN}$ provides a seamless integration of XML in the programming language, e.g. using XML variables in C and Embedded C [96]. It thereby hides the complex task of specialized XML data binding in wireless sensor node programs. This includes the choice of an energy and memory efficient data representation and the transformation task into this representation. Furthermore, it provides a runtime framework to allow accessing the represented XML data and further management tasks, e.g. XML data acquisition and XML data updates.

The idea of integrating XML in the programming phase to support developers is derived of the $XOBE$ and $XOBE_{DBPL}$ framework which have been introduced for the Java programming language in [150, 207]. These predecessor projects concentrated on the integration of XML objects in Java to bind XML data that is included in the Java source code. XML objects are plain Java objects that represent the XML data but desist from using any further compression. XPath was used to provide access to the XML objects. The most important feature of $XOBE$ was that each XML data operation was statically type checked against a given XML schema, e.g. DTD. $XOBE$ was extended by $XOBE_{DBPL}$ regarding the manipulation and management of XML objects. The focus of the initial $XOBE$ project was the development process with the ability of statically type checking the XML data and the active manipulation of XML objects during runtime. The follow-up project $XOBE_{DBPL}$ extended these concepts with a persistence layer that allows to deal with persistent objects, e.g. storing XML objects transparently in a database without any knowledge about the database connection. Furthermore, the data access possibilities were extended by allowing XQuery queries and XML updates with dynamical type checking.

The predecessor projects hence clearly motivated the concepts of this thesis and the $XOBE_{SN}$ programming framework. All three approaches focus on XML as a central de-facto standard data format for data exchange. The developer should be supported in handling XML during the engineering phase by enabling the transparent usage of XML constructs in the target programming language. The actual data binding is hidden to the developer and completely processed by the $XOBE$ framework autonomously.
Nevertheless, there are significant differences in the goals of the approaches. The main difference is that both predecessor projects were targeted for non-distributed systems without any resource limitations. The transformation of the embedded XML data in representations of the programming language resulted in a flexible object representation following the design guidelines of object oriented programming. In wireless sensor networks we have to deal with very scarce resources. Moreover, beside the iSense operating system, sensor network programming is not done in object oriented programming languages. Hence the transformation process is more complex including the need for efficient memory allocation routines and efficient garbage collectors. Object representations have generally a high memory demand. Hence, the representation of the XML data on the sensor nodes needs to be further compressed without loosing the ability of dynamically accessing the XML data, e.g. for evaluating queries.
For accessing the XML data in the sensor network, XPath can also be used. However, as wireless sensor networks are large scale distributed systems, new concepts for query dissemination and energy efficient result processing and delivery need to be introduced. This includes an extension to support continuous XPath queries. Since sensor networks are usually deployed in a target area without the ability to maintain applications in field, the process of type checking is also very important. Statically type checking the XML data in the programs supports finding errors that may have significant impact on deployed applications, e.g. making it necessary to redeploy the entire network because an error can not be corrected in field.
In summary, the target platform makes the integration of XML more difficult as for the predecessor projects. $XOBE_{SN}$ is mainly focussed on data acquisition and management.

Hence, it is more related to the first $XOBE$ project. While the efficiency and safety of the framework is essential for sensor network engineering, the creation of a persistence layer is not a main design issue. The support of update languages has also not been discussed as a requirement in the sensor network community. Nevertheless, it may become an extension field for future work.
In the following parts of this section, we discuss the integration process in the field of wireless sensor networks and the integration of XML data in the Embedded C programming language in detail. The XML integration process consists of different phases which are described in the subsequent sections.

3.1.1 Architecture

The entire $XOBE_{SN}$ framework consists of a *compile time* part and a *runtime* environment. The part for the *compile time* is responsible for handling any XML data that is used during sensor network application engineering. The tasks during compile time can be summarized as program parsing, type checking and transformation. $XOBE_{SN}$ acts as a precompiler and translates XML enriched C programs into code that is compilable with standard C compilers. Thereby, it supports the following key features that are introduced in the following sections:

- *Transparency:* Direct and transparent XML usage in the engineering process.
- *Stability:* Static XML typechecking to ensure stable applications.
- *Efficiency:* Memory and energy efficient XML transformation in the target language.

The *runtime* environment provides methods for handling and managing transformed XML data on running sensor nodes. This includes XML document access and the evaluation of queries on the XML data. Furthermore, evaluation results need to be themselves compressed to reduce the size of data that needs to be transmitted. Lastly, compressed data that is received by any sensor node or the gateway needs to be uncompressed, e.g. in order to process it outside the network. Hence, every sensor node running the $XOBE_{SN}$ framework also includes output routines.
We present the general architecture and the program flow of $XOBE_{SN}$ in Figure 3.1. $XOBE_{SN}$ takes any sensor node program code, e.g. in Embedded C, and a given XML schema, e.g. a DTD file, as an input. As introduced before, the source code may include the usage of XML data, e.g. plain XML assignments and XML variables. We discuss this syntactical extension in the next section. The source code is parsed to create an abstract syntax tree that is used for the following processing steps. Besides, the schema file is also parsed to store information about the type and structure of XML elements and attributes. The source code parser is responsible for checking if the XML is well-formed. If not, $XOBE_{SN}$ cancels the compiling task. The validity of the embedded XML data is checked in the next type

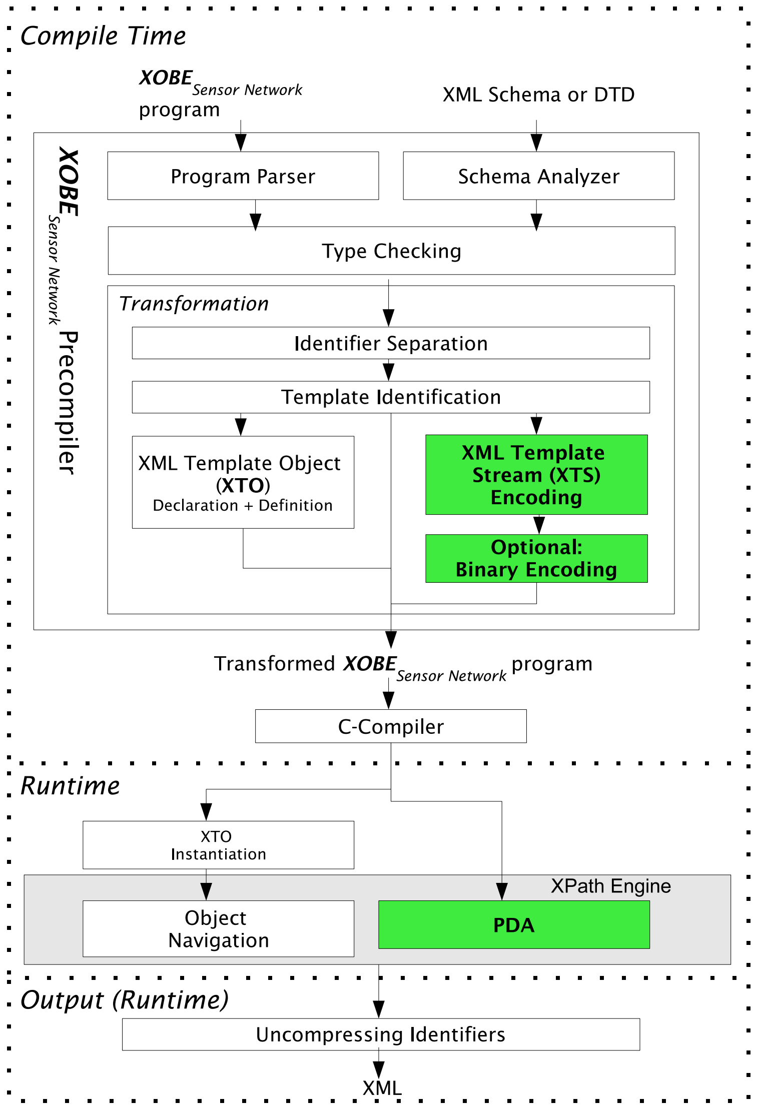

Compile Time
XOBE Sensor Network program
XML Schema or DTD
XOBE Sensor Network Precompiler
Program Parser
Schema Analyzer
Type Checking
Transformation
Identifier Separation
Template Identification
XML Template Object (XTO) Declaration + Definition
XML Template Stream (XTS) Encoding
Optional: Binary Encoding
Transformed XOBE Sensor Network program
C-Compiler
Runtime
XTO Instantiation
XPath Engine
Object Navigation
PDA
Output (Runtime)
Uncompressing Identifiers
XML

Figure 3.1: $XOBE_{SensorNetworks}$ Architecture

checking phase that we discuss shortly in Section 3.3. The last phase during compile time is the transformation which can also be denoted as data binding. As the source code needs to be compiled by the standard C compiler all XML fragments need to be transformed in compiler conform C code that needs to represent the XML in a compressed form to meet the hardware restrictions on the sensor nodes. We discuss the efficient transformation step in Section 3.4. The result code of $XOBE_{SN}$ is now forwarded to the standard C compiler chain. After compilation, the program can be loaded on the sensor nodes. The $XOBE_{SN}$ runtime environment then supports input and output routines of the XML code, e.g. serialization for data transmission. It further enables evaluating XML queries using different concepts as proposed in Chapter 5.

3.2 Transparency

As discussed before, the difficult programming phase of WSN applications may be even more tedious when dealing with different data exchange formats, even with XML. The native use of XML data in the program code can significantly simplify the programming phase, because the developer can use XML in its natural form without thinking about internal representations in the target programming language. This idea is the basic feature of $XOBE_{SN}$. $XOBE_{SN}$ ensures transparency by integrating direct XML usage in the program code, e.g. plain XML code can be used in declarations of dedicated XML variables and assignments. The task of $XOBE_{SN}$ is to parse the program code and additionally analyze a given schema that restricts the XML usage within the program as shown in Figure 3.1.
We give an example for $XOBE_{SN}$ code in Listing 3.1 that is also used throughout this thesis in various forms to explain the concepts of XML compression and query evaluation. Additional examples are given in the following sections in Listing 4.1, 4.13 and 4.14.

Listing 3.1: $XOBE_{SN}$ Example Code

```
1  ...
2  xml<nodereport> ReportOne;
3  xml<nodereport> Report;
4  xml<sensors> Db[2];
5  ...
6  int i=0;
7  for(i=0; i<2; i++) {
8    Db[i] = <sensors rev="1" id={i}>
9                  <id>{i}</id>
10                 <bat>3</bat>
11                 <sens>21</sens>
12             </sensors>;
13 }
14 ...
15 ReportOne = <nodereport>
16            {Db[0]}
17        </nodereport>;
18 ...
19 Report= <nodereport>
20            {xmlUnfold(Db,2)}
21        </nodereport>;
22 ...
```

The basic syntactical extension is the introduction of XML variables. XML variables are declared using the marker *xml* followed by the type and a variable name, e.g. *xml<nodereport> Report*. Additionally, it is possible to declare arrays / lists of XML variables as shown in Listing 3.1 Line 4. Arrays are declared using standard C syntax, e.g. defining the array length in brackets. XML variables are now used for XML as-

signments, e.g. assigning plain XML code to the XML variable: *xml<type> varname = <type>PCDATA</type>*. While this simple example shows how to assign static XML data, assignments may include references to other regular and XML variables as shown for *ReportOne*, *Report* and *Db* in Listing 3.1. Any references in assignments are thereby processed as call by value, e.g. a copy is placed in the data structure instead of a cross reference. Dynamic information like the loop index or actual sensor readings can be embedded in the XML code.
In the assignment of the array entries *Db* we show how to embed the id of a node in the XML fragment. Moreover, it is possible to embed other XML variables as shown for the XML variable *ReportOne* where the first entry of the Db array is embedded. The embedded XML variable can be the assignment variable itself, resulting into recursive assignments, which are useful for storing historical information in chronological order. For better handling of XML arrays, we introduce the *xmlUnfold* operator that takes any XML array and an integer as a parameter. The xmlUnfold function then flattens the embedded array by copying the given number of entries of the array in the XML data that is currently assigned. In our example assignment of *Report*, the final variable will include the first two entries of *Db* in the XML data as children of the element *nodereport*.
While we introduced the declaration of XML variables and assignments, following the predecessor projects it is also possible to use XML queries, e.g. stated in XPath, in the $XOBE_{SN}$ source code, e.g. *ReportOne = var(Db)/sensors[0]*. However, due to the practical unimportance for sensor network applications, e.g. making this feature rather optional syntactical sugar than essential syntax, this functionality was left out in the current release for optimizing the program code size. Nevertheless, it can easily be integrated, if future application fields make use of it.
Finally, the actual transformation in plain C code is hidden to the developer which simplifies engineering and furthermore avoids errors during programming. This concept is defined as transparent XML engineering or short transparency. The $XOBE_{SN}$ precompiler automatically transforms the assignments into correct C code using efficient transformation rules that are described in the subsequent sections.

3.3 Stability

The next phase is a static type checking that is based on hedge grammars [122] to verify static type correctness of the included XML code. This phase is of significant importance to ensure more stable programs and avoid costly maintenance of deployed networks. In general, if XML documents are managed it is often necessary that the documents are not only well-formed but also valid concerning a given XML schema file, e.g. given by a DTD or XML Schema document. To verify that a given program produces valid documents it is normally necessary to perform comprehensive tests at runtime. This procedure is denoted as dynamic type checking. To avoid runtime expensive validity tests in deployed

networks, $XOBE_{SN}$ instead offers the possibility of static type checking. A program analysis at compile time decides if the given program produces a valid XML document or not. As an example, $XOBE_{SN}$ can decide for the assignment of *Db* the right hand side of the assignment does not conform to the type of *sensors*. If not, the static type checking procedure reports an error.
According to Kempa et al. [122] and Schuhart et al. [207], the type checking algorithm consists of the following parts. First, the formalization phase translates a given XML schema file into a formal representation based on regular hedge expressions. In Figure 3.1, we introduce the schema parser that is responsible for building up this formal representation. The next phase is type inference, that is given by the XML variable declaration. For a more complex type inference, e.g. determing the type of query and update expressions, we refer to [207] as it is currently not addressed by $XOBE_{SN}$.
Using the described type checking algorithm, it is possible to check if assigned XML data is valid, if assigned XML data corresponds to the XML variable and if referenced XML data conflicts with the type of the XML fragment where it should be embedded. In this way even the return values of queries, e. g. formulated in XPath, can be guaranteed to be valid. So the effort needed for testing a program is significantly reduced and more stable programs are generated. This is especially important in the area of sensor networks because programming and testing of sensor nodes is an error-prone and tedious process in itself. If an error is not detected until all sensor nodes are deployed, in the worst case the whole sensor network can become useless and a redeployment of the whole network may not be profitable. Even if it is affordable to reprogram the whole sensor network, this often means reprogramming each sensor node via a physical link, which is a time consuming procedure.

3.4 Efficiency

We previously denoted that the $XOBE_{SN}$ precompiler is responsible for transforming the XML assignments in plain C code for the sensor nodes. In Section 2.1.3, we introduced the technical constraints in wireless sensor networks, e.g. low energy, processing and memory resources. In Section 2.3.5, we pointed out that these scarce resources create strict limitations for integrating XML usage in wireless sensor networks. These challenges may be the reason why native XML support has not been proposed for wireless sensor networks. The general verbosity of XML conflicts with the limited energy and memory capacities of sensor nodes. For this reason, native XML support has to be based on efficient XML data binding structures that save space, time and energy by eliminating the XML overhead. Furthermore, sensor networks are typically deployed for data acquisition. Dynamical XML processing, e.g. querying and updating, still has to be possible and XML data binding libraries have to be small sized, which limits the compression possibilities and makes XML data binding a difficult task in sensor networks. The term *efficiency* therefore describes the transformation of XML in a memory and energy efficient representation. Moreover, it describes the efficient management and processing of transformed XML during runtime. This includes an efficient transmission of XML queries and results.

$XOBE_{SN}$ Compression and Data Holding Concepts

$XOBE_{SN}$ therefore supports two separate compression and data holding concepts that are introduced in the following sections. Both strategies are defined as *template based compression schemes* as they first analyze the given XML data for repeating structures that will be described by defined templates as described in Section 4.1. However, they differ in the way the actual compressed data is further processed and stored in memory. We introduce *XML Template Objects (XTO)* as the first implementation of the concept in Section 4.2, whereby a light weight object model is used to preserve direct access to the XML data. A more memory optimized implementation approach are *XML Template Streams (XTSs)* that are discussed in Section 4.3. Hereby, the compressed encoding of XML documents is kept in memory in a plain encoding format without explicit object model. While this approach does not provide the direct access to the represented document, the absence of object structure in memory results in a more efficient memory usage. We will show that both approaches support dynamical evaluation of XML queries in the corresponding sections.

Transformation Requirements

In summary, an efficient XML data binding solution for sensor networks has to fulfil the following criteria that define the requirements for the $XOBE_{SN}$ transformation process:

- *Memory Efficiency:* Representing a high amount of XML data with a low amount of allocated memory.
- *Runtime Efficiency:* Using only a minimal number of processing cycles for processing XML data.
- *Processability:* Allowing to process XML data dynamically without an expensive decompressing step.
- *Communication Efficiency:* Serializing XML in a compressed way for reducing the transmission costs.

Following these requirements, we introduce the implemented transformation concepts of $XOBE_{SN}$ in the subsequent chapters. In detail, the next chapter includes strategies on storing and managing transformed XML on the sensor nodes at a high memory and runtime efficiency. In Chapter 5, we discuss the dynamical, runtime efficient evaluation of XPath queries which includes representing XPath results at a high memory efficiency.

Chapter 4

The XML Template Compression Scheme

In this chapter, we introduce a scheme for compressing XML data in wireless sensor networks based on XML templates. The general idea of this compression scheme is defined in the next section. The compression scheme however only provides a general concept of how XML data can be compressed. We therefore introduce two separate implementations of this concept in the subsequent sections. Both implementations are focussed on special requirements in the sensor network, e.g. either providing an efficient data access framework or reaching the highest compression ratio. Nevertheless, they both are fully applicable on today's sensor node platforms and support a high memory efficiency, runtime efficiency and processability as demanded in the last chapter. We finalize this chapter by comparing both implementations and evaluating them by using common benchmarks.

4.1 The XML Template Compression Scheme

In Section 2.3.5, we pointed out that there are limitations on representing high amounts of XML data on sensor nodes. Especially, if XML is used as a centralized data format in wireless sensor networks where the size of data may become very high for long term measurement applications. Traditional XML data binding techniques without any compression are not suitable for representing XML in sensor node programs as described in Section 2.3.6. To take care of these hardware limitations and system requirements, in this section we define a XML template compression scheme.

XML Templates

This template compression scheme is based on the idea of splitting up XML fragments into dynamic and static parts, which can be encoded and compressed individually. Like in the example in Listing 3.1, resulting XML documents often consist of repeating structures, e.g. the structure of the *sensors* element. These repeating structures can be defined as static parts, because they do not change over time. The other parts, like the *id* in Listing 3.1 are considered as dynamic parts and can be processed separately. The only consideration that needs to be taken care of is to link static and dynamic parts for processing the entire document, e.g. for output routines or query evaluation.

We engross this idea by giving another example for an $XOBE_{SN}$ program that produces rapidly growing XML documents by using recursive assignments. While the previous example in Listing 3.1 included a non recursive structure, we explicitly use a recursive structure in this example because it clearly visualizes the idea of XML template compression. Nevertheless, for both examples the results of the template compression scheme are the same. The example is given in Listing 4.1. It describes the sensor data of a Btnode that stores the current sensor data identified by an measurement *id* on the first level and embeds the historic data in a recursive way.

Listing 4.1: XML Template Compression Scheme Example Code

```
xml<btsysinfo> sensor;
sensor = <btsysinfo>
             <id>0</id>
             <bat>10</bat>
         </btsysinfo>;
for (int i=1; i <=n; i++) {

  int battery = getBat();

  sensor = <btsysinfo>
          <id>{i}</id>
          <bat>{battery}</bat>
          {sensor}
           </btsysinfo>;
}
```

For every iteration of the loop the program generates a new XML fragment with three elements. If there are no optimizations for transferring this program into Embedded C program code, there will be a high demand of memory and the memory of the processing sensor node will be exhausted soon. Representing the XML with strings and char arrays respectively without any further compression will result in a memory demand of 57 bytes for the XML structure and 2 bytes for the dynamic integer values. The string representation will therefore demand more than 1 kByte after 16 iterations. If only the dynamic integers are stored, 250 iterations would be possible before 1 kByte of data is exceeded. This reveals that XML data binding for sensor network programming is full of potential for compression.
The proposed XML template compression scheme now defines an XML fragment sepa-

ration. The static part consists of the tags <*btsysinfo*>, <*id*> and <*bat*>. The dynamic part is linked by placing insertion markers for the integer variables *i*, *battery* and the XML element variable *sensor*. The resulting static template is shown in Listing 4.2.

Listing 4.2: Resulting XML Template for Example Code Listing 4.1

```
<btsysinfo>
  <id>@0</id>
  <bat>@1</bat>
  @2
</btsysinfo>
```

During runtime, the program will produce a growing XML structure / document. How dynamic parts and static parts are linked together is up to the corresponding implementation of the scheme. Nevertheless, in every implementation the XML template should be stored only once. In Figure 4.1 we summarize the result of the XML template compression scheme during runtime after two iterations. The corresponding XML document is shown in Figure 4.2.

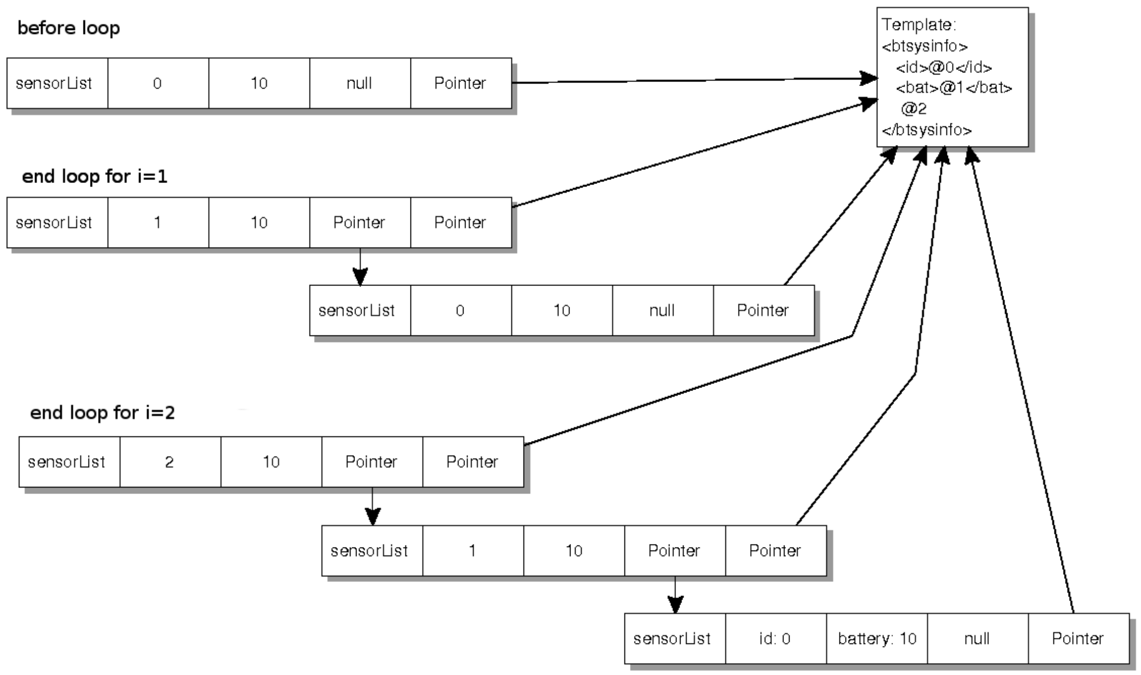
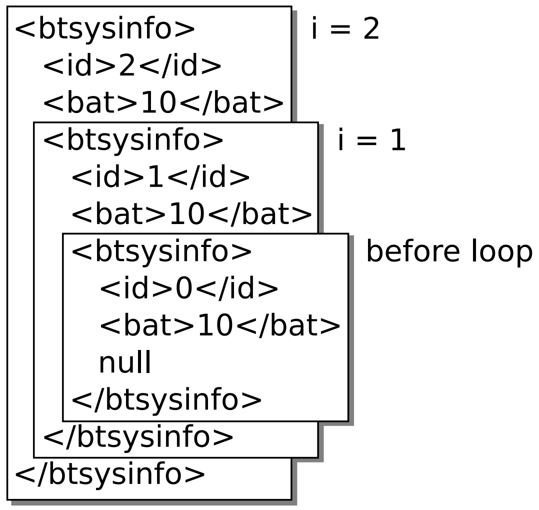


Figure 4.1: Result of the XML Template Scheme after 2 Iterations

As denoted previously, this recursive example clearly visualizes the benefits of XML template compression. The previous example of Listing 3.1 that uses an XML array instead of a recursive structure can be compressed in an identical way. An XML template can either describe recursive embedded XML fragments or a list of fragments. The result of the compression is the same as we will show in the following sections.



```
<btsysinfo>
  <id>2</id>
  <bat>10</bat>
  <btsysinfo>
    <id>1</id>
    <bat>10</bat>
    <btsysinfo>
      <id>0</id>
      <bat>10</bat>
      null
    </btsysinfo>
  </btsysinfo>
</btsysinfo>
```

Figure 4.2: The XML Document produced by Listing 4.1

Optimizing XML Templates using Separated and External Identifiers

XML templates are generated during compilation of XML based applications. This brings up the question if XML templates are needed during runtime. This answer depends on the application. If XML is only used to store data in a structured format and then to send it back to a gateway, the XML template can be stored outside the sensor network. Dynamic data is nevertheless strictly related to the external XML templates and can be used to generate XML documents at the gateway. However, if XML and XML queries are processed dynamically on the sensor nodes, storing only the dynamic parts is not enough. In this case, the XML structure needs to be accessed and therefore the XML templates need to be within the sensor network. A compromise is achieved by storing XML identifiers outside the network and the structure inside the network. This simple approach provides significant compression results, since XML identifiers are the most verbose part of XML in a considerable number of cases. However, the query engine needs to take care of rewriting XML queries before sending them into the sensor network. Listing 4.3 shows this simple but effective approach by an example.

The presented XML template compression was shown in theoretical manner. Nevertheless, there are open questions on how to implement this scheme that we will discuss in the following paragraph.

Implementation of the XML Template Compression Scheme

$XOBE_{SN}$ generally implements the XML template compression scheme transformation by performing three steps as shown in Figure 3.1:

1. In the first step identifiers are separated from the actual XML document. In detail,

Listing 4.3: XML Query Rewriting after Identifier Optimization

```
external array of identifiers:
name_array[1] = "btsysinfo";
name_array[2] = "id";
name_array[3] = "bat";

XML Template for <btsysinfo>:
<1>
  <2>1</2>
  <3>15</3>
  <1></1>
</1>

Rewritten XPath Query
/btsysinfo/bat => /1/3
```

an array of identifiers is generated and identifiers within the XML are replaced by references to the array.

2. The second step is a complex analyzation step in which XML assignments are collected and evaluated to find repeating structures. These parts are denoted as templates and can be reused for describing the static XML data as proposed previously. To adapt to dynamically changing XML structures due to the program flow, e.g. if-else constructs, a dynamic analyzer needs to be implemented that manages templates, adapts them and checks and updates existing cross references to the changed templates.

3. The last step is the actual transformation. The XML assignments need to be translated into code that is processable by the used standard C compilers. This step is often referred to as data binding.

As discussed in Section 2.3.6, most of the existing XML data binding solutions are not applicable. For example, a simple but transparent solution is to represent the dynamic and static XML data as plain strings in memory. Insertion markers in the template strings are used to link the template and the dynamic content. Generic C structs can be introduced for combining template and dynamic content information to save further memory. A result is shown in Listing 4.4. We denote the generic C struct an *XmlObjectString* that represents XML data, e.g. an XML fragment, in the source code.
In the variable *dE*, the dynamic part of the *XmlObjectString* is saved, whereby the static part (template) is stored in the char array variable *template*. The template is described in a plain string format. The insertion markers are standard C operators, e.g. %s. Any assignment will then be transformed to a constructor call as shown for the variable *sensor*. The constructor *newXmlObjectString* stores the dynamic variables in the array *dE*, e.g. 1

Listing 4.4: Generic String Representation of XML Templates

```
struct XmlObjectString {
  char * template;
  void ** dE;
};
...
// Example Assignment
char * template = "<btsysinfo>
            <id>{%s}</id>
            <bat>{%s}</bat>
        </btsysinfo>";

XmlObjectStringType sensor = newXmlObjectString(template,1,32);
```

and 32, and assigns the template. However, by accessing the dynamic part during runtime, a typecast from the original type of the dynamic variable and vice versa has to be done, e.g. integer, string or another *XmlObjectString* to string, in order to embed the dynamic part in the template at the insertion markers. The most memory efficient way to store the original type would be to store it in the template together with the insertion marker. On the other side, this would result into parsing the template string every time a typecast has to be done. Moreover, by using strings to store the template, parsing is needed for processing navigation steps of XML queries, e.g. XPath navigation. More parsing means more computing, which results into higher energy consumption that should be avoided in energy limited sensor networks. In conclusion, it is better to use alternative structures, e.g. tree object structures, for representing the static template instead of uncompressed string representations because:

- Subsequent parsing of the template for accessing dynamic elements should be avoided.
- Accessing dynamic elements has to become faster.
- Dynamic element types should be stored within the template.
- Evaluating XML queries should be simple and more energy efficient.

In summary, this simple string based solution still suffers of high memory usage because it desists from further textual compression techniques. Moreover, the overhead of parsing the entire string to access dynamic elements reduces the usability.

Other data binding frameworks like autoXML [123] generate structs and functions for every XML element according to a given schema. If the schema file is big, much memory is spent for this transformation method including the function library overhead. Therefore, they are not applicable for template encoding. Besides, they require a runtime environment that mostly exceeds the available program memory of the sensor nodes as introduced in

Section 2.3.6.

As a solution that is applicable for wireless sensor network applications, in the following sections we propose two separate data binding solutions that implement the XML template compression scheme. Both solutions are fully applicable for wireless sensor network engineering and have been evaluated on real deployed sensor nodes as presented in the evaluation in Section 4.4. We present the first solution *XML Template Objects (XTOs)* in the next section. In Section 4.3, we show how this solution can be further optimized for application scenarios with strict memory limitations by introducing the second solution: *XML Template Streams (XTSs)*.

4.2 XML Template Objects (XTOs)

As denoted in the previous section, the basic concept of the XML template compression scheme is to analyze given XML fragments in order to retrieve dynamic and static parts. While this process is just a logical information separation to optimize the compression of the XML data, the actual degree of compression depends on the implementation of this concept. An implementation needs to take care of the following tasks:

- encoding templates
- linking dynamic content and static templates
- processing the entire represented document, e.g. output of the document
- providing access to the entire represented document, e.g. accessing parts of the document

In this section, we discuss the XML Template Object (XTO) framework as the first implementation of the XML template compression scheme. This transformation is shown in the left path of Figure 3.1. Templates that are retrieved during the analyzation phase are represented as object structures. An XML variable, e.g. an XML fragment, is a structure that consists of dynamic content, e.g. the *battery (bat)* value in Listing 4.1, and a pointer to one or more representing templates. We denote this structure as an XML Template Object (XTO). Templates can be reused within the same XTO or be shared by various XTOs. Using an optimized C struct representation of those objects in memory and sharing repetitive information as templates results in a significant economy of memory as discussed in the last section.
While details on the implementation of XTOs are given in the next subsection, we give a practical introduction into the XTO transformation process and show how XTOs are processed in memory in the subsequent subsections.

4.2.1 XTO Implementation Data Model

In Section 4.1, we introduced the XML template compression scheme that includes a separation of the XML data in dynamic and static parts that need to be linked together. The XTO implementation maps this idea that is visualized by example in Figure 4.1 into an object model in Embedded C using C structs. In Listing 4.5, we show the resulting C structs that are the basis for further enhancements on XTOs.

Listing 4.5: Generic XML Template Tree Structure Representation in Embedded C

```c
typedef struct _xmlObject xmlObject;
typedef xmlObject *xmlObjectPtr;
struct _xmlObject {
  // static Part XML Template
  xmlTemplatePtr t;
  // dynamic Part
  void ** el;
  int noel; // #elements
};

typedef struct _xmlTemplate xO;
typedef xO *xmlTemplatePtr;
struct _xmlTemplate {
  int name;  // name marker
  int atname; // attribute marker
  struct _element** el; // elements
  char ** attributes; // attributes
  int noel; // number of elements
  int noatt; // number of attributes
};

typedef struct _element elem;
typedef elem *elemPtr;
struct _element {
  void * content;
  int name; // name marker
  char type; //type id
};
```

In detail, the object model consists of three C structs for generic *elements*, *templates* and an XTO representation (*xmlObject*). An XTO (*xmlObject*) encapsulates a template *t* and the dynamic content *void** el*. The *void* array only includes the addresses of allocated memory without any types of the stored data. Hence, it can be used to store dynamic data of various types simultaneously. This is important, because beside textual information other embedded XTOs may need to be stored in this array. The pointer *t* references the actual template that is described by the struct *_xmlTemplate*. In Table 4.1, we describe the meaning of the parts of the template struct. Elements and attributes that are stored directly in the template are static. Static elements of templates are represented by their value, name and type as shown for the *_element* struct. The integer *name* is a number reference to the array of identifiers that is created during identifier separation as proposed in Section 4.1. We will discuss the implementation of this identifier array in the next paragraph.

Element	**Description**
int name	Pointer to the identifier array (identifier of the root element)
int atname	Pointer to the identifier array (identifier of the first attribute of the root element)
struct _element** el	Array of child elements of the root element
char** attributes	Array of values for the root element attributes
int noel	Number of child elements (length of *el*)
int noatt	Number of attributes (length of *attributes*)

Table 4.1: Parts of the C Struct *_xmlTemplate*

The *type* of the element content is important because the actual values are stored type-less in a *void* array. Processing the *type* char is therefore mandatory for typecasting the content to its real type. The XTO implementation currently includes seven types as presented in Table 4.2.

Type Value	**Description**	**Resulting C Type**
1	Static textual content	char*
2	Static element node of a template	xmlObjectPtr
3	Empty element	void
4	Dynamic textual content	char*
5	Dynamic element node (embedded XTO)	xmlObjectPtr
6	Dynamic Array of element nodes (Array of embedded XTOs)	xmlObjectPtr
7	Dynamic element node without template (embedded light XTO)	xmlObjectPtr

Table 4.2: XTO Element Content Types

The integer type is not supported in the basic XTO implementation. Like the predecessor projects $XOBE$ and $XOBE_{DBPL}$, this implementation is focussed on DTD usage where content is defined by #PCDATA that is represented as a string. The type *7* represents an embedded XTO that has no own template and rather uses the template specification of the parent XTO. This concept is useful for optimizing the memory demand if templates are only shared by children of XTOs and not by two separate XTOs (no existing cross-references). Arrays of embedded XTOs (type 6) always share one template for the entire array in this XTO implementation to improve the compression ratio.

In the following parts of this thesis, we use a graphical representation of XTOs to improve the readability by avoiding long listings of source code. We therefore show the basic graphical representation of an XTO in Figure 4.3. The representation is hereby motivated by common graphical modelling languages, e.g. UML [20].

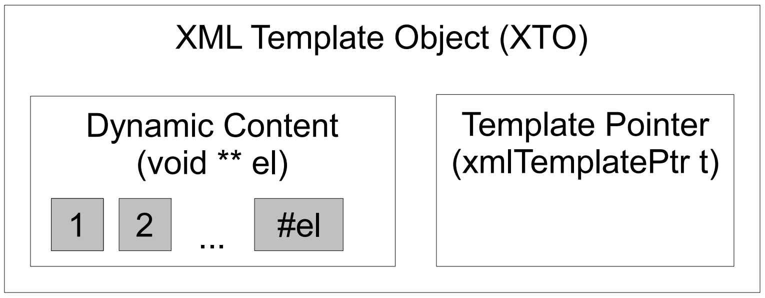


Figure 4.3: Graphical Representation of an XTO

DTD Analyzation and Generation of the Identifier Array

In Section 4.1, we discussed the separation of identifiers as the first compression step, which is shown in Figure 3.1. Beside the introduced XTO data model, an implementation of the XML template compression scheme needs to create a data structure that manages the identifier separation. $XOBE_{SN}$ therefore generates a separate routine that initializes an array of identifiers that can be referenced by the templates as proposed in the last paragraph. In detail, during the schema file analyzation that is shown in Figure 3.1, a methode *void xmlinit(void)* is generated that embeds the initialization of an array that includes all element and attribute names and the types of elements. The method is called during the boot process of the sensor nodes to provide this array during runtime. References are realized using the array position numbers as proposed in the last paragraph. As denoted previously, this array can also be stored externally, e.g. outside the sensor network at the gateway, to improve the memory demand. Nevertheless, this requires rewriting existing queries as shown in Listing 4.4.
In Listing 4.6, we show a possible DTD that corresponds to the previous example codes.

Listing 4.6: DTD File for BTstatus Examples

```
<!ELEMENT btnodes (btsysinfo)*>
<!ELEMENT btsysinfo  (id, bat, btsysinfo*)>
<!ELEMENT id       (#PCDATA)>
<!ELEMENT bat       (#PCDATA)>
```

The DTD is analyzed and the *xmlinit* method as shown in Listing 4.7 is generated. This method consists of the array *tG* for the identifiers and furthermore two variables *dynamicPart* and *XMLTemplate* that are used during the XTO instantiation process which we will describe in the next subsection. The even array entries represent the identifiers. The uneven array entries represent types that are further generated during the DTD analyzation phase. They are used to optimize the memory demand of XTOs and signal if an element that corresponds to an identifier contains textual information (type 1, e.g. #PCDATA) or may embed other child elements (type 2). The reason for this preliminary type system is, that less memory for textual element nodes can be reserved during initialization of XTOs.

In contrast, complex elements that may include child elements demand more memory, e.g. for additional pointers. Hence, during runtime this type information is used to differentiate between both types of element nodes.

Listing 4.7: Identifier Array Implementation

```
void xmlinit(void) {
  tG = realloc(tG, 8*4);
  tG[0]="btnodes";
  tG[1]="2";
  tG[2]="btsysinfo";
  tG[3]="2";
  tG[4]="id";
  tG[5]="1";
  tG[6]="bat";
  tG[7]="1";
  dynamicPart = malloc(2*sizeof(char));
  dynamicPart[0] = ',';
  dynamicPart[1] = '\0';
  XMLTemplate = malloc(2*sizeof(char));
  XMLTemplate[0] = ',';
  XMLTemplate[1] = '\0';
}
```

XTO Generation and Template Sharing

In this thesis, we consider using XML during programming as proposed in the last chapter, e.g. writing sensor network applications that make extensive use of XML documents, like saving sensor values in them. While the XML data definition during runtime is also considerable, we assume that the XML data is mostly defined at programming and compile time. In this case, working with XML can be seen as assigning XML fragments to variables.

The generation of XTOs always depends on where and what XML fragments are used in the program code. To avoid generating unnecessary XML templates $XOBE_{SN}$ searches for XML assignments during compile time to generate an XTO consisting of the static XML template and the dynamic parts of the assignments. This approach generates an XML template for every assignment and hence the instance of an XTO is stricly related to an XML template. In the first step, this approach has no advantage over representing XML as strings. A significant advantage results if XTO variables are used frequently within the program code. This is even more true for loop constructs and recursive functions, as they often result into rapidly growing data structures for the collection of data generated in each iteration. The more the XTO is used in the application, the more the memory efficiency becomes apparent.
However, generating XML templates for every assignment also misses further chances on

saving memory. In the case of a later assignment that differs from an assignment before in a minimal way, XML templates should be adaptable, e.g. extended if the assigned XML code is more complex and reduced in the opposite case. A first solution would be to create a generic XML template that represents the whole or a significant part of the given schema file. Obviously, even for DTDs it is not possible to create a generic XML template because of the freedom of expression of each schema. A schema consisting of many optional elements will result into a generic XML template with a high overhead for small XML documents consisting of only a few elements. Non used optional elements will therefore be represented by empty pointers, which actually also need to be stored and hence require memory space. In Listing 4.6, we show a DTD schema file that can represent the XML fragments in Listing 4.1. During compile time it cannot be determined how many *btsysinfo* elements will be inserted in each iteration. For this purpose a dynamic analyzer checks for changing XML structures during runtime and adapts XML templates if changes are detected, e.g. the size of the *btsysinfo* list is changed. To avoid destroying cross referenced XML templates, the analyzer logs the number of assignments to each XML template. Only if there is no other assignment the XML template is changed. Otherwise, a new XML template is created for the actual assignment and the old XML template is left untouched for the other references.

4.2.2 XTO Transformation Process

In the last paragraph of the previous subsection, we discussed the generation of templates in general. In this subsection, we describe how to instantiate XTOs during runtime and summarize the general XML to XTO transformation process.

According to the $XOBE_{SN}$ architecture in Figure 3.1 the transformation process works on XML assignments that have been already type checked in the previous phase. The plain XML data needs to be translated into the C code that is compilable with standard C compilers. According to the XTO data model that we introduced in the previous subsection, this transformation results into XTO instantiations based on the C structs presented in Listing 4.5. In the next paragraph we describe how to create / instantiate XTOs consisting of the XML templates and the dynamic parts that represent the assigned XML data. The last paragraph summarizes the entire transformation process including the management of XTOs during runtime.

XML Assignment to XTO Instantiation Transformation

For better understanding of the XML assignment to XTO instantiation transformation process, we refer again to the example code given in Listing 4.1. For simplification, we will leave the loop construct out. We assume that all generic template structures from Listing 4.5 have been generated.

For the transformation of the code that is shown in Listing 4.1 the XML variable declaration needs to be processed first. We therefore define a new XTO *sensor* as follows:

```
xmlObjectPtr sensor = newXmlObject();
```

The function *newXmlObject()* is a constructor that returns a pointer on an empty XTO that has been allocated in memory. The actual memory demand of an XTO is adapted dynamically when adding new elements. The remaining part of the assignment is the right part as shown in Listing 4.8. This part represents the actual XML fragment that is parsed by $XOBE_{SN}$ two times.

Listing 4.8: XML Assignment to XTO Instantiation Example

```
sensor = <btsysinfo>
          <id>{i}</id>
          <bat>{battery}</bat>
          {sensor}
           </btsysinfo>;
```

As a result, the parser generates two internal representations. The first is the XML template, which is the static structure of the assigned XML fragment, the second is the dynamic part of the XTO. For both parts, we use an internal representation to let the XTO be initialized by a central constructor during runtime to save unnecessary program code. The resulting code in an abbreviated form is shown in Listing 4.9.

Listing 4.9: Definition and Instantiation of an XML Template Object

```
sensor = (xmlObjectPtr)generateXTO(dynamicPart, sensor, 0);
sensor->t = (xmlTemplatePtr)generateXTO(XMLTemplate, sensor,1);
```

The parameter *0* determines that the dynamic part will be defined, while the static part is marked with a *1*. As mentioned before, this code only displays the first instantiation of the XML variable *sensor*. Further reassignments may result into adaptation or even regeneration of XML templates.

As denoted previously, the usage of the internal representation strings *dynamicPart* and *xmlTemplate* that are parsed by the function *generateXTO* is motivated by the significant reduction of program code. Rather than embedding the complete instantiation sequence of the XTO in the program code, the function *generateXTO* parses both strings and derives the XTO structure and instantiates the XTO accordingly. This concept has been co-designed with the encoding concepts of the XTS approach which we discuss in the next section.
In detail, if the XML fragment will be transformed directly into a sequence of C operations to instantiate an XTO and to add elements to this XTO, the result is a significant longer program code. We give a short example for the direct transformation into C operations in Listing 4.10. Hereby, we leave the construction of the template out and concentrate on the

Listing 4.10: Direct Transformation of XML into C Operations for XTO Instantiation

```
xmlObjectPtr sensor = newXmlObject();
...
// Add dynamic content
addEl(i, '1');
addEl(battery, '1');
addEl(sensor, '2');
...
```

Listing 4.11: Instantiation Loop for Adding Elements to an XTO

```
xmlObjectPtr sensor = newXmlObject();
...
for (int i = 0; i < elements ; i++) {
  addEl(content[i], '1');}
...
```

instantiation of the dynamic part. In this example, we use a method *addEl* that adds static and dynamic elements to the previously declared *sensor* variable. The parameters are the actual values of the elements *id* and *bat* and the embedded *sensor* value of the previous loop iteration. The second parameter of the *addEl* function determines the type as introduced previously, e.g. #PCDATA or another XML fragment.

If a high amount of elements needs to be added during runtime, the *addEl* function call has to be repeated over and over again. In this case it is desirable to put the actual information that needs to be added as an element in an array. This array is then processed in a loop and every entry is added as an element using only one program code line. An example is given in Listing 4.11.

Nevertheless, this example shows a significant drawback. The type information about the content that needs to be added gets lost. Hence, the element sensor will not be added correctly. Moreover, the number of elements that determines the length of the loop is variable during runtime. Hence, it has to be determined before this loop. In our $XOBE_{SN}$ XTO implementation we therefore introduce the usage of encoded representations for XTOs that need to be instantiated. As shown in Listing 4.9, we define two strings that separately encode the template and dynamic content of the XTO that is instantiated. The function *generateXTO* now implements a parser that parses both strings and processes them analogously to the example loop in Listing 4.11. Both encoding strings (*dynamicPart* and *xmlTemplate*) encode the sequence of C operations that is needed to generate the XTO that represents the current XML fragment. For every possible part of an XTO instantiation there is a corresponding code as shown in Table 4.3. In example, a new static element in a template can be encoded by using the *(e* marker followed by a *name* reference to the element name followed by the static *value*. We use the markers (oVar, (oVarArray and (o for describing special element types. *(oVar* signals that another XTO is embedded, whereby the reference to this XTO is stored in the array of dynamic elements as proposed previously. The *(oVarArray* represents an array of *(oVars* with shared template. The *(o* marker introduces an element that may include attributes and embeds other elements, e.g. children. We denote this kind of element as an complex element. The marker *attributes* is an integer that defines the number of attributes that should be added to the XTO that is currently generated.

Encoding	Description
(e , name, value	New static element value
(e_d , name	New dynamic element value
(oVar	New dynamic XTO
(oVarArray	New array of dynamic XTOs or elements
(o , name, attributes	New static element that is parent of other elements (complex static element)
(*	End of static XML fragment
(a , name, value	New static attribute value
(a_d , name	New dynamic attribute value

Table 4.3: Encoding the XTO Instantiation Operation Sequence

Finally, in Listing 4.12 we show the resulting code that instantiates the representing XTO for the XML fragment of Listing 4.8.

Listing 4.12: Resulting Code for the XTO Instantiation

```
void ** elemArrays;
...
elemArrays = realloc ( elemArrays, 1*4 )
elemArrays[0] = sensor // old sensor value

// define the dynamic part of the XTO
sprintf(dynamicPart, ",(e,%d,(e,%d,(oVar,",i,bat);
// define the template of the XTO
sprintf(XMLTemplate, "(o,0,0,(e_d,1,(e_d,2,(oVar,");

// instantiate the XTO during runtime
sensor = (xmlObjectPtr)generateXTO(dynamicPart, sensor, 0);
sensor->t = (xmlTemplatePtr)generateXTO(XMLTemplate, sensor,1);
```

The void array *elemArray* is generally used in the XTO implementation to store embedded dynamic XTOs (e.g. (oVar and (oVarArray). It corresponds to the *el* array in the XTO struct description. During instantiation the entries of the *elemArray* will be copied to the *el* array of the corresponding XTO. The dynamic content *dynamicPart* is defined as values of the elements *id* and *bat* and a reference to the *elemArray* where the embedded XTO *sensor* can be found. The *XMLTemplate* is defined as the complex root element *btsysinfo* followed by a dynamic element *id*, a dynamic element *bat* and a dynamic XTO *sensor*. During runtime the method *generateXTO* will now instantiate the XTO as proposed previously.

Summary of the XML Template Object Transformation Process

In the last two subsections, we discussed when and how to generate XTOs. Beyond the scope of initial instantiation of new XTOs, every reassignment of these types of variables during the program flow will result into an XML template update process. In this paragraph, we propose the complete transformation process for transforming XML to XTOs that is summarized in Figure 4.4.
Starting with the assigned XML fragment and a given XML schema file, the *Schema Analyzer* separates XML identifiers as described in Section 4.1. The next step is to generate the generic XML template structs, which have been introduced in the previous subsection. This includes the integration of further header files including the XML template object framework. For all XML assignments the transformator will then define XTOs and rewrite the assignment as shown in the previous paragraphs. The result of these compile time processes is a compilable Embedded C program.
During runtime, blocks using the XTOs will be reached and the constructor of them is initiated like described in the previous paragraph. This process is called *Assignment Interpreting*, because the constructor interprets the internal representations of XML templates and dynamic XML parts before instantiating the XTO. As also mentioned in the previous paragraph, the interpreter is used to avoid repeating function calls and hence to minimize the program code. At this moment, the system knows a variable of the proposed XML template type representing an XML fragment and document respectively. Reassignments are then observed continuously. If the new assigned XML can be represented by the XML template, only the dynamic parts are assigned. Otherwise we need to change the template. If there are cross references, it is not allowed to change the template, because this may end up in an unstable system status. In this case, generating a new XML template is the only alternative. Nevertheless existing XML templates can be referenced to store the new XML template in an optimized way. If there are no further cross references, we are able to adapt the XML template directly without any consequences for other assignments. With XTOs the system is always able to produce the represented XML in its native form. Furthermore, it integrates into heterogeneous networks like the WWW and is accessible by a native XML query engine.

4.2.3 XTO Transformation in Example

We engross the theoretical background on XTO transformation by giving practical transformation examples in this subsection. The first example is given for the $XOBE_{SN}$ example code in Listing 4.13.
First, we declare three XML variables *Report, Db, NodeInfo*. As a special case, *Db* is

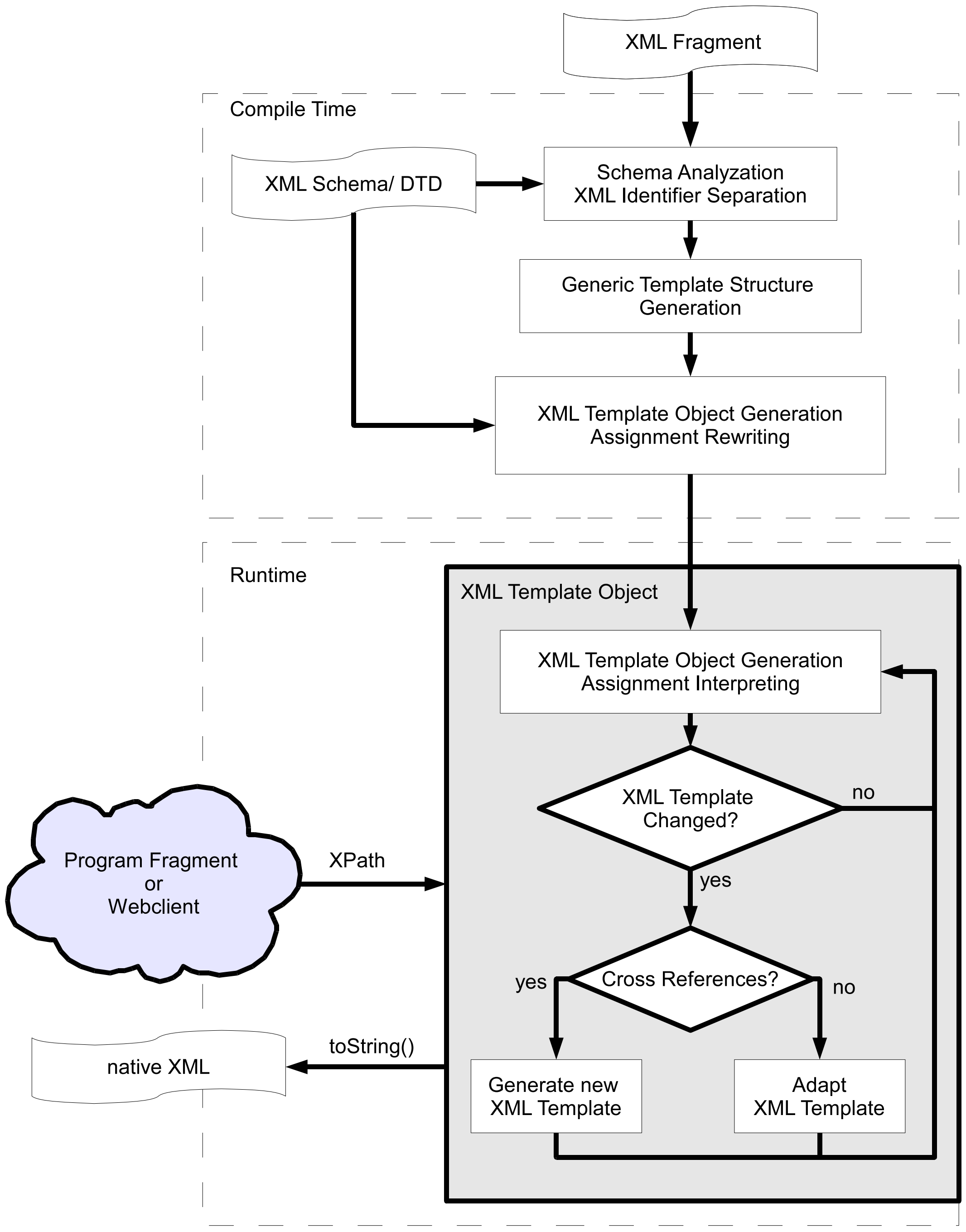


Figure 4.4: XML to XML Template Object (XTO) Transformation Process

an array of XML data of type *sensors*. The current status of the node is then assigned to *NodeInfo*. In detail, the XML variable *NodeInfo* consists of static attributes *id* and *rev* and a dynamic attribute *battery*, which is calculated by a function *remainingEnergy()* to insert the remaining energy of the node.

In a repeated task, new sensor data is then assigned to *Db* that acts as a ring buffer for sensor

Listing 4.13: XTO Transformation Example Code 1

```
...
xml<nodereport> Report;
xml<sensors> Db[] = new xml<sensors>[100];
xml<nodeinfo> NodeInfo;
NodeInfo = <nodeinfo id='1' rev='0.1'
            battery={remainingEnergy()}/>;
...
// Repeated Sensory Task
// t and l are the current sensor values
// i is the next insertion place in a ring buffer
Db[i] = <sensors>
          <temp>{t}</temp>
          <light>{l}</light>
        </sensors>;
...
// Task in case of query for full report
Report = <nodereport>
           {NodeInfo}
           {XMLUnfold(Db,100)}
         </nodereport>;
```

data. In case of a report query, the entire sensor data is summarized in the XML variable *Report*. For the *Db* array this means that it has to be flattened into the resulting XML report document. As introduced previously, the $XOBE_{SN}$ programming environment provides the integrated *XMLUnfold()* operator, that lets the developer determine how many entries should be inserted into the variable *Report*. Each *Db* entry thereby consists of dynamic elements *temp* and *light* to include current temperature and light values into the XML fragment *sensors*. As a special case *Report* includes complex XML data. First, the assigned variable *NodeInfo* and then the flattened complex array object *Db*. In summary, the example shows how we assign XML data to three separate XML variables, by directly writing plain XML on the right side of the assignment.
We show the corresponding XTOs that are generated by $XOBE_{SN}$ in Figure 4.5.
The variables *Db*, *NodeInfo* and *Report* are transformed into XTOs that consist of dynamic content and a pointer to a representing template. In detail, for the variable *Report* the dynamic content consists of two included complex XTOs *NodeInfo* and *Db*. The dynamic content of the XTO *Db* consists of a list of two doubles for the sensor values *t* and *l*, each pair representing an entry of the array. As denoted previously, a template of an XTO represents the static part of an XML fragment and describes how the dynamic content is related to this initially assigned XML fragment. By sharing common template structures like the template of *Db* for each entry that is also used by the recursively included old version of *XVar*, XML data is initially compressed. The templates and the dynamic content can then be further compressed, e.g. using the SAX event serialization, identifier

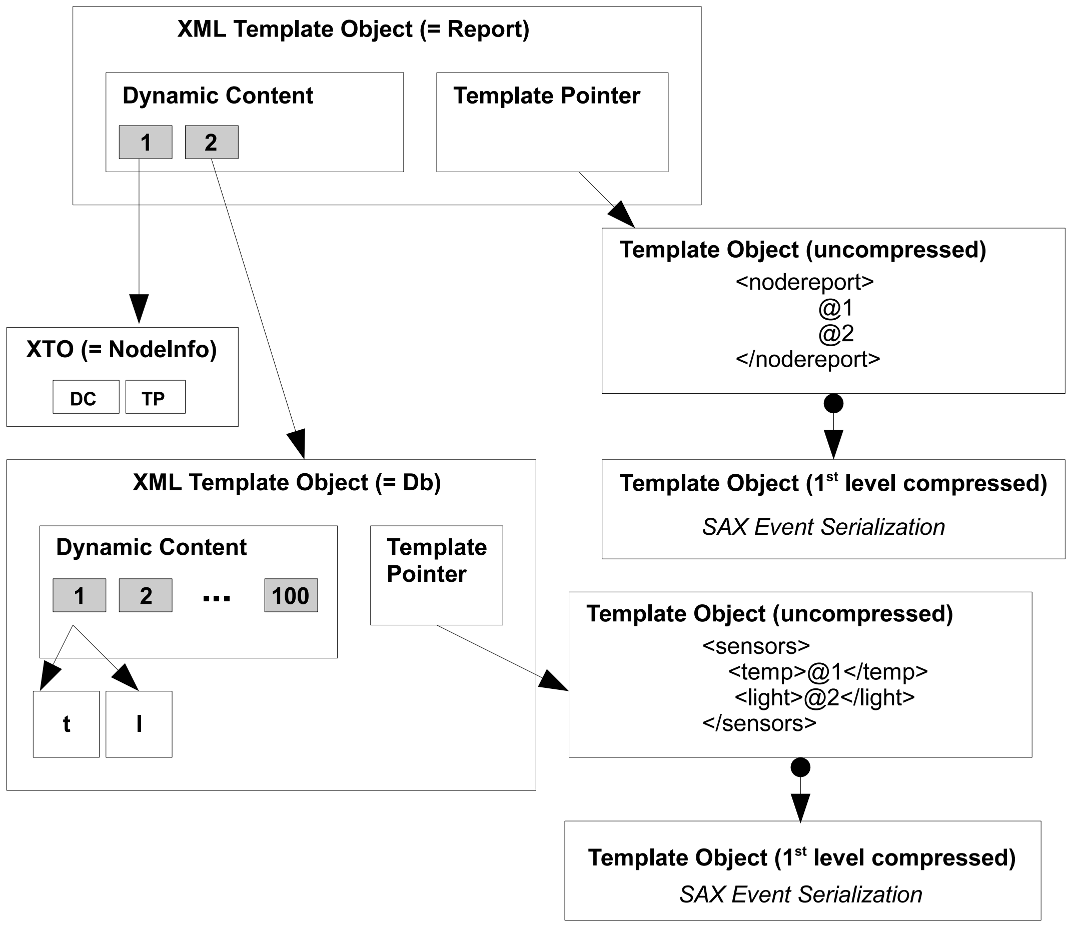


Figure 4.5: Resulting XML Template Object for Listing 4.13

separation, automata transformation and standardized binary compression. This approach is comparable to coding SAX events like presented in [39]. However, the key feature of XTOs is that each complex element that is represented is directly accessible. As we will show in Chapter 5, this feature can be used to store context node lists as a list of pointers, which makes the XPath evaluation efficient.

The second example in Listing 4.14 shows that $XOBE_{SN}$ is not only able to process direct assignments of simple variables, arrays or lists of XML data, but also supports recursive insertion of XML variables. In detail, the XML variable *XVar* consists of dynamic elements *temp* and *light* to include current temperature and light values into the XML fragment *sensors*. Again as a special case *XVar* also includes the variable *NodeInfo* as complex XML data. First, the assigned variable *NodeInfo* and then the variable *XVar* with its values before the assignment are included recursively. The corresponding XTOs is shown in Figure 4.6. The variables are transformed into XTOs and the dynamic content of the XTO *XVar* includes its values before the assignment recursively. The template remains shared among all instances of *XVar*. The two given examples are a simplified demonstration of the in-

Listing 4.14: XTO Transformation Example Code 2

```
...
NodeInfo = <nodeinfo id='1' rev='0.1'
           battery={remainingEnergy()}/>;
XVar = <data>
    {NodeInfo}
    <sensors>
      <temp>{t}</temp>
      <light>{l}</light>
    </sensors>
    <previous>{XVar}</previous>
     </data>;
...
```

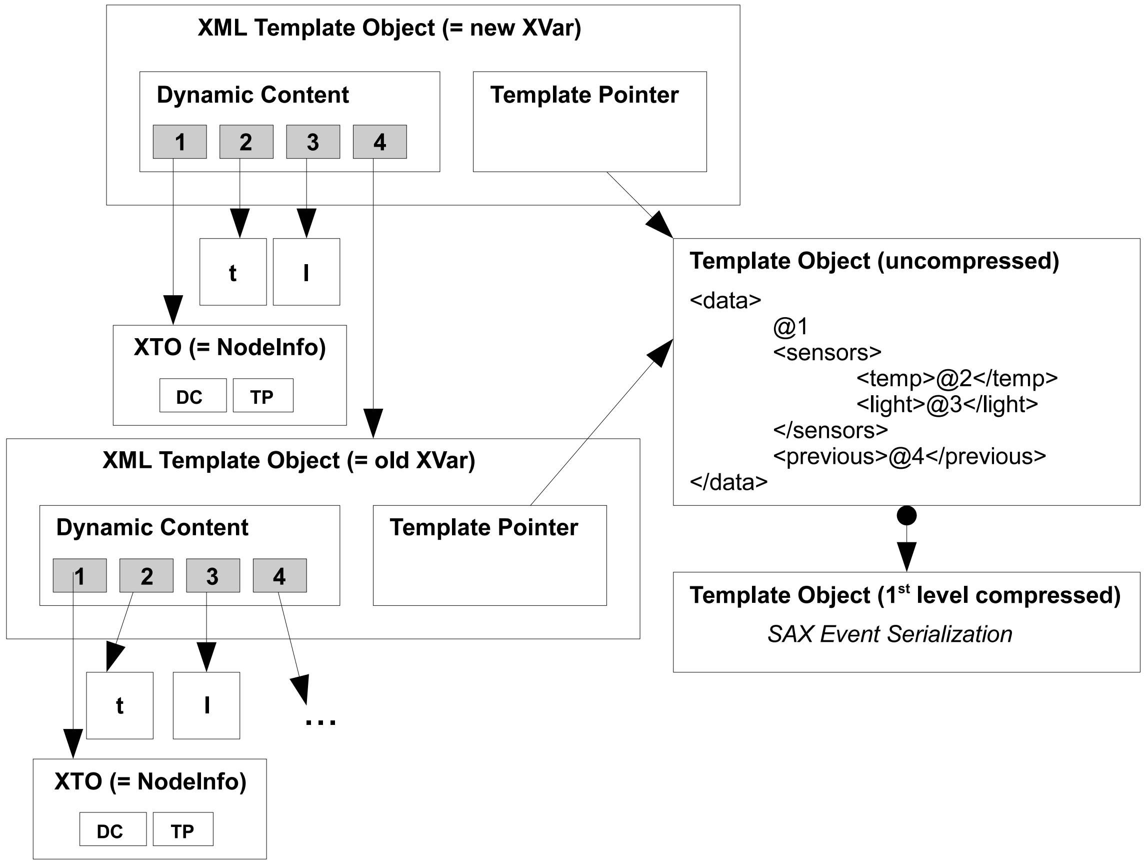


Figure 4.6: Resulting XML Template Object for Listing 4.14

ternals of $XOBE_{SN}$. Nevertheless, they show the benefits of the XTO concept which we summarize in the following subsection.

4.2.4 The Benefits of the XTO Concept

The benefits of the XTO concept can be summarized as follows:

- Significant XML compression by separating the dynamic content from the static XML structure.
- The structure of the original XML fragment is conserved.
- Providing direct access to the represented XML by building up a tree-like object model and allowing to place pointers in this object model.
- Efficient XML processing by traversing the XTOs object tree, e.g. using existing tree navigation algorithms.
- During traversing the XTO, regular SAX events can be thrown to be used for query evaluation.

However, the built-up object structure requires an overhead of memory for the initiated XTO objects and used pointers. We therefore discuss another implementation of the XML template compression scheme in the next section.

4.3 XML Template Streams (XTSs)

In the previous section, we presented the XTO approach as a first implementation of the XML template compression scheme. The XTO approach in general is an efficient solution for integrating XML data management and XPath query evaluation in wireless sensor networks. The biggest advantage of this solution is a high compression ratio while the direct access to XML content is retained.
In this section, we discuss an optimized implementation of the XML template compression scheme to store more complex, larger XML fragments in wireless sensor networks. The new approach hereby relies on processing binary streams instead of instantiating an object model during runtime. By omitting the possibility of directly accessing XML content, the compression ratio can be further optimized. Furthermore, in the next chapter we show that the evaluation of XPath queries on stream-based template objects is possible and applicable in wireless sensor networks.

Limitation of the XTO Concept

The previous XTO approach follows the strategy to build up objects that conserve a tree structure. For each complex object element, e.g. embedded XTO or attribute, an own tree-like instance is generated that is connected to the other elements by using pointer structures. This approach basically simplifies processing the entire XML document and searching for sub elements using existing tree navigation algorithms. However during the compression step of large and complex XML document representations, an overhead of instances and pointers may be generated for each element resulting in a non optimal memory usage on the sensor nodes. Moreover, instantiated XTOs need to be serialized first in order to transmit the represented XML data in the network.

Introducing the XTS Approach

In contrast to this previous approach, we introduce the XML Template Stream (XTS) approach that desists from using an object structure. The basic idea is to use binary encodings for the template and the dynamic part of the XML fragment. This idea is related to the idea of using encoding representations before object instantiation for reducing the program memory demand as proposed in the previous section. The template and content encoding are encapsulated in a data model that is denoted an XTS as described in the next subsection. Rather than instantiating an XTS like an XTO, during runtime the XTS is left in its encoded form. If the represented data needs to be processed, e.g. for evaluating XPath queries, the XTS is processed directly as a stream using a pushdown automaton (PDA) without instantiating sub parts of the XTS. This approach requires special concepts that will be introduced in the subsequent subsections.

4.3.1 XTS Implementation Data Model

In this section, we introduce the implementation of the XTS approach data model that implements the XML template compression scheme. As denoted previously, in this new solution we desist from using an object model and rely to process the encoded XML fragments and the included templates as a stream. As a result, no objects need to be instantiated during runtime, as shown for the XTO approach in the left path of Figure 3.1, resulting in a significant memory economy if the encoded documents are large and complex.
The XTS data model is represented by a struct encapsulating the dynamic part of an XML fragment and the template structure. We present the XTS struct in Listing 4.15. Unlike the previous XTO approach, this struct representation is only a container for both encoding strings. It does not represent any parts of a tree structure. The structure consists of two encoding strings. The string *t* represents the template. The string array *d* represents the dynamic content. The integer *noDyn* defines the number of elements in the array d. The dynamic content *d* is explicitly represented as an array because it is used for encapsulating separate XTS with the same template. Accordingly, each XTS will store its dynamic content in array *d* and use the shared template description *t*. The final encapsulation product is again an XTS. The array of *xmlTSObjectPtr* includes all embedded complex XTS with own template, e.g. similar to the (oVar and (oVarArray types in the XTO implementation.
After introducing the XTS data model, we discuss the XML fragment to XTS transforma-

Listing 4.15: XTS Data Model

```
typedef struct _xmlTSObject xmlTSObject;
typedef xmlTSObject *xmlTSObjectPtr;
struct _xmlTSObject {
  char *t;
  char **d
  int noDyn;
  xmlTSObjectPtr * elem;
};
```

tion process in the next subsections. This especially includes how the template and the dynamic content of XTSs are encoded.

4.3.2 XTS Transformation Process

The XML to XTS transformation process consists of the following four steps as also shown in the right path of Figure 3.1:

1. Identifier Separation
2. Template Identification
3. Encoding of XML data as an XML Template Stream (XTS)
4. An optional binary encoding of the XTS

As denoted in Section 4.2, the first two steps can be considered as common for every implementation of the XML template compression scheme. For the sake of completeness, we describe these two steps again:

1. **Identifier Separation**
   Like in the previous XTO approach, the first step of the transformation is to separate identifiers, e.g. element and attribute names, from the actual XML data. Identifiers are stored in program and flash memory or can even be stored outside the network. Storing identifiers outside the network requires translating queries to use references instead of the identifiers. Unlike the XTO implementation, the XTS implementation optimizes the implementation of the identifier array by separating element and attribute identifiers as shown in Listing 4.16. Attributes are always defined as values. Hence, a differentiation between different types like for elements, e.g. complex elements vs. #PCDATA elements, is not needed. By separating the identifiers of the attributes into an additional array the memory demand can be significantly optimized.

2. **Template Identification**
   The second step is also common to the previous approach as the XML data is analyzed to find shared repeating structures and to derive templates for representing these structures. Hereby, the static and dynamic content, e.g. *id* in the example Listing 3.1, are separated to allow dynamic processing of the XML contents. Together they form a unit that we denote XML Template Stream (XTS) as proposed before. In this XTS the static parts are replaced by references to the derived templates. Templates are shared and even reused within the same XTS to save memory on the nodes.

Listing 4.16: XML Fragment and resulting XTS Identifier Array Implementation

```
<btnodes>
  <btsysinfo att1="1" att2="2">
    <id>2</id>
    <bat>3</bat>
    <sens>21</sens>
  </btsysinfo>
</btnodes>


void xmlinit(void){
  tG = (char**) realloc(tG, 10*4);
  tG[0] = "btnodes";
  tG[1] = "2";
  tG[2] = "btsysinfo";
  tG[3] = "2";
  tG[4] = "id";
  tG[5] = "1";
  tG[6] = "bat";
  tG[7] = "1";
  tG[8] = "sens";
  tG[9] = "1";
  tAttG = (char**) realloc(tAttG, 3*4);
  tAttG[0] = "Elem:btsysinfo";
  tAttG[1] = "att1";
  tAttG[2] = "att2";
  }
```

The last two steps are the actual transformation of the XML fragment into an internal representation that can be processed dynamically on the sensor nodes. This internal representation needs to implement the template compression scheme to optimize the memory demand of the represented XML data. We discuss these parts of the XTS implementation in the next subsections.

4.3.3 Encoding of XML Data as an XML Template Stream (XTS)

To optimize the memory demand, XTSs need to be compressed. We denote this step as XTS encoding. Unlike the previous XTO approach, we desist from using an object model and encode XTSs in a way that they can be processed by a pushdown automaton as a stream during runtime.

Listing 4.17: $XOBE_{SN}$ XTS Example Code

```
...
xml<nodereport> Report;
xml<sensors> Db[2];
...
int i=0;
for(i=0; i<2; i++) {
  Db[i] = <sensors rev="1" id={i}>
                <id>{i}</id>
                <bat>3</bat>
                <sens>21</sens>
            </sensors>;
}
...
Report= <nodereport>
          {xmlUnfold(Db,2)}
      </nodereport>;
...
```

We define an XTS as a structure consisting of a template *t*, dynamic content *d* and an array of embedded XTSs *elem* as introduced in Section 4.3.1. To explain the encoding concepts, we will refer to the example $XOBE_{SN}$ code shown in Listing 4.17. For this example we retrieve two XTSs: *Report* and *Db*. Encoding an XTS is done by encoding the templates and the dynamic content. In the following, we present the stream encoding for templates. The dynamic content is encoded analogously.

To support memory efficient template encoding of the static XML data we distinguish between the structure of different node types, e.g. dynamic or static attributes, simple dynamic or static elements or complex elements, e.g. other embedded XTSs. We denote these distinct rules as encoding tuples. Each encoding tuple consists of encoding tokens and contains information about the node type through a marker, a reference to the array of identifiers and the value of the actual XML document. Based on the node type, additional meta information, e.g. the element type or references to identifiers, is attached to the tuple. Each encoding token is separated by the token delimiter ’,’.
In summary, templates are encoded using the grammar in Listing 4.18 and the code markers from Table 4.4.
The grammar includes the following terminal values:

Marker	Description	Element Node Type Encoding	Huffman Encoding
$(o$	complex element (XTS) incl. sub-elements/attributes		111
$(a$	static attribute		100
$(b$	dynamic attribute		1101
$(e$	static element	-1 atomic value -2 comment node -3 text node -4 processing instructions	001
$(d$	dynamic element		1100
$(v$	referenced complex element (embedded XTS)		0001
$(w$	array of referenced complex elements (array of XTS)		00001
$(*$	end of complex element (end of (o)		101
$(n$	empty element		00000
$,$	token delimiter		01

Table 4.4: Overview on Encoding Marker and their Function

Listing 4.18: XTS Encoding Grammar

```
T ::= "," id "," "0" "," TC
TC ::= (( O | E |  D | V | W )",")*
O ::= "(o" "," id "," aido "," (( A | B )",")*
       ( O | E | W | V ) "," "(*"
E ::= "(e" "," id "," value
D ::= "(d" "," id
A ::= "(a" "," value
B ::= "(b"
W ::= "(w" "," number
V ::=  "(v"
```

- *id* is a reference to the identifier array.
- *aido* is an attribute identifier offset used to reference the identifier of the first attribute. The following attributes can be found by incrementing this offset continuously.
- *value* is the actual value of the node in the XML fragment.
- *number* is an integer value determining the number of elements in an array of XTSs.

A template encoding starts with the root node (*T*). Hereby, a reference to the identifier (*id*) is defined. The following token determines the number of attributes, whereby standard is 0.
As denoted previously, the now following grammar rules depend on the actual node type:

- **Attributes and Elements**
  Rule *A* and *B* describe static and dynamic attributes. A dynamic attribute is marked by *(b*. The value of this attribute can be found in the dynamic content of the current XTS. For a static attribute marked by *(a* the value can be included in the template. Rule *E* and *D* describe element nodes without attributes and sub elements. Hereby, *E*

describes a static element that can be included in the template whereby the value of D is found in the dynamic content. Both rules include a reference *id* to the identifier of the element. We further introduce additional node types for static elements to support values (#PCDATA), comment nodes, processing instructions and text nodes as shown in Table 4.4. These element types are referenced using negative values instead of the reference *id*.

- **Complex Elements**
  Rule O introduces a complex element that can include other elements of various types and attributes. Complex elements are introduced by *(o* followed by the identifier reference *id* and an attribute identifier offset *aido*. Next, there is a list of static and dynamic attributes followed by a free selection of sub elements, sub complex elements and embedded XTSs or arrays of XTSs. Complex elements are closed to keep track of the recursion depth on the processing PDA stack.

- **Embedded XTSs and Arrays of XTSs**
  A significant benefit of the stream-oriented encoding is to embed external XTSs or even arrays of XTSs in XTSs. As shown in Figure 4.7 for the XTS *Report*, an XTS structure includes an array of XTSs *elem*. Entries of this array are referenced in the template stream by the rules V and W. Rule V describes an embedded XTS. If an embedded XTS occurs in a template stream marked by *(v* it is looked up in the *elem* array and processed separately under the usage of its own external template encoding. After processing the embedded XTS the enclosing XTS is continued to be processed. Rule W describes an embedded array of XTSs with *number* elements. Each entry of this array, will be processed stepwise like single embedded XTSs under the usage of one shared external template. Using embedded XTSs results in a significant memory usage optimization since templates can be reused. The compression optimizer in the analyzation phase is therefore trying to use as many embedded XTSs as possible.

XTS Transformation Example

We explain the encoding rules by example based on Listing 4.17. In this example two XTS *Db* and *Report* are encoded. The $XOBE_{SN}$ precompiler analyzes both XML fragments and generates two separate template encodings to describe the static structure. The template of *Db* is encoded as follows:

```
Db.t = ",2,0,(a,1,(b,(d,4,(e,6,3,(e,8,21"
```

This template will be shared among both entries of the *Db* array. Beside the identifier reference (2: *sensor*), it consists of a static attribute with value 1, a dynamic attribute that will be described in the dynamic content *Db.d*, a dynamic element (4: *id*) and two static elements with values 3 and 21 (6: *bat*; 8: *sens*). The usage of a complex embedded XTS is shown for *Report*:

```
Report.t = ",1,0,(w,2,"
```

Beside the identifier reference, it consists of an array of 2 embedded XTS (*(w,2,)* that are defined by the dynamic content *Report.t*. In this example, these XTS are both of type *Db*. Hereby the previously derived template encoding for *Db* is shared by both embedded XTS. The final transformation result is shown in Figure 4.7. We hereby show the resulting dynamic content *d* that is encoded during runtime analogously to a template encoding.

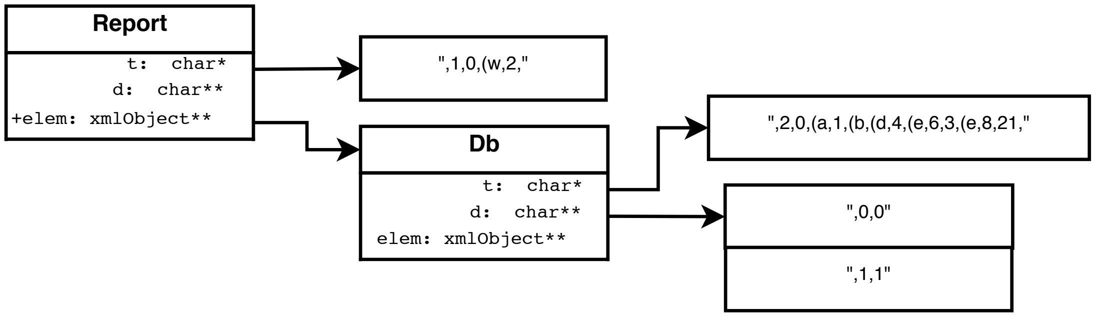


Figure 4.7: Structure of XML variable *Report*

4.3.4 Binary encoding of the XTS

In the last step of the transformation process we further optimize the template stream by using a binary encoding for the structure markers. In Table 4.4 we show the huffman encoding [107] we used for the evaluation of our approach. This encoding is representative and optimal for the general usage as no assumptions of the used XML code are made. However, the huffmann encoding can be adapted for special purpose scenarios where background information about the estimated XML document structure is present or runtime adaptation is possible [128].

4.3.5 Processing compressed XML documents using PDAs

Unlike the XTO implementation, the XTS implementation desists from using an instantiated object model. Hence, common tree algorithms can not be used for processing the represented XML data during runtime. For processing an XTS during runtime we use a pushdown automaton (PDA) with 5 input tapes and two output tapes. The input consists of the actual XTS that represents the XML data (one tape per template, dynamic content and *elem* array), an XPath query (see next section) and a binary stream which is used to select

dedicated sub elements in an XTS. If an XML fragment needs to be processed, the representing XTS is loaded to the input tapes. If the XML needs only to be read, this will be the only input. If a query needs to be evaluated additionally, the XPath query is loaded to the XPath Query input tape. The Bintype input tape is initialized as we describe later. During processing the XTS, several actions are possible. Beside realizing the context sensitivity of the grammar in Listing 4.18, the stack of the PDA is used whenever embedded XTS, e.g. introduced by the encoding marker (v, are processed.
These embedded XTS are loaded to the working tape and processed directly on it while the enclosing XTS is saved in its current processing state on the stack. The Bintype output tape is used for processing and encoding the results of XPath queries as we will discuss in the next chapter. The output tape is used for debug purposes to show the uncompressed XML fragment.

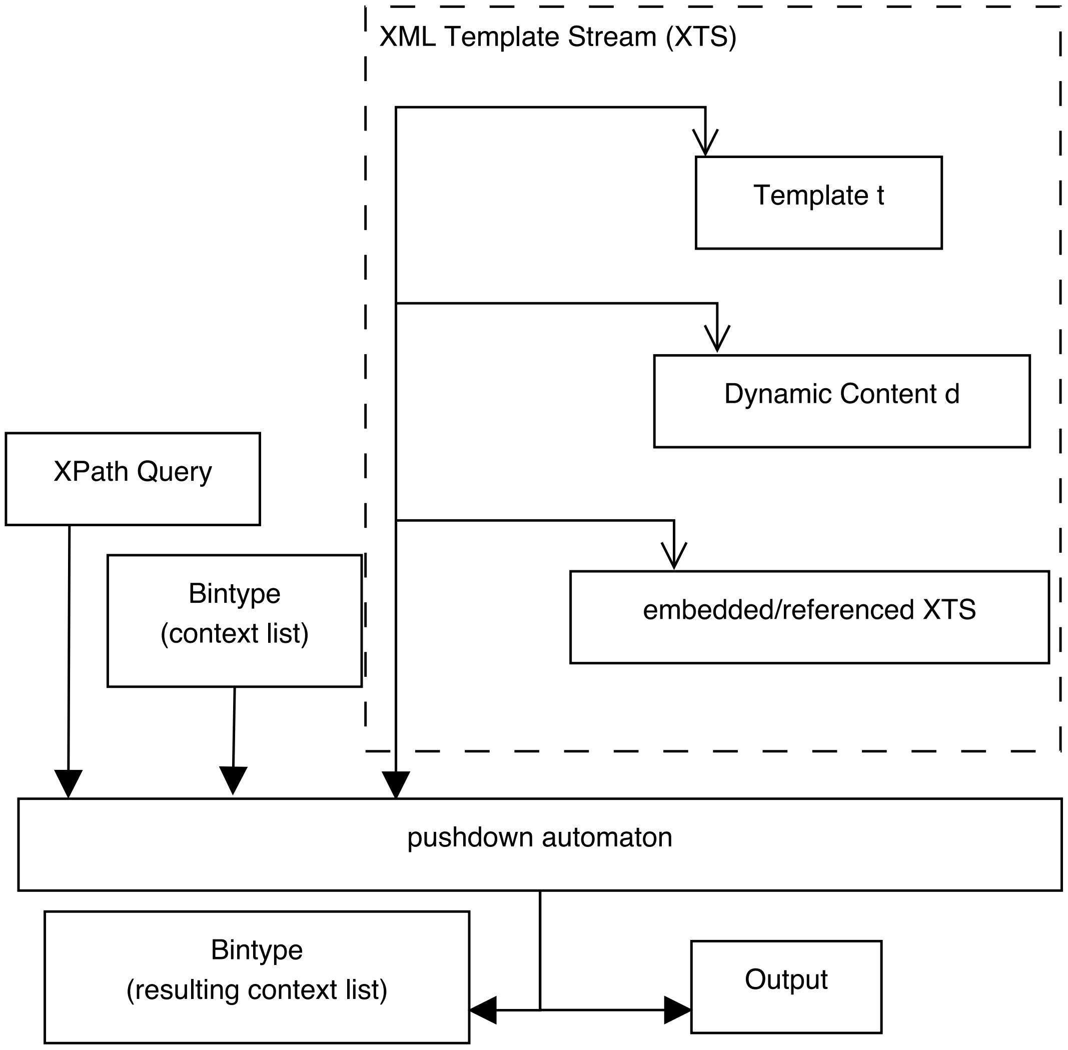


Figure 4.8: PDA for Processing Template Streams

Listing 4.19: XTS Bintype Implementation

```
typedef struct _binTypeObject binTypeObject;
typedef binTypeObject *binTypePtr;
struct _binTypeObject {
  int* bitsArray;
  int quant;
};
```

Binary Encoding of Node Lists

Selecting only sub parts of an entire XML document is important for output routines and especially for evaluating XPath queries. In detail, we therefore define a binary encoding of node lists, e.g. a context list in an XPath evaluation. The previous approach uses pointers within the object model to select elements. The XTS encoding desists from using pointers. Instead, we count the state transitions of the PDA when processing an XTS. A context node is uniquely determined by an automaton state transition id and the current stack content / height. We realize a memory efficient representation of the context lists by using a bit array (*Bintype*) that is shifted with each state transition. If the actual bit of the Bintype is one, the current state transition represents the start of the next context node. In the XTS implementation, we use integers instead of a binary type in Embedded C as they provide up to 32 bits. If the number of elements exceeds the 32 bit boundary, we concatenate another integer. The introduced C type *Bintype* represents this concept as shown in Listing 4.19.
The variable *quant* determines how many integers are in the *bitsArray* that represents an concatenated binary stream. The *bitsArray* hence is the actual implementation of the Bintype as shown in Figure 4.9. The $XOBE_{SN}$ framework includes an API for handling the Bintype type, e.g. functions for checking whether a bit is set or not or cloning functions that are used during the XPath evaluation process.

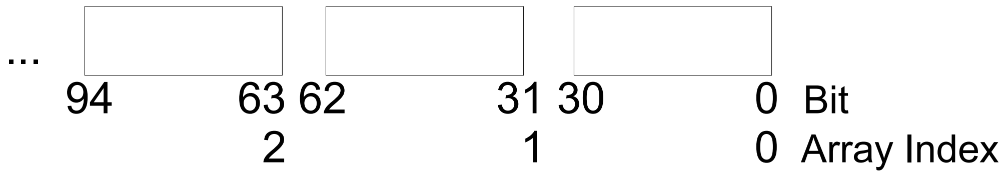


Figure 4.9: Structure of a Bintype

Finally, in Listing 4.20 we give an example for setting a Bintype to represent context nodes. In this example, we show an XML fragment whereby the elements *id* and *sens* should be selected by a Bintype. The XML fragment includes five elements and three atomic values. Hence, for selecting parts of the fragment at least 8 bits are needed if we assume a stepwise processing of the encoding XTS by the PDA. Additionally, we reserve the first bit (bit 0) for selecting all elements which results in a bit demand of 9 bits. If we assume that the

Listing 4.20: XTS Bintype Example

```
<btnodes>
  <btsysinfo att1="1" att2="2">
    <id>2</id>
    <bat>3</bat>
    <sens>21</sens>
  </btsysinfo>
</btnodes>
```

elements *id* and *sens* should be selected, we can define that the third bit and the seventh bit should be set to one. Accordingly, the integer of the *bitsArray* will be set to $2^3 + 2^7 = 136$. Since only one integer is needed, *quant* will be set to 1.
This example simplifies the Bintype concept by assuming that every new node in an XML fragment is represented by the next bit of the Bintype. In the actual XTS deployment this is only partly correct, as the next bit of the Bintype always represents the next state of the processing PDA. Accordingly, which bit needs to be set depends on the processing steps of the PDA which we will discuss in the next paragraph.

PDA Processing Action

The XTS PDA is constructed using the grammar rules of Listing 4.18. Hence, the PDA accepts the language defined by template encoding grammar. The stack of the PDA is used to save temporary contexts and keep track of the recursion depth of the represented XML document. A recursion happens if the XTS template encoding includes an embedded XTS or an embedded array of XTSs as defined by the markers (v and (w. The PDA processes a different action for every encoding marker in the template stream. In detail, the PDA acts as follows:

- **Attributes and Elements**
  Static attributes and elements are processed according to the encoding grammar under the unexceptional usage of the corresponding template. Dynamic attributes and elements require processing the dynamic content of the XTS. Therefore the PDA switches to the corresponding tape after reading the template information and switches back to the template tape after reading the dynamic content encoding. Both actions require keeping track of the head positions on both tapes.

- **Complex Elements**
  Processing complex elements requires storing the current context as a returning point on the PDA stack. This action is mainly used for query evaluation as described in the next subsection. When leaving the complex element the context is popped from the stack.

- **Embedded XTSs and Arrays of XTSs**
  When an embedded XTS or array of XTSs occurs in the template stream, the current state is stored in an extra stack buffer and the XTS is loaded to the three additional working tapes including the external referenced template. The embedded XTS is now processed as described before. For recursive embedded XTSs the extra stack buffer is used to store additional states. The maximum depth of recursively embedded XTSs can be limited by a definable maximum context stack size for optimization purposes.

We summarize the program flow of the XTS PDA in Figure 4.10.
The markers *t*, *d* and *elem* define if the *template tape*, *dynamic content tape* or *elem tape* is read. According to the XTS grammar, the PDA starts reading the template tape for getting the identifier and the attribute identifier offset of the XML data root element. In the trivial case of an XML fragment consisting just of the root element, e.g. <*sensor*/>, the PDA will finish afterwards. In general, we expect larger and more complex XML fragments and hence XTS representations. While the XTS template stream still has markers, the PDA will continuously analyze the next marker and perform the actions depending on the marker type as described before. We hereby point out the occurrence of the complex element markers *(o*, *(v* and *(w*. In case of a complex element marker (o, the PDA additionally pushes the (* marker on the stack to keep track of correct element nesting and to keep information about the hierarchical structure of the read stream. This information is needed during XPath query evaluation as we will describe in the next chapter. The embedded XTSs, e.g. introduced by the markers (v and (w, require pushing the current state of the enclosing XTS on the stack. This action is represented by the recursive call of the PDA as shown in Figure 4.10. Hereby, the embedded XTS is used as a parameter and processed analogously. After processing the state of the enclosing XTS will be popped from the stack and the processing action will be continued.

XTS Processing Example

For better understanding of the XTS PDA processing action, we give a short example. When the PDA processes the variable *Report* from Listing 4.17 it loads the template *Report.t* and dynamic content *Report.d* to the working tapes. First the identifier reference is resolved (1: *nodereport*) and the information that no attributes occur is processed. In the following, the token *(v,2* is read signalizing an embedded array of XTSs of the length 2. The PDA reads the *Report.d* tape in order to find the corresponding XTSs, e.g. the array of two *Db* XTSs.
The current context of *Report* is stored on the stack and the XTS *Db* is loaded to the working tapes. In the following, the PDA will load the dynamic contents of each embedded XTS (e.g. *DB[i].d*) to the working tape and process these streams separately following the previously denoted actions. After finishing the array the old *Report* context is popped from the stack and the PDA finishes processing the entire XTS.

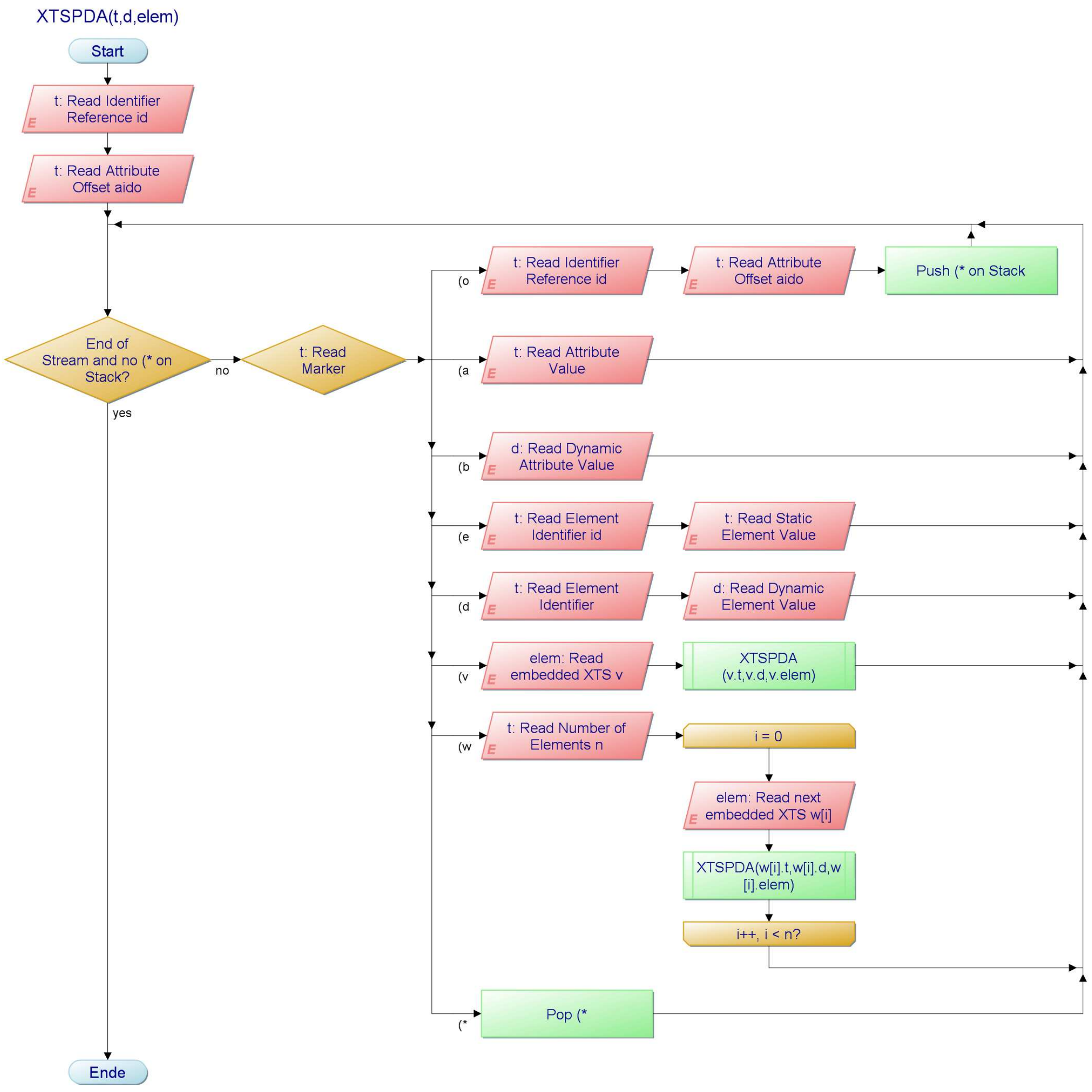

XTSPDA(t,d,elem)
Start
t: Read Identifier Reference id
t: Read Attribute Offset aido
End of Stream and no (* on Stack?
no
yes
t: Read Marker
(o
t: Read Identifier Reference id
t: Read Attribute Offset aido
Push (* on Stack
(a
t: Read Attribute Value
(b
d: Read Dynamic Attribute Value
(e
t: Read Element Identifier id
t: Read Static Element Value
(d
t: Read Element Identifier
d: Read Dynamic Element Value
(v
elem: Read embedded XTS v
XTSPDA (v.t,v.d,v.elem)
(w
t: Read Number of Elements n
i = 0
elem: Read next embedded XTS w[i]
XTSPDA(w[i].t,w[i].d,w[i].elem)
i++, i < n?
(*
Pop (*
Ende

Figure 4.10: XTS PDA Flowchart

4.3.6 The Benefits of the XTS Concept

The benefits of the XTS concept can be summarized as follows:

- Significant XML compression by separating the dynamic content from the static XML structure.
- Further optimizing the compression ratio by avoiding costly pointer structures and object models during runtime.
- Providing navigational access to the represented XML data using a PDA.
- Preserving SAX events that can be recognized by the PDA in order to evaluate queries.
- Introducing the Bintype concept for efficient XML node selection.
- Implicit serialization for efficient XML data transmission.

While we proof the memory efficiency of XTS in the evaluation in the next section, we discuss the drawback when processing the represented XML data in contrast to XTO in the next chapter.

4.4 Evaluation of XML Data Binding Techniques in Wireless Sensor Networks

In this section, we present the evaluation results for the presented XTO and XTS implementations. In detail, this evaluation covers tests and comparisons of the data management efficiency of both solutions. Efficiency is evaluated regarding the memory demand, the processability and the energy demand for data processing.

4.4.1 Evaluation Test Setup

All of the following tests and evaluations have been made on real sensor node products. We hereby use Pacemate nodes [153], based on a Philips LPC 2136 Processor, and iSense core modules, based on a Jennic 32bit RISC Controller [45] as shown in Figure 4.11. The available RAM of the iSense OS was 96kByte shared for program and data (heap memory was $\approx$15kByte, program memory was $\approx$81kByte). The evaluation test setup therefore conforms to the sensor node metrics that we introduced in Section 2.1.3. The results are transferable to other sensor node platforms that can be programmed in standard C. For example, the XTO implementation tests have also been repeated on the BTnode sensor node platform [28] which is related to the Mica platform [93]. In the following subsections, we present the main evaluation results.

Figure 4.11: Evaluation Sensor Nodes: Pacemate [153] and iSense Core Module [45]

4.4.2 Evaluation Criteria

As proposed in Section 2.1.3, sensor network programs have to satisfy the criteria of limited hardware resources to be applicable for long time running sensor network deployments. Hence, in this evaluation we test different XML representation techniques including the XTO and XTS implementations for the criteria that have been introduced in Section 3.4:

- *Data Management Memory Efficiency:* We evaluate the memory efficiency of the presented solutions for managing large scale XML data in memory. Hereby, an XML representation is memory efficient if for a fixed amount of allocated heap memory a high amount of native XML is represented. Less allocated memory also means a reduction of energy consumption for memory operations and data transmission.

- *Runtime Efficiency:* The energy consumption in wireless sensor networks not only depends on transmission but also on processing cycles. Therefore, a runtime efficient representation is defined by using a minimal number of processing cycles for processing a certain XML usage scenario. A minimal number of processing cycles results into less power consumption and thus extends the lifetime of the whole sensor network. Moreover, the runtime efficiency is crucial for time-critical application scenarios. In the next chapter, we concentrate mainly on the runtime efficiency when evaluating XPath queries in the network.

- *Processability:* We point out that both solutions (XTO and XTS) have been developed to support highest processability during runtime. The represented XML data is fully accessable and a plain output can be directly generated. XML data will change over time, e.g. sensor data will be updated, and whole XML fragments have to be accessible for the query engine. Thus, it is important that the representation techniques compress XML in a sufficient way but always allow to process XML data dynamically without an expensive decompressing step.

4.4.3 Data Management Memory Efficiency

In this section, we evaluate the XML Template Object approach (XTO) and the XML Template Stream approach (XTS) for the data management memory efficiency criteria. The evaluation results cover XML data representations for typical sensor network application scenarios, e.g. environment monitoring, and the wide spread XML Benchmark *XMark*. As denoted in the previous subsection, for an evaluation of the energy efficiency we refer to the XPath evaluation in the next chapter.

The results cover a comparison of the XTO approach, the XTS approach and other applicable XML data binding solutions. In detail, we compare different XML data binding techniques for their capability of representing XML on the sensor nodes:

- *XML string representation:* Representing XML in a native form as a string has been discussed in Section 4.1. The processability of this approach is limited due to the need of parsing the whole XML representation for every access. There is no compression of XML, which makes the native size of XML a lower bound for the memory efficiency of this approach.

- *libXML2 DOM API:* A DOM API can be used for representing XML. This is an approach with a high processability because single parts of the XML can be accessed using the DOM tree. We chose the libXML2 DOM implementation [227] for evaluation.

- *autoXML:* As presented in Section 2.3.6, autoXML [123] is a state-of-the-art C data binding framework. However, no compression is supported so that it is to expect that this approach is not memory efficient.

- *XML Template Objects (XTO):* The previously announced XML template compressing scheme whereby an object model is initiated during runtime that enables a high processability using tree navigation algorithms.

- *XML Template Stream (XTS):* The stream based XML template compression approach as described in Section 4.3 whereby the processability is limited to the capacities of the processing XTS PDA.

As discussed before, the evaluation was done directly on the sensor nodes. We measured the heap memory demand. This makes our results representive for today's most used sensor node platforms. However, the *libXML2 DOM* and the *autoXML* framework demand so much program memory themselves that they are not applicable for today's sensor nodes hardware. We therefore evaluated these techniques by using the AVR simulator [208].

Monitoring Benchmark

The first test case was a typical monitoring scenario. Each sensor node is running an application where current sensor data is embedded in an XML fragment. Historical data is stored continuously within the same document. The related source code examples are shown in Listing 3.1 and 4.1. We hereby point out that the actual implementation requires scheduled tasks using component based programming as usual for sensor node programs. Hence, the loop in the example listings is just there for simplification and is replaced in the actual sensor node program by iteratively scheduled sensory tasks.
The XML fragment grows with every iteration. The results are shown in Figure 4.12. Hereby, the x-axis denotes the actual native size of xml data that needs to be represented. The y-axis denotes the actual memory consumption on the sensor nodes for managing that data.

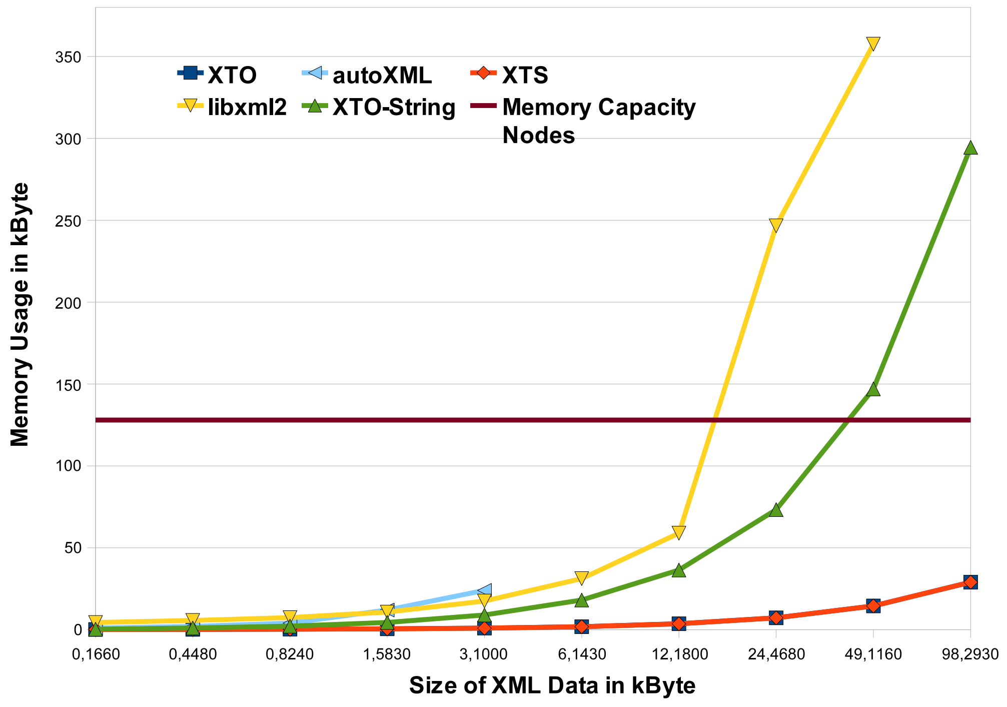


Figure 4.12: Monitoring Application Memory Usage

As a result, the optimized XTS approach and XTO approach perform similar. This minor optimization was supposed due to the less complex XML data structure. However both solutions outperform the competitors by having a significant lower memory usage. They are the most memory efficient way to represent XML within this application. By using XTOs or XTSs we reach a compression factor of 33% of the native XML documents.
Representing XML by using strings consumes twice as much memory as the native XML document. The reason is, that by reassigning the XML variable to itself the application

needs a temporary variable. This is always necessary when XML variables are embedded in others and shows up another problem of representing XML by strings in C.
Using libXML2 DOM in the application leads to a high memory demand. This is not unusual since libXML2 DOM is not a typical data binding application and is more related to simplify accessing XML.
Using autoXML for this sensor node application during tests caused a stack overflow early during runtime. However, we managed to measure autoXML's results until the native XML size reached 6 kByte. The increasing size of the autoXML representation was the most memory consuming one in our tests. The memory demand was steeply increasing, making this data binding framework not applicable for sensor network programming. Besides, autoXML proved to be not stable in real runtime deployments.

XMark Benchmark

To verify our results we use a sensor network application based on the XMark Benchmark data generator [206]. This data generator produces capacious XML documents and is controlable by a scaling factor *sf*. Because even the size of the native XML documents generated with a scaling factor greater than 0.006 excesses the memory restrictions of the BTnode platform, we tested this application with scaling factors less than 0.006. The results are shown in Listing 4.13.

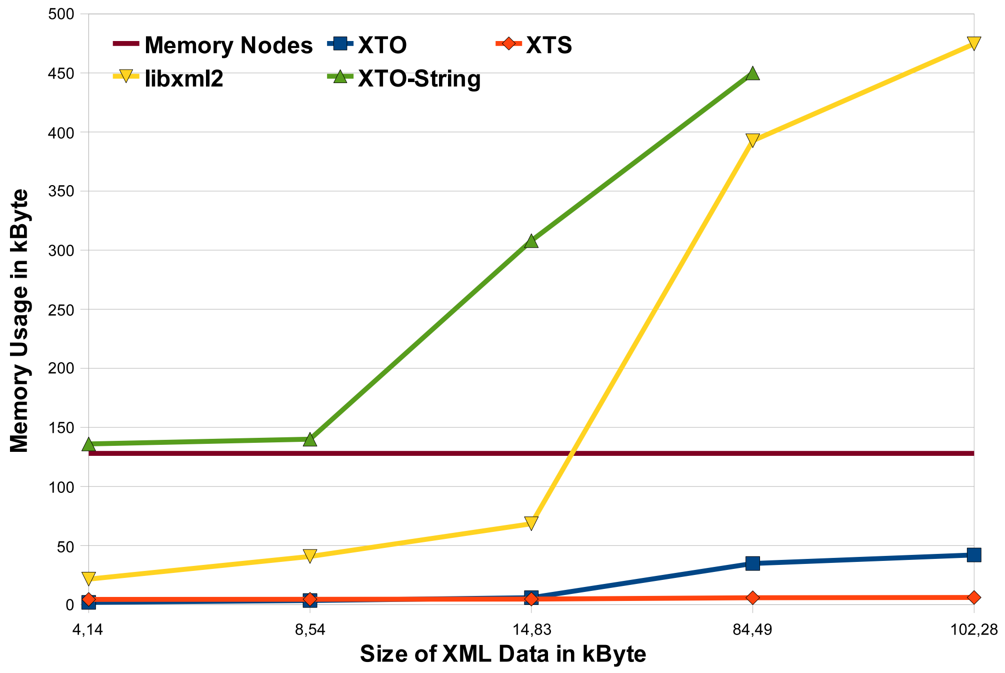


Figure 4.13: XMark XML Memory Usage

As a result, the XTS and XTO approaches also outperform the competitors. The interesting

questions in this evaluation was how the XTS approach performs in relation to the XTO approach. The XTS approach is mainly targeted for large scale, complex XML data that occurs in in-network storage or service oriented achitectures. For very complex XML data we expected an overhead memory demand caused by the XTO pointer structures. As a result, the optimized XTS approach significantly outperforms even the previous XTO approach by an absolute compression ratio of 35%.
In summary for the data memory efficiency, the XTS approach is fully applicable for large scale complex documents. It reduces the memory usage within the sensor network significantly. Even for simple applications it performs better or at least equal than the previous announced XTO approach. The positive benefits like direct query evaluation, result optimization and template caching remain. However, we point out that processing documents is slower due to the need of the pushdown automaton evaluation. We will show in the next chapter that the higher memory efficiency can also be achieved for evaluating XPath queries on the data representation. This again comes at cost of runtime efficiency.

Chapter 5

XML Query Evaluation in Wireless Sensor Networks

In the introduction of this thesis, we pointed out that wireless sensor networks are mainly deployed for data acquisition. Accordingly, data management solutions for integrating XML in wireless sensor networks need to provide query interfaces to retrieve dynamic sensor node data during runtime. In this chapter, we therefore introduce concepts for evaluating XPath queries on XML fragments that have been compressed using the concepts of the previous chapter.

5.1 XML Query Processing on XTOs

Processing XML queries natively on sensor nodes is limited by the general hardware restrictions, e.g. memory and energy resources. To the best of our knowledge there is no existing XPath query engine for existing sensor nodes or comparable devices.
The main reasons are the limited memory capabilities. First, the engine itself has to be stored in program memory. However, the program size of existing engines exceeds the memory of today's sensor nodes. Omitting parts of the query engine like the parser is no solution for scenarios, where the nodes run autonomously. This includes to be able to process the entire query from parsing over optimization to evaluation. The second drawback of existing query engines is the dynamic memory demand. The query evaluation itself has to be memory optimized by avoiding the creation of unnecessary temporary results and by creating memory optimized representations of necessary temporary results.

In the following parts of this chapter, we will discuss our design decisions for memory and energy optimized XPath evaluation on sensor nodes. In detail, three main aspects have to be considered:

- How to access XML data for evaluation ?

Listing 5.1: Example XML Document

```
// Template 3
<nodereport>
 // Template 1
 <nodeinfo id='1' rev='0.1' battery='0.89'/>
 // Template 2
 <sensors>
   <temp>21</temp>
   <light>10</light>
 </sensors>
 // Template 3
 <sensors>
   <temp>20.2</temp>
   <light>0</light>
 </sensors>
</nodereport>
```

- How to store XPath results?
- How to optimize the XPath evaluation?

This discussion is based on using XTOs as an implementation of the XML template compression scheme as introduce in the previous chapter. In Section 4.3 we presented an optimized implementation denoted as XML Template Streams (XTSs). This optimization requires additional logic for evaluating XPath queries. We therefore discuss special design decisions for this implementation in Section 5.2 that are nevertheless based on the following strategies.

The discussions in this chapter will not cover general issues on implementing an XPath evaluator. Algorithms for efficiently evaluating XPath queries in general have been introduced in [73, 80, 183]. Theoretical upper bounds of XPath query evaluation have been discussed by Gottlob et al. [79]. In this chapter we explicitly investigate practical solutions and design issues to enable native, energy efficient XPath evaluation in WSNs.

To give examples for the design decisions for evaluating XPath queries on XTOs, we will refer to the XML document in Listing 5.1 that has been assigned to the variable *Report* according to Listing 4.13.
We review the representation of this XML fragment as an XTO. According to Section 4.2 and Figure 4.5, *Report* is a variable of type XTO and the entire XML document is composed of the complex XTOs *NodeInfo* and *Db*. Each complex XTO consists of its dynamic content, e.g. temperature and light values, and a self-descriptive template. We marked the corresponding templates in the comments in Listing 5.1. As described in Chapter 4,

Listing 5.2: Template 2 for *sensor*

```
startElement("sensors",[])    //<sensors>
startElement("temp",[])       //  <temp>
@1                            //    @1
endElement("temp")            //  </temp>
startElement("light",[])      //  <light>
@2                            //    @2
endElement("light")           //  </light>
endElement("sensors")         //</sensors>
```

$XOBE_{SN}$ automatically conserves the SAX event structure of templates of XTOs. In Listing 5.2 we show the resulting representation of the Template 2 as SAX events. The dynamic content is referenced with @1 and @2. The final XML template object representation of *Report* is shown in Figure 5.1.

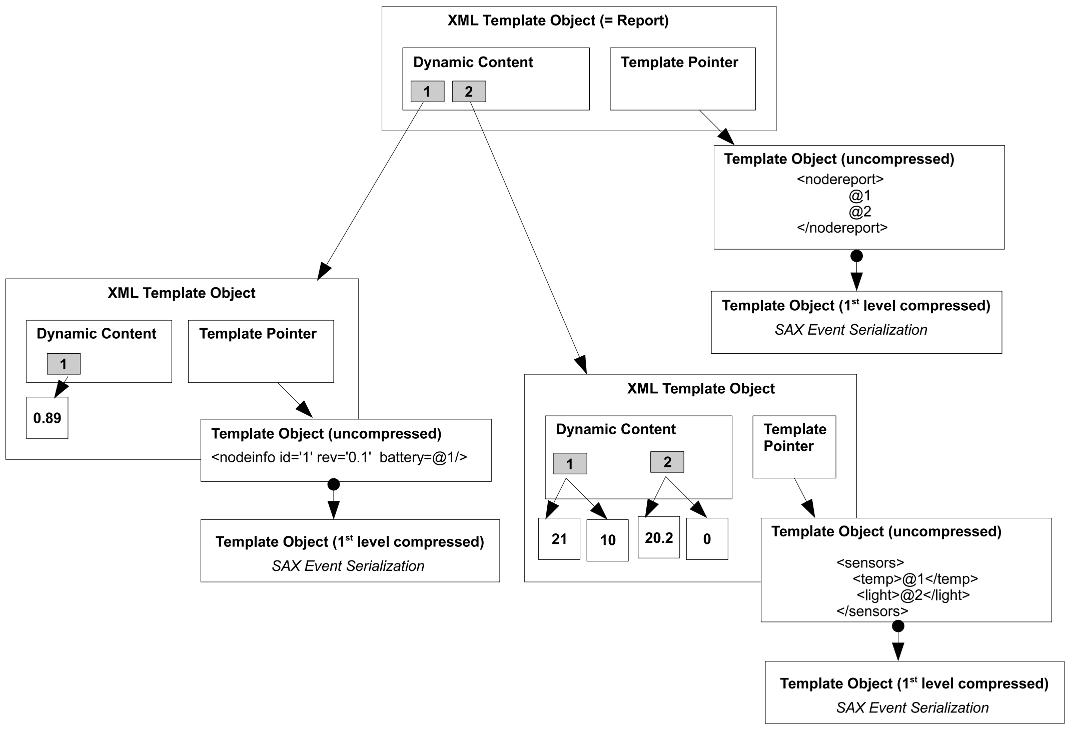


Figure 5.1: XML Template Object for Example Listing 5.1

5.1.1 Data Access

Evaluating XPath queries requires not only parsing the query but also parsing and accessing all parts of the XML document on which the query should be processed. While an extended object model like DOM provides random access to the entire XML document and therefore

does not require an additional parser, simple string and binary representations of XML data require parsing the document for every step in the query evaluation process.
The evaluation presented in the previous chapter showed that a simple DOM implementation is too complex and memory inefficient to be applied in WSNs. Nevertheless, the XTO compression approach creates a pre-parsed compressed structure, consisting of serialized SAX events that represent the XML document.
Processing the XML by using the template structures of XTOs implicitly invokes events for every element, sub-element, attribute and included complex XML object and hence acts like an inherent event based parser, e.g. SAX, without the need of a separate parser in program memory. As a result, the axes evaluator acts like a mealy machine with the SAX events as input, working on the corresponding XTO. We give an example for the child axis evaluation in Figure 5.2. The mealy machine consists of three states. The transition descriptions are described by (Input, Output). Hereby, '-' denotes *No Output* ,'Hit' denotes *Child Found* and '*' denotes *Any Other Input*. A transition is followed iff the corresponding SAX event has been read in the template description. Starting at the root node, the machine will accept all elements that start directly under the hierarchy of the root element until the *endElement* event of the root node is read. The evaluation of the other axes is done analogously following general algorithms for efficiently evaluating XPath axes as discussed in [73, 80, 183].

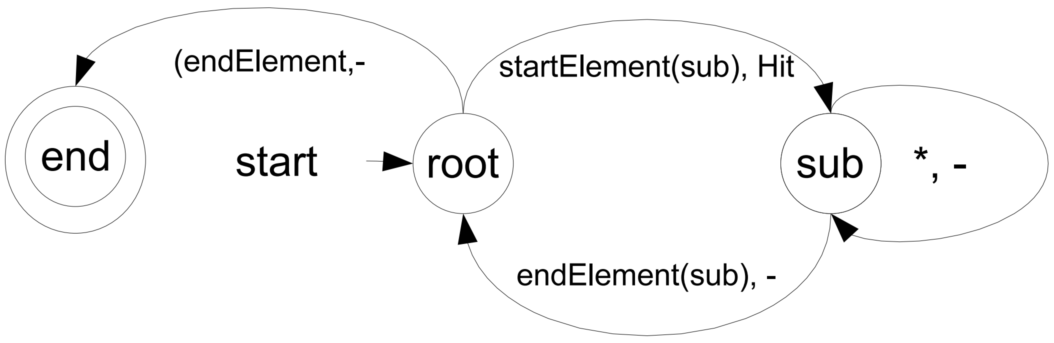


Figure 5.2: Mealy Machine for Child Axis Evaluation

5.1.2 Representing Results

Evaluating XPath queries requires storing temporary and final results. Storing these results separately is a waste of memory since redundant information might be stored. To avoid storing redundant information, we propose the usage of marker structures that include pointers to templates, to mark parts of the XTO that the template represents. The overhead of storing each result separately is avoided by using the existing template.

In detail, a pointer is set to a single SAX event in the XTO template representation like described in Section 4.2. For example, by setting a pointer to an element event, we can explicitly identify an XML sub-fragment with the element as root. Entire complex elements and root elements are accordingly selected by pointing at the entire corresponding XTO. In Figure 5.3, we show a marker object that marks the *sensors* node. This example also shows a list structure to represent the context node list. In this example we furthermore save the position of the represented node in the marker structure. The memory demand of representing results depends on the granularity of the used marker structures, e.g. how many pointers are used. Thus, the overall memory demand for evaluating XPath queries depends on the memory demand of the marker structures and the maximum number of results. An upper bound is given for the descendant-or-self axis (//*), because it selects each node in the entire XML subtree. For this case we can only optimize the memory demand by using small marker structures, which has consequences for the evaluation strategies, like described in the following.

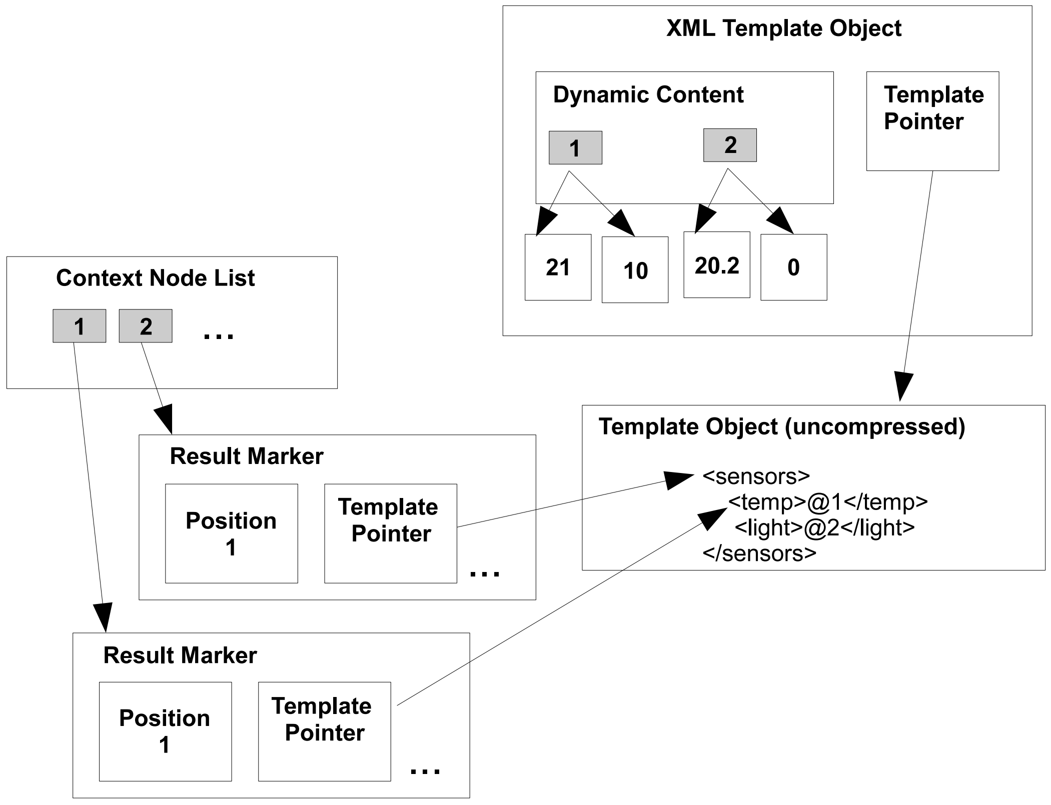


Figure 5.3: Representing Results by using Markers

The marker objects shown in Figure 5.3 are implemented using generic C structs. In Listing 5.3 we show the corresponding representation of an XPath node in the evaluation tree. The representation conforms to the official specification in [234]. XPath is a language for selecting elements of XML data. Updates are not supported in XPath. Hence the XPath nodes must not include the actual content of the XTO but just references to the XTO elements itself. The elements of the node struct are:

1. *elemPtr node*: A pointer to the template of the selected XTO

2. *int arrayOffset*: If the selected node is dynamic content of the XTO, the arrayOffset determines its position in the elem array
3. *char type*: The XPath node type
4. *xmlObjectPtr arrayObj*: A pointer to an embedded selected XTO
5. *nodePtr parent*: An optional pointer to allow direct evaluation of the parent axis as proposed in the next subsection.

Listing 5.3: XTO XPath Evaluation Node

```
struct _node {
  elemPtr node;
  int arrayOffset;

  // Types
  // 1 = Element Node
  // 2 = Text Node
  // 3 = Attribute Node
  // 4 = Dynamic Attribute Node
  char type;

  xmlObjectPtr arrayObj;
  nodePtr parent; // Optional
}
```

A context node list is represented using the structs presented in Listing 5.4.

Listing 5.4: XTO XPath Context Node List

```
typedef nodeSet *nodeSetPtr;
struct _nodeSet {
  nodePtr * node;
  int noel; // number of elements

}
typedef contextNodeList *cnPtr;
struct _contextNodeList{
  nodeSetPtr cnList;
  int pos;
}
```

An XPath location path is processed by evaluating the axis, the node test and the predicate for all nodes in the context node list. Selected elements will be stored in a result node list of type *nodeSet* as shown in Listing 5.4. This list hence also acts as a temporary result list during the evaluation. The next node to be processed is always selected using the position marker *pos* in the context node list. If all nodes in the context node list have been processed, the actual global context node list of type *contextNodeList* will be updated before processing the next location path.

5.1.3 Evaluation Strategies

XPath evaluation strategies can be focused on the processor usage and hence energy conservation or the memory demand. As benefit of our template approach we can choose the granularity of the used marker structures and hence decide for one of the strategies. As an example, we assume that we only use small marker structures consisting only of pointers to current results without pointers to the result's ancestors, like shown in Figure 5.3. Accordingly, we desist from using the *nodePtr parent* in Listing 5.3.

If the XPath query now includes finding the parent of the current context nodes, we have to reevaluate the location step from the root of the document until we find an element that includes the context node as a child. In other words for a context node A we have to rewrite the XPath query, e.g. //A/parent::*, to a new query that looks for a node A as a child, e.g. //*[A]. The strategy of reformulating XPath parent axes was discussed by Olteanu et al. [178]. In contrast to this general approach, to save memory we let the evaluator reevaluate the axes step by step instead of generally reformulating the XPath query and finally evaluating the new query without parent steps.
Following this evaluation strategy we see a trade-off between energy consumption and memory utilization. While including pointers to ancestors like shown in Figure 5.4 extends the memory demand of the marker structure it will save computation steps.
On the other side using small marker structures results into reevaluating temporary results and hence consumes more energy. Depending on the sensor nodes in use, the developer has to choose the optimal strategy. However, as the results in Section 5.4 will show, for today's sensor nodes saving memory can be crucial when it comes to processing XML data.

Another problem during the evaluation process is the support of filter predicates. The XPath 1.0 reference [237] includes unlimited nesting of predicates. As each predicate is evaluated for the entire context node list, an upper bound for the memory utilization is twice the size of the largest possible context list. This might be reached for the descendant-or-self axis (query: //*[.descendant::*]). Due to the low memory capacity of today's sensor nodes, deep nesting leads to high usage of memory exceeding the memory limit. Currently our XTO and XTS implementations can support unlimited nesting but we limit it to one step nesting to ensure stable programs. We believe that one step nesting should be enough for most application scenarios.

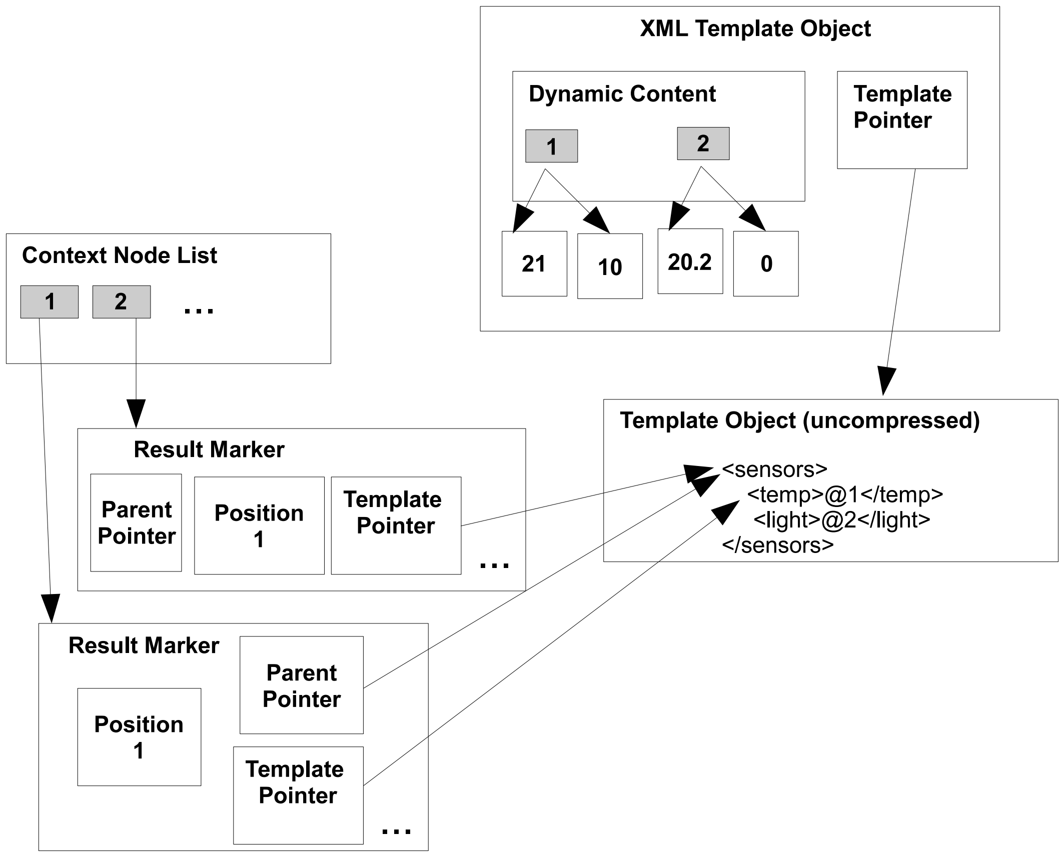


Figure 5.4: Using Parent Pointers in Result Markers

5.2 XML Query Processing on XTSs

The previously discussed XTO approach implicitly contains an object model that is comparable to a tree model like DOM. XPath queries can be directly evaluated using tree navigation algorithms. Using the stream-based template compression approach requires additional functionality because the compressed XML document does not include an object model. Hence, a special concept for evaluating XPath queries on XTS has to be developed.

To allow dynamic data acquisition, we implemented an XPath Engine that evaluates queries on the XTS. To process XPath queries, we enhanced the functionality of the PDA as shown in Figure 4.8 by the tapes for Bintypes and XPath queries. The query evaluation uses a context list, e.g. a given Bintype, and the actual query and outputs an additional result Bintype. The engine supports forward and backward axes. In the following we will discuss the forward axes for a simple introduction. For the backward axes the PDA works in a reverse processing mode which is a significant benefit of the XTS encoding.

We describe the PDA action for the XPath axes as follows:

- **Self Axis**
  The PDA checks the actual identifier (node test). The current state is marked in the Bintype if this test is successful.

- **Child Axis**
  The PDA stores the actual stack height h. Possible result candidates are static elements (marker = (e), complex elements ((o, (v, (w), whereby the candidates are only allowed to be processed at stack height h or $h+1$, e.g. they are on the child level of the context node. Each of these possible candidates is checked for a node test and included in the output Bintype in case of success. The axes evaluation ends when the stack height drops to h-1.

- **Descendant Axis**
  The *descendant* and *descendant-or-self* axes are evaluated using calls of the previously described *self* and *child* routines. In detail, the *child* routine is called recursively and the global result Bintype is constructed by calculating a logical OR of the Bintypes from all recursion levels. The *descendant-or-self* evaluation simply adds a *self* call at the end.

- **Parent Axis**
  The XTS is processed backwards. The PDA stores the actual stack height h. Possible result candidates are complex elements ((o, (v, (w), whereby the candidates are only allowed to be processed at stack height h-1, e.g. they are on the parent level of the context node. Each of these possible candidates is checked for a node test and included in the output Bintype in case of success. The axes evaluation ends when the XTS is read to its beginning.

- **Ancestor Axis**
  The *ancestor* and *ancestor-or-self* axes are evaluated using calls of the previous described *self* and *parent* routines. In detail, the *parent* routine is called recursively and the global result Bintype is constructed by calculating a logical OR of the Bintypes from all recursion levels. The *ancestor-or-self* evaluation simply adds a *self* call at the end.

- **Preceding Axis**
  The PDA processes the XTS backwards. Dynamic elements and static elements, e.g. (d and (e, that are retrieved are possible result candidates. Because the XTS is always encoded depth first, these types of nodes appear preceding to the context node. If a complex element (o is found, it is also added to the list of possible result candidates if this element has been closed on the way by an (* marker. The PDA now only checks these candidates for the node test. The *preceding-sibling* evaluation simply checks if

the results are on the same layer in the virtual XPath tree, e.g. have the same stack height.

- **Following Axis**
  The *following* axis is the most complex axis for XTS XPath axis evaluation. The reason is the implicit order in the XTS encoding, that represents a depth first tree order. We implemented this axis using separate calls of descendant, preceding and ancestor-or-self. The resulting Bintypes are processed with a logical AND. Finally the logical NOT is processed and the resulting Bintype is the result of the following Axis. The PDA then processes the node tests and predicates. The following-sibling axis is directly processed by processing the stream like for the child axis but only accepting (e and (d and (o before the stack drops to *h-1*. Accordingly, we can use the query reformulation rules as introduced in [61], e.g. following::n is equivalent to ancestor-or-self::node()/following-sibling::node()/descendant-or-self::n.

In Section 5.1.2, we introduced the marker concept to represent context nodes in the XTO XPath implementation. For XTSs we use the Bintype concept as introduced in Section 4.3.5. For every qualifying node during XPath location path evaluation, the bit of the corresponding state transition of this node is set in the Bintype. For details, how to set a bit in the Bintype we refer to our previous discussion in Section 4.3.5. Using this concept has the following benefits:

1. The Bintype is the most compact representation of a context node list. We discuss this benefit in the evaluation in Section 5.4.
2. The Bintype is already serialized and ready to be transmitted directly in the network. A transmitted Bintype can be used e.g. for applying the XPath evaluation result on replicated XTSs in the network without reevaluation.

Nevertheless, we point out that the Bintype concept requires processing the entire XTS every time a single context node is required.

After presenting practical solutions and concepts for evaluating XPath queries on XTOs and XTSs, we introduce XML query dissemination techniques and optimizations concepts in the next section. These approaches are fully applicable for the XTO and XTS implementations and have been evaluated as shown in Section 5.4.

5.3 XML Query Dissemination and Optimizations

In this section we discuss how XML queries are inserted into WSNs and how the processing of queries and results can be further optimized for energy efficiency.

5.3.1 Query Dissemination

The previous aspects affect the efficient dynamical evaluation of XPath queries on single sensor nodes. Besides, XPath query dissemination and decomposition bring up new open issues and optimization possibilities. In general, in our work XPath queries are injected into the network via a gateway that can be located anywhere in the entire WSN. Like in previous approaches [156] the queries are then routed to each sensor node. This can be done by using a fixed topology, like described in [156], or by using full broadcast communication and hence avoid communication bottlenecks. By remembering active query ids, answering single queries multiple times is avoided. Furthermore, we can support multi-queries in our approach. While this approach is straight forward, further optimizations can be done by supporting adaptive query decomposition with the help of indexing and caching structures. We give an example in Figure 1.3. The importance of using indexing and caching structures in XML WSNs stems from the fact that a WSN comprise sensors sensing different data in different regions, as already described in the introduction of this paper. Thus, spatial location and type of physical phenomena specified in the query could be a query decomposition criteria. In detail, in Figure 1.3 the subnetworks I, II, III include different data and data structures. To optimize the query dissemination and avoid sending queries to parts of the entire network that can not fulfill the query criteria, we target using caching and indexing nodes for subnetworks in the entire WSNs. For example, the nodes of region I in Figure 1.3 have the ability of caching subnetwork data to avoid inserting the query in the corresponding subnetwork. Besides, these nodes can be used for indexing the distribution of XML data and hence optimize the query dissemination. While these optimizations are still in development and a detailed description is out of scope of this thesis, we already tested the main functionality as a prototype and gained promising first results. Furthermore, we discuss a fully implemented caching feature for query results in the next chapter.

5.3.2 Processing and Transmitting Query Results in the Network using Template Caches

One of the main aspects in previous work about data management in WSNs was to lower the communication demand for delivering query results to the gateway [156, 253]. Evaluating XPath queries might result into large sets of XML documents. As denoted previously, evaluating the descendant-or-self axis for the root node without filtering resulting elements by name (e.g. //*) results into a set consisting of the document itself and all sub-documents. By using the $XOBE_{SN}$ XML template compression approach, we can adapt the marker evaluation strategy by sending a template for the surrounding documents and mark included qualifying sub-documents. The remaining part of the result transmission, is to serialize the result templates before transmission. We consider a template consisting of different possible parts:

1. static attributes or static elements
2. dynamic attributes or dynamic elements
3. dynamic XML template objects, e.g. separate complex XML fragments included in the XML document

While for case 1 and 2 the result template is a plain copy out of the template of the reviewed document, for case 3 we have to flatten the template, since we deal with a complex object with an own template that is not included in the document's template. At the end of this step the result template can be further compressed using standardized compression techniques and is then ready to be sent together with the dynamic content to the source of the query, e.g. gateway. For querying large WSNs it is most likely to assume that the result template of each node will be partly or fully equal. In these cases, we suggest a further optimization step that tries to avoid sending redundant information like early in-network aggregation. In our approach each sensor node acts like a global template cache. For the query source it is only important to have exactly one template once all results have arrived. This template can be used together with each dynamic content part to produce every node's own result XML document. Especially for very large networks, this idea promises good results on lowering the communication demand. An open issue is which nodes (template caches) should be responsible for sending the query. We suggest three different strategies:

1. randomized template transmission that depends on a demanded probability of receiving at least one template at the query source
2. distributed template caching with explicit cache data retrieval
3. template estimation at the query source

While we investigate the first solution in Section 5.4, to analyze the general communication optimization possibilities, the solutions 2. and 3. are out of scope of this thesis. They are currently addressed in our ongoing and future work.

5.3.3 XML Query In-Network Aggregation Support

Previous work has shown that in-network aggregation is a means towards energy optimized query evaluation. This is even more true for handling large sets of XML results. Although XPath supports general aggregation through functions, an extended aggregation method to support heterogeneous aggregation on complex XML data would be desirable. Therefore, we suggest supporting complex aggregation by sending multiple aggregation rules for query result templates. While the rules can be expressed in the return statement with the support of XQuery, for now these aggregation rules are separately attached to the XPath query and gradually applied to the query result. In this way different aggregation functions can be applied to different simple and complex elements and attributes of the query result. The aggregation itself is organized using existing techniques, e.g. TAG [155] or Synopsis Diffusion [170]. However, having multiple different aggregation functions in different hierarchies of the resulting XML document, duplicate sensitive aggregation requires additional coordination. A simple solution is the aggregate-all approach, where the aggregator has to process the entire result. A more complex solution is to distribute the aggregation, so that the aggregator processes only parts of the preliminary result. This shows up new possibilities of energy conservation and failure tolerant result processing by having multiple ports for computing complex aggregation functions. However, due to the limited space of this thesis, we leave a detailed description of this work in progress out for future work.

5.3.4 Bounded Continuous XML Queries

As denoted in Section 2.2.2, model-driven queries have been introduced for wireless sensor networks to further reduce the communication demand using stochastical estimation and filtering. One important kind of model-driven queries are bounded continuous queries that have been introduced in our previous work for efficiently evaluating queries in the world wide web [132, 133].

Bounded continuous search queries (BCSQ) are queries that include a quality demand that needs to be respected to reduce the number of query results that are presented to the query issuer. One example are *top-k queries* which restrict the result delivery to the best k results in respect to a given quality degree, e.g. result scoring. The idea of bounded continuous queries can be extended for wireless sensor networks. While a user is often interested in sensor values and analyzation results that suffice a given quality degree, the previous approaches on evaluating such bounded continuous queries can be used to filter unnecessary temporary results deep in the network and therefore reduce the communication demand significantly.

In detail, in wireless sensor network deployments the user is often interested in continuous information, e.g. measured data, over a long period. Therefore, in [156, 182] queries are mostly continuous. For real-time sensor network deployments like the ALERT deployment [7] and other forecast scenarios it is especially important that measured data is checked immediately to trigger events in case of an emergency and sensitive situations respectively. An approach to support alert and event sensor networks is an external storage or so called dumb data collection sensor network [182]. By using an existing sensor network query engine each measured value is simply transmitted to the base station and evaluated immediately. Although this could be a reliable approach, its main disadvantage is the potentially high energy consumption when data is sampled with a high frequency or resolution. Hence, we argue that data should be filtered within the sensor network.

BCSQ are a new means towards energy efficient evaluation within wireless sensor networks especially for alert and event based queries. Analogous to the requirements of an optimal continuous query system in the WWW, alert and event based queries in wireless sensor networks have two basic requirements:

1. A notification about important sensor data has to be immediate (real-time notification).
2. The limited energy capacity of wireless sensor networks requires only to send messages if something important happens, e.g. the best sensor value to a given criteria.

While the first requirement is the same for either BCSQ in the WWW as in sensor networks, the second requirement has also similarities to the search in the WWW. While for evaluating queries in the WWW a result set should not become unbounded to avoid user message filtering, in sensor networks we want to avoid sending unimportant messages to save energy and hence extend the networks lifetime.

To point out the possibilities of BCSQ in wireless sensor networks we discuss an extension of the XPath query evaluator to allow bounded continuous XPath queries like:

```
forcon trigger=1TimeUnit  start=now end=100
estimated greatest 1 in
//sensors/temp
```

This example query, that is given in a BCSQ extension of the XPath [13] query language with *forcon* signalizing a new BCSQ, has the purpose to extract the maximum temperature. It has a duration of 100 time units (i.e. hours), which correspond to 100 temperature sensor values. For simplification we only want to find the maximum temperature in this period. Besides, it is important that sensor nodes process any measured data immediately. An extension to find the greatest k>1 temperature values immediately has been discussed in our previous work [5, 6]. As described before, the standard approach to evaluate this query is to send each temperature value to the base station to calculate the maximum temperature after 100 time units (retrospective analysis). Another approach is to estimate the maximum temperature randomly. However, in [97] we showed that both approaches have drawbacks, e.g. either in energy demand or result quality.

In our previous work [132, 133], we further showed that our BCSQ strategies are able to find a high percentage of maximum ranking scores, which motivates its usage in a sensor network scenario where the task is to find the maximum temperature. While the kSSP approach [132] works without any further knowledge of a distribution of the sensor values, the distribution based methods [133] require this knowledge. However in typical sensor network scenarios like [248] historical data is often present (static distribution) or the query duration is long and the distribution learning strategy (dynamic distribution) is applicable. As a result, the introduced bounded continuous XPath queries will significantly reduce the communication demand if the concepts of our previous work [132, 133] are applied for evaluation. We therefore discussed a fundamental evaluation in our paper in [97]. Due to the space limitations of this thesis, we refer to the paper for this topic. Nevertheless, we again point out that the introduction of BCSQs in wireless sensor networks has a lot of benefits regarding the energy efficient query evaluation and should be investigated in more detail for future work.

5.3.5 Data Caching

The evaluation of queries in-network can be significantly optimized by using data caches on the route from the gateway to the actual data source as proposed in [156]. By letting the data cache answer the query the query transmission distance is limited and hence the overall communication demand reduced. Previous approaches as presented in [57, 156, 190] presume dedicated network topologies and are very limited in their practical applicability. Open questions in the area of data caching in wireless sensor networks are:

- How are data caches placed?
- How can data caches be kept consistent? How are they updated?
- How are data caches localized and used respectively?

Due to the complexity of this topic, we discuss data caching in wireless sensor networks separately in the next chapter. We further introduce a dynamic approximative caching scheme that includes optimized concepts for cache coherence and localization.

5.4 Evaluation

To evaluate our XTO and XTS strategies on evaluating XPath queries on sensor nodes, we have implemented an XPath engine that is capable of a large subset of XPath 1.0 [237]. This includes support for all element and attribute axis, filter support, operators and node tests. Due to their low importance for sensor network applications, we left out support for comments and processing instructions.
There is a subset of built-in XPath functions, e.g. *contains* and *position*, which we have implemented. However, implementing other functions in our query engine is straightforward and left for future work. The XPath engine has been implemented in C running on sensor nodes that make use of the iSense operating system [45]. To verify the functionality of the XPath engine we have used the common XPathMark benchmark [70]. We ran the

XPathMark queries directly on the sensor nodes platforms (Pacemate and iSence Core Modules) and evaluate the functional test. The available RAM was again 96kByte shared for program and data (heap memory was ≈15kByte, program memory was ≈81kByte).

As proposed in Section 4.4, we explicitly tested the following evaluation criteria:

- *Query Evaluation Memory Efficiency:* The memory efficiency can also be verified for evaluating queries dynamically on the sensor nodes. Hereby as in [102] we evaluate the size of memory needed during dynamically evaluating XPath benchmarks.
- *Query Evaluation Runtime Efficiency:* The runtime demand for evaluating queries needs to be evaluated as it is directly linked to the runtime energy consumption of the nodes. As sensor nodes are devices with strict energy limitation, saving as much energy as possible is a design issue to guarantee long term network deployments.

In the following subsection, we present the main evaluation results for the memory efficiency and the runtime efficiency of evaluating the queries of the functional test of the XPathMark [70].

5.4.1 Functionality Test

The first test covers the functionality of our XPath engine implementation. Therefore we ran the XPathMark benchmark on a sensor node and evaluated the functional test cases, that are described in detail in [70] for both solutions (XTO and XTS). The test cases evaluate if the engine and the data binding framework are able to process all possible XPath queries, e.g. all axes.

In Figure 5.5, we show the dynamic memory demand of evaluating the XPathMark axis tests which were measured by the iSense memory profiler. The x-axis denotes the test case and the y-axis denotes the memory demand in byte.
As a result, every test delivered a valid result. For both presented approaches the memory demand is still far under the limitation of the sensor nodes.

Next, we were explicitly interested in a performance comparison between both approaches. For the XTO approach we get the following results: the child axis evaluation (e.g. A1) needed less memory than the descendant axis (e.g. A4). The backward axis needed more memory due to temporary results, whereby we see a difference between the parent axis (e.g. A5) and the preceding/following axis (e.g. A9). Due to the limited memory resources, we reevaluated forward axes to evaluate backward axes like described in Section 4.2. Using complex marker structures like parent pointers resulted into a memory demand that was more than twice as large than the demand shown in Figure 5.5. On the other side, caused by the memory energy trade-off, we expected the backward evaluation to be slower and

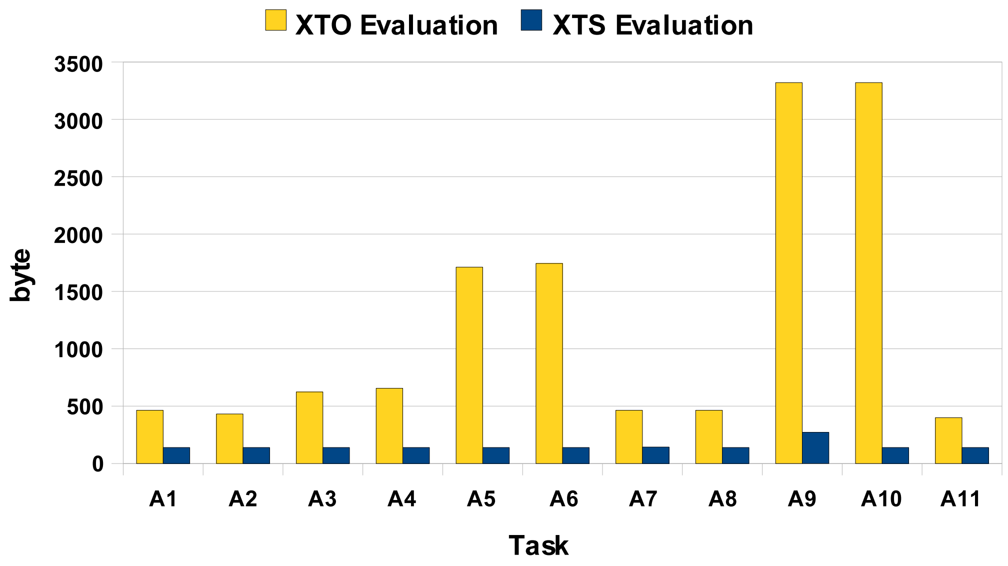


Figure 5.5: Memory Demand of XPathMark Tasks

more energy consuming. Like described in Section 4.2, we have to repeat the evaluation of the forward axis at least by the number of ancestors.

In Figure 5.5, we also included the XTS results for evaluating the functional tests of the XPathMark. We see that the memory efficiency of the XTS approach that has been shown in the previous section for the general data management is also given for the XPath evaluation. By using the Bintype concept that has been introduced in Section 4.3.5, XPath results, e.g. context node lists, can be stored with nearly constant memory that is significantly lower than using lists of pointers in the XTO approach. Furthermore, we can directly evaluate backward axis due to the pushdown automaton that is capable of processing the encoding streams in both directions.

The positive memory efficiency for evaluating XPath queries in the XTS approach comes at a price of higher runtime demand. Preliminary results can often not be adressed directly without letting the pushdown automaton process the entire stream to the corresponding context node. We evaluated this fact according to our previous evaluation in [102]. In Figure 5.6 and 5.7 we see the evaluation results for the energy consumptions for the test cases that show the prognosed runtime efficiency of both solutions. We measured the processing cycles and put them into relation to the upper bound of energy consumption per cycle [45]. Figure 5.6 shows the results for the Pacemate platform, while Figure 5.7 shows the results for the iSense Core Modules. Both results are relatively comparable. They differ mostly in the absolute energy consumption depending on the corresponding platform.

Evaluating backward axes in a memory efficient way consumes more than three times the energy of evaluating forward axis. Besides, we see that in contrast to the memory demand there is no difference in the results between parent and preceding/following axis.

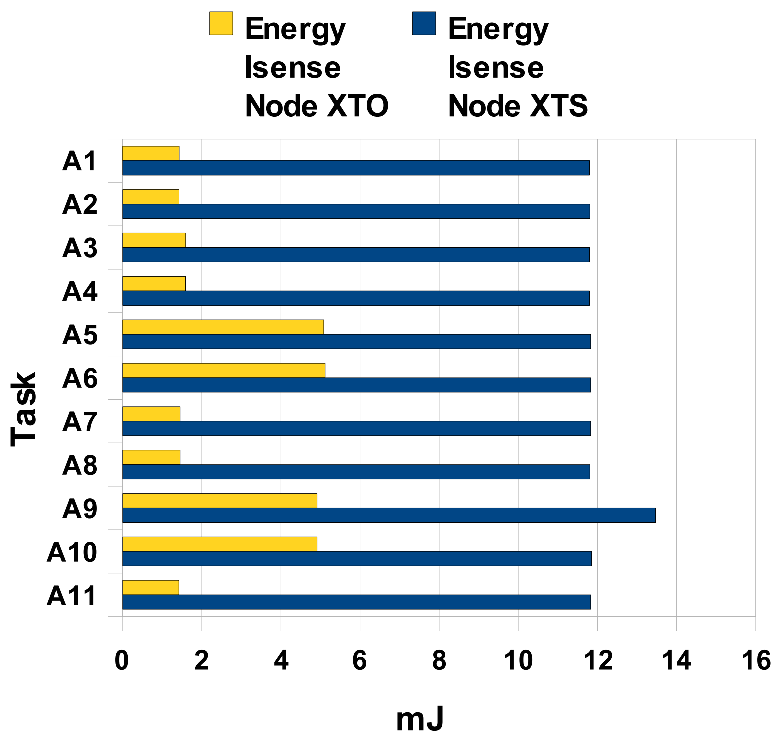


Figure 5.6: XPathMark Energy Consumption Results for Pacemate Nodes

When comparing with the XTS approach, we see that the XTS approach's energy consumption is up to five times the XTO approach when evaluating the functional test of XPathMark. Thereby we only see a slight difference between performing the different tasks. This is caused by the extensive runtime demand of the pushdown automaton that is similar for all tasks based on processing preliminary results and the absence of direct access over pointers. This trade-off has to be accepted based on the memory efficient data storage of this approach.

Detailed Engine Memory Demand

We evaluated the program memory demand of the XTS approach and compared it to our previous XTO approach. It is shown in Table 5.1. The actual *XML Framework* is only given for the XTO approach as it includes generic object structs and help routines for handling pointers and objects. As the XTS approach is based on processing plain encoding strings no further help routines are provided. The entire work is done by the pushdown automaton during evaluation. This pushdown automaton is included in the *Debug Routines* as it is needed for processing an XML output. For the XTO approach the *Debug Routines* are much more simpler as only one depth first search like tree algorithm is needed for processing the tree like XTO structure. The *XPath Engine* itself has been improved for the XTS approach.

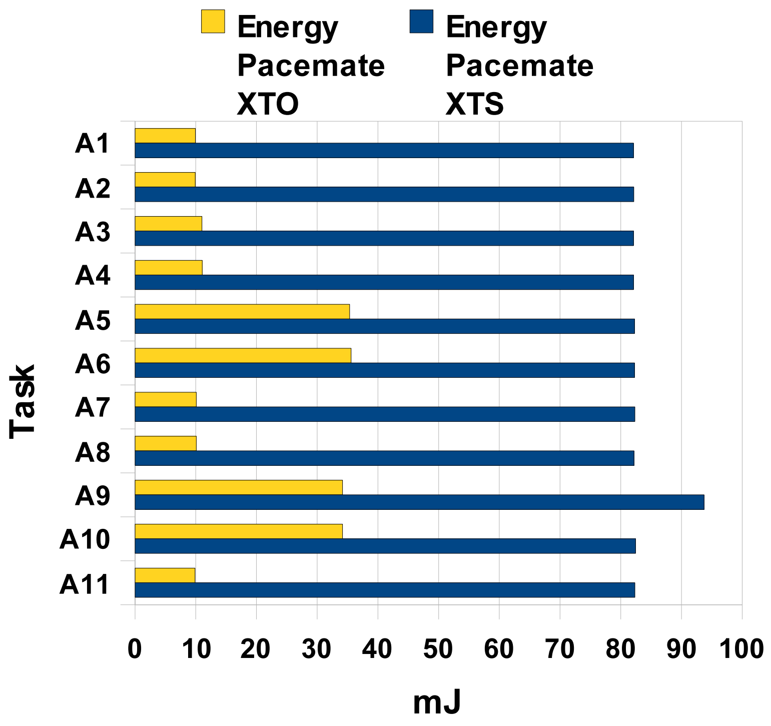


Figure 5.7: XPathMark Energy Consumption Results for ISense Core Modules

It now needs approximately 8Kbyte less memory and also uses the generic pushdown automaton of the *Debug Routines*. However, we point out that the XTO *XPath Engine* could also be improved, e.g. up to 30% memory saving by reusing shared components, that have not been yet optimized. We left this for future work. The actual *XPathMark Program* size for the XPathMark function test is only minimal different between both approaches.

Program Functionality	**XTO Memory Demand in bytes**	**XTS Memory Demand in bytes**
XML Framework	7744	0
XPath Engine	18734	11236
XPathMark Program	704	820
Debug Routines	668	3098
Sum	27850	14390

Table 5.1: Program Memory Demand of XTO and XTS

In summary, we show that both solutions provide an XML data management framework that is fully applicable for today's sensor node products, while keeping enough space for extended applications. By reviewing the results of the data management tests and the XPath engine evaluation we show that both solutions (XTO and XTS) have their application domains. The benefit of the XTO approach is the direct access to sub elements of the XML data. If this is a design issue than this approach can be considered as optimal.

It further provides a very good memory efficiency. However, this memory efficiency is further optimized by the XTS approach which is the best solution for sensor networks with strict memory limitations when processing and storing large scale XML data. We point out that the XTS approach does generally need more energy when processing XML data or XML queries. However, the energy consumption should be significantly lower than during communication as shown in [155]. This can be an issue for XTS when it comes to transmitting XML in the network without further optimizations as described in the next section.

5.4.2 XPath Result Transmission

In Section 4.2, we present a strategy for processing XML results in the network that benefits from the template-based XML compression approach. In Figure 5.8, we show the evaluation results.

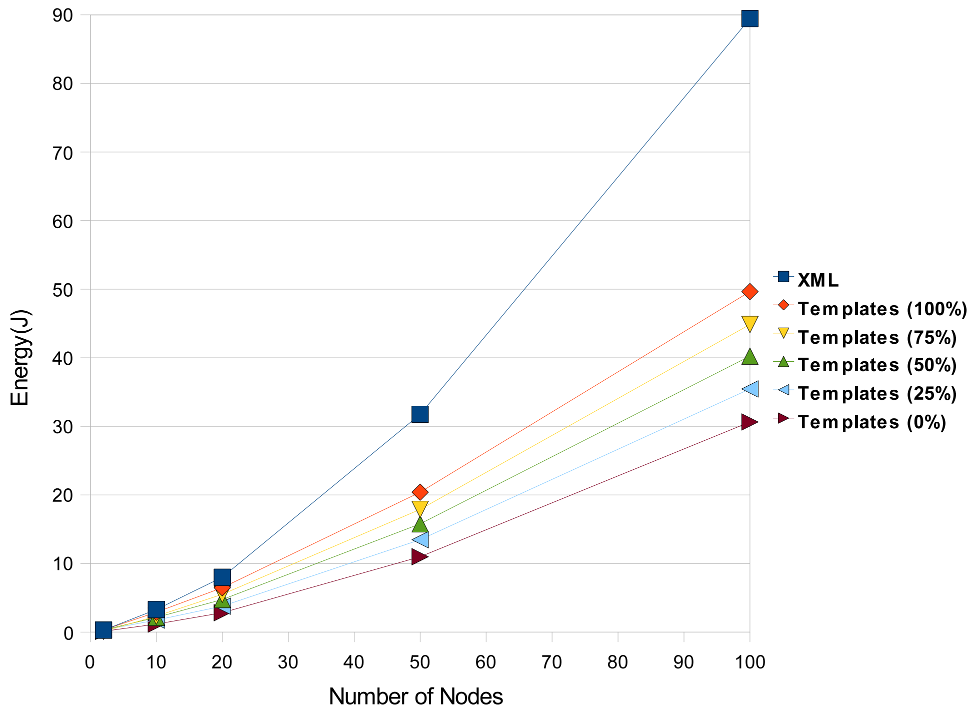


Figure 5.8: Transmission Optimization

We tested different application scenarios, e.g. XPathMark result documents and result documents, that are based on typical sensor data measurements like the example in Listing 4.13. The x-axis of Figure 5.8 denotes the number of nodes that were participating in the query evaluation. The y-axis denotes the energy for the overall communication until the result was received by the gateway. The energy is related to the bits that have been sent and is an upper bound for the maximum transmission range [45]. We compared different strategies: first the transmission of the plain XML string, second the transmission of serialized

XML template objects, like described in Section 4.2. Hereby, we varied the probability of sending the template and the dynamic content or only the dynamic content (from 0% to 100%).
As a result, the template-based approach outperforms the string based approach. Furthermore, it becomes obvious that the more it can be avoided to send templates the more the compression ratio increases. While in large networks we can assume that sending only a small number of templates might suffice to process the results at the gateway correctly, the 0% approach assumes that we can estimate the template as described in Section 4.2. It is therefore a lower bound for the results. The results are a motivation for further investigating template estimation and caching strategies for templates.

5.4.3 XPath Result Aggregation

We further investigated XML data in-network aggregation like described in Section 4.3. We chose an application like the example in Listing 4.13 and extended it by the number of measured sensor values. We issued several queries on the data generated by the application each including heterogeneous aggregation strings like described in Section 4.3. The results in Figure 5.9 show that in-network aggregation can reduce the communication demand by a factor of three. It is shown that a combination of the template-based result transmission as described in Section 4.2 and in-network aggregation can further optimize the communication demand.

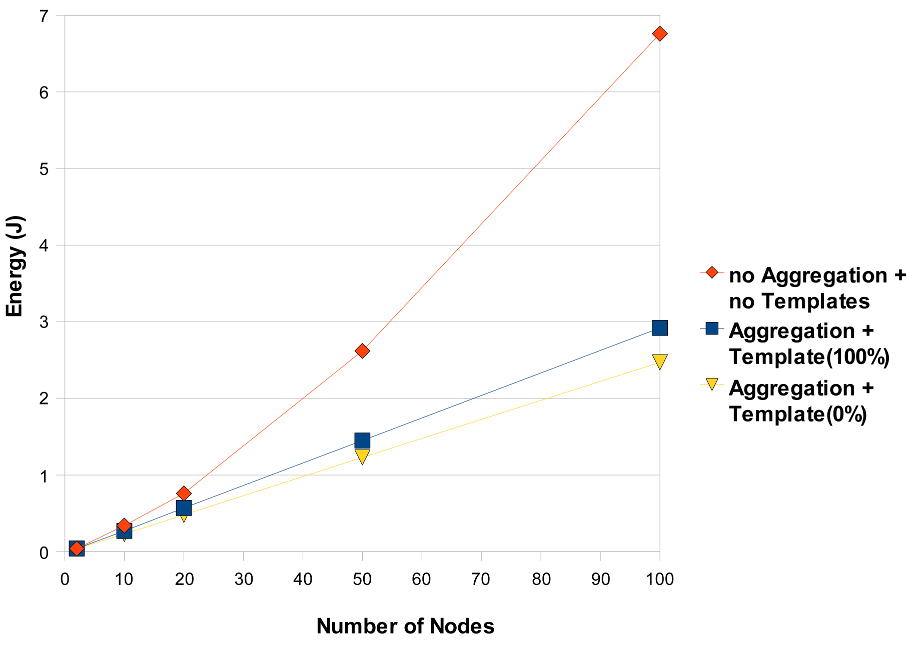


Figure 5.9: XML Aggregation Results

Chapter 6

DACS: A Dynamic Approximative Caching Scheme for Wireless Sensor Networks

The previously presented data acquisition optimization strategies presume the query result retrieval from deep within the network at the data sources. Nevertheless, the communication demand can be further reduced by using only a limited part of the entire network to evaluate the query. One possible approach is to use data caches instead of sending queries to each data source. As a result, the communication demand can be significantly reduced. In this chapter, we introduce the **D**ynamic **A**pproximative **C**aching **S**cheme: **DACS**. Previous work on data caching in wireless sensor networks require the usage of hierarchical communication topologies with deterministic data routes, e.g. [155]. Unlike these approaches, DACS works without topology assumptions and is robust against communication limitations. The framework consists of a dynamic distribution of data caches in the sensor network. In order to extend the lifetime of the network and take care of the unreliable communication channels, a weak cache coherence is discussed that is based on an approximative update policy. By attaching a result quality requirement to a query, DACS automatically retrieves query results from caches nearer to the data sink and further ensures that the degree of result quality is not violated. As a result, queries do not need to be sent to all sensor nodes deep within the network which reduces the communication overhead. Evaluations show that by using DACS and by accepting a minimal deviation in the query result the network's lifetime can be significantly enhanced.

In Figure 6.1, we give a simplified example for a network running DACS. In a uniform distributed network data source nodes are cached throughout multiple layers. Gateway nodes are used to send queries into the network. In this example, we define a data source node (*SN*) and four gateway nodes (*GW*). Every other node is a possible cache node. However, unlike this simplified example, DACS supports unlimited, randomly placed data source nodes, gateways and cache nodes, e.g. each data source node is cached separately by DACS and each data source node acts also as a cache for other nodes.

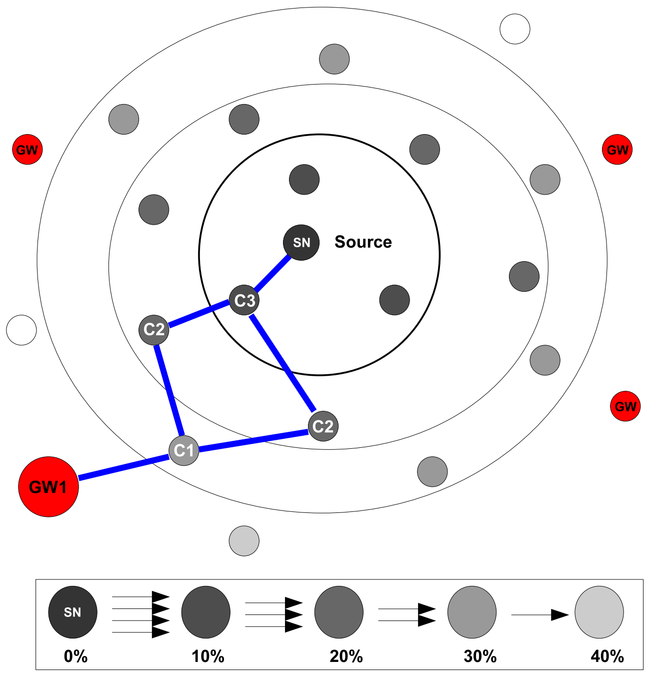


Figure 6.1: Overview DACS Simple Network Model Example

The basic concept of DACS is to use cache nodes instead of the actual data source nodes for retrieving information, e.g. temperature values. Hereby, the communication demand can be significantly reduced by using caches that are layered nearer to the gateway than the actual data source, e.g. using the caches C1, C2 or C3 instead of the data source SN. However, cache nodes need to be kept coherent. Sensor node communication in wireless sensor networks is generally unreliable. As a result, strict coherence or even consistency is not possible. Besides, keeping every cache node coherent comes at high communication costs for rapidly changing data at the data sources. To overcome these limitations, DACS uses an approximative update policy. As shown in Figure 6.1, the cached data can deviate from the actual source data based on the distance between cache and data source. For a simplified example, we can define a maximum deviation of the cache value for each source-cache distance as shown in Figure 6.1, e.g. using a linear increasing deviation of 10% per hop.

As a result, the estimated communication demand for cache updates decreases with increasing distance from the data source as highlighted by the number of arrows. In this chapter, we introduce how DACS implements this approximative cache coherence. Using approximative caches requires using model-driven queries since it is not obvious how deep queries should be sent in the network to accordingly reach caches with lower deviation. DACS hides the cache localization process to the user by allowing to issue queries with additional demand of result quality, e.g. get temperature values that might include an overall

maximum deviation / error of 30%. In this chapter, we discuss this localization approach for unlimited data sources and data caches.
The remaining parts of the chapter are organised as follows: In the next section, we give an overview on important previous and related work. In Section 6.2, we give a detailed introduction into DACS covering issues of Cache Placement (Section 6.2.1), Cache Coherence (Section 6.2.2) and Cache Localization (Section 6.2.3). In Section 6.3, we discuss the general architecture of DACS and give details on its implementation. Finally, in Section 6.4 we evaluate the framework and show results on the communication efficiency and the robustness of the framework.

6.1 Related Work

In this section, we present previous work related to data caching and data management in wireless sensor networks. As proposed in Chapter 2, previous work in data management in wireless sensor networks was initially focussed on deep in-network aggregation and data acquisition optimizations to save energy during processing query results [156, 253]. Using caching structures to optimize in-network query processing in wireless sensor networks was firstly discussed for the query engine TinyDB in [155]. While there has been a clear recommendation for using caches to evaluate queries closer to the actual data sink and hence to save energy, the used strategies provide only a simple round-based caching scheme.
In this approach, in a static aggregation tree aggregation values of child nodes are cached by their parents for a determined number of rounds. The actual aggregation values are not regarded and the caching scheme is strictly connected to the TAG tree topology. Beside this approach, other caching strategies for wireless sensor networks have been presented in [35, 144, 145, 190, 195]. Most of these approaches require the usage of hierarchical communication topologies with deterministic data routes, e.g. TAG [155]. Hence, we denote them as hierarchical caching strategies. The strategies can be further divided in non-approximative and approximative approaches. In detail, Shashi et al. [190] describe an optimal cache placement strategy in tree topologies by finding nodes in Steiner Data Cashing Trees in a scenario where multiple subscribers are receiving data from one source. Multi-source scenarios like described in this work are left out for future work. Chand et al. [35] describe a cooperative caching scheme to improve data access performance and availability in mobile ad-hoc networks. Their work is focused on a new utility based cache replacement strategy in contrast to a usage based policy. However, this approach is not optimized for energy constrained wireless sensor networks and therefore seems not to be applicable in the presented way. In [195], Rahman et al. propose strategies for improving the energy efficiency of wireless sensor networks. The presented caching strategy is focussed on avoiding unnecessary sensing by estimating the data change frequency, comparable to the acquisitional data processing strategy of TinyDB. Strategies for in-network

caching to optimize query evaluation like described in this thesis are not presented. Jung et al. [114] focus on an external cache-based sensor network bridge to avoid querying the entire wireless sensor network by using cached results. The approach works for non-constrained external devices (e.g. gateways) and hence is not applicable for in-network query optimization.

6.2 The DACS Framework

In this section, we give a detailed description of the DACS Framework. The basic concept of DACS is to provide a general data cache solution that does not rely on any given topology assumptions. The design goal of DACS is to reduce the communication overhead by letting queries be evaluated by data caches on the route to the actual data source. Reducing the communication overhead increases the lifetime of the network significantly.
For setting up a general caching scheme for wireless sensor networks, the following issues need to be reviewed:

- **Cache Placement (Section 6.2.1):** A general strategy on where to place data caches on the communication routes needs to be resolved. A general network model is the basis of this strategy.
- **Cache Coherence (Section 6.2.2):** Data caches need to be updated when new data occurs at the data source.
- **Cache Localization (Section 6.2.3):** Queries need to be redirected to adequate data caches for evaluation. The localization process should be hidden to the user, e.g. by a black box behaviour.

6.2.1 Cache Placement and Network Model

Wireless sensor networks are highly dynamic networks. The exact position of nodes after deployment, e.g. out of a plane, is often not precise and the lack of communication robustness does not guarantee fixed data routes in general. As a result, the placement of data caches cannot be verified before deployment and the caching structure needs to be set up during runtime autonomously. DACS is able to adapt the distribution and placement of data caches dynamically in case of changing network conditions. Additionally, no assumptions on the network topology are made. The organization of the cache structure is a collaborative decision of the network itself.
The following general network model is defined to make DACS suitable for most application scenarios:

- A network consists of a uniform distribution of N nodes.

- In the network each node except the gateway nodes can produce data, e.g. measure environmental information. Measurements are taken on predefined intervals continuously.
- The general DACS approach supports multiple gateways, whereby each gateway is managed separately in identical manner. However, in the following discussion we assume the network to have one dedicated gateway for better understanding.
- To avoid bottlenecks on the routing path, DACS relies on using broadcast communication. We desist from using fixed topologies to improve failure tolerance and to support a maximum number of deployment scenarios. However, to avoid energy inefficient flooding of the network we use ring-oriented, directed communication as proposed in [170].

In DACS we desist from using dedicated cache placements. Instead, every node besides its own measurement task is a potential cache for other nodes. Thereby, nodes decide which data they cache based on their distance to the data source that is determined by the hop count of update messages. Nodes can then be classified concerning their membership to distance layers. We therefore define two different layers:

1. *Cache Layer:* The cache layer defines the distance between the gateway and a cache node, denoted by d_{GC}. For example, in Figure 6.1 *C1* is on cache layer 1 for Gateway *GW1*.
2. *Update Layer:* The update layer defines the distance between the data source and a cache node, denoted by d_{SC}. For example, in Figure 6.1 *C1* is on update layer 3 for the source node *SN*.

As denoted previously, we use a ring-oriented broadcast communication as proposed in [170], e.g. cache results are sent to the corresponding gateway over decreasing cache layer and update messages are sent to the caches over increasing update layer. The actual placement of the cache is now adjustable by placement rules, e.g. caches are placed every second or third layer, depending on the memory and energy resources of the application scenario. The actual update logic is defined by the cache coherence protocol that is described in the next section. Finally, by avoiding dedicated caches and instead using cache layers we further introduce an implicit data replication which optimizes the node failure tolerance and makes the networks more stable concerning communication path failures. For using dedicated replication techniques we refer to our previous work in [171, 172] which is out of scope of this thesis.

6.2.2 Cache Coherence

Cache consistency predefines that for each point in time the cache is consistent to the data source whereas cache coherence demands that the cache is consistent to the data source during the evaluation of queries. Cache coherence therefore significantly reduces the effort of keeping the cache up-to-date and hence the communication demand. However, both claims can mostly not be satisfied in wireless sensor networks because of the unreliable communication and the energy demand of continuous cache updating that conflicts with the general hardware restrictions in wireless sensor networks. In detail, the times when queries have to be evaluated are generally not predictable resulting in a continuous updating process of the caches which significantly reduces the lifetime of the entire wireless sensor network due to the communication overhead.

To overcome these problems, DACS introduces an approximative cache update policy. Each cache node does not store the actual value of the data source but rather stores a value for that a maximum deviation / error is guaranteed. An update is only processed if the maximum deviation is exceeded. The maximum deviation for each cache item is set based on its distance to the actual data source.

In Figure 6.1, we give an example for an approximative update policy with linear increasing error (10% per hop on distance d_{SC}). In the following, we give a detailed description on how DACS supports this policy.

As described in the previous section, nodes are classified based on their membership to cache and update layers. Each logical update layer consists of nodes with the same distance d_{SC} to the data source and includes a maximum deviation / error regarding the cached values and the values of the actual data sources.

This error is defined by a function $\Upsilon(d_{SC}) \rightarrow e_x$ that calculates the maximum error e_x for a given cache data source distance $x = d_{SC}$. The function is initially known to all nodes based on the application scenario and the efficiency predefinitions. For Figure 6.1 it can be defined as $\Upsilon(d_{SC})$ =$d_{SC}/10$. In general, the higher the steepness of the function, the lower the estimated communication demand in general for updates. The high impact of choosing the error function according to efficiency predefinitions that can result into significantly lower communication demand is also shown in the evaluation of this work in Section 6.4. Moreover, the function can be adjusted concerning the estimated number of queries per update as also shown in Section 6.4.

On incoming update messages, nodes decide based on the function $\Upsilon(d_{SC})$ whether an update message needs to be processed, e.g. the value needs to be cached and the message needs to be forwarded to higher update layers.

DACS currently supports three forward policies, that can easily be extended for future work:

1. *Decision at Source Node:* The source node tracks the progress of cache values based on a virtual layer scheme and determines itself how many hops an update needs to be sent in order to verify the update policy. This approach prevents policy violation and is denoted as stable approach.

2. *Forward If Updated:* Cache nodes forward as long as the update policy results into cache updates. The forward process is a dynamical decision on the update layer path.

However, monotonic changes in source values can produce temporary violation of the demanded deviation gradation, e.g. higher update layers exceeding the maximum deviation temporary. It can be shown that the maximum error can be nearly squared. Nevertheless, this simple solution performs well in average without policy guarantees.

3. *Weighted Forward If Updated:* The problem of the error policy violation of the *Forward If Updated* forward strategy can be solved by dynamically weighting the function $\Upsilon(d_{SC})$. With increasing distance from the data source to the data cache the function weight is increased to smooth out the squared error. However, this will result into more updates and is therefore less energy efficient due to the communication overhead

In the experiments of this work we have used the forward policies 1. and 2. which both ensure a stable performance. The forward policy 3. is only needed if monotonic sensor values are likely to occur. We give an example for this scenario in Figure 6.2. The source node continuously samples new temperature data. We assume a monotonic increase of 2 degree Celsius per sample. In Phase I all nodes are updated due to the initial start of DACS. Hence, no errors exist and the cache nodes are consistent. In Phase II all nodes are weakly consistent as the cache a value that deviates from the source node within the error tolerance. In Phase III the nodes on update layer 1 (ahc = 1) need to be updated. However, the nodes on update layer 2 decide not to update due to the not violated tolerance and do not forward the message. In Phase IV the problem of monotonic sensor values appears for the first time. The nodes on update layer 1 do not update and hence do not forward the update message. Accordingly, the nodes on update layer 2 are not updated even though they should. In this situation we define these cache nodes as out of error tolerance and hence inconsistent. Nevertheless, the error will be corrected with the next update cycle as shown for Phase V. This example shows the problems that may occur for monotonic sensor values when using the Forward If Updated policy. However, for natural measurements monotonic sensor values are not a common behaviour, so that the Forward If Updated policy produces good enough results. Nevertheless, if inconsistent states are not allowed, the Decision at Source Node and Weighted Forward If Updated policies represent alternative strategies.
In summary, we have shown in this section how cache coherence can be guaranteed by allowing an average error gradient in the network based on an adjustable error function. In the next section, we discuss how caches can be localized for given user queries.

6.2.3 Cache Localization

In DACS, caches can be localized using model-driven approximative queries [56]. The query issuer defines a quality demand that needs to be resolved by DACS autonomously. DACS estimates the distribution of the nodes in the network and derives the error distributions of the caches in the network. Based on the distributions DACS extrapolates the next

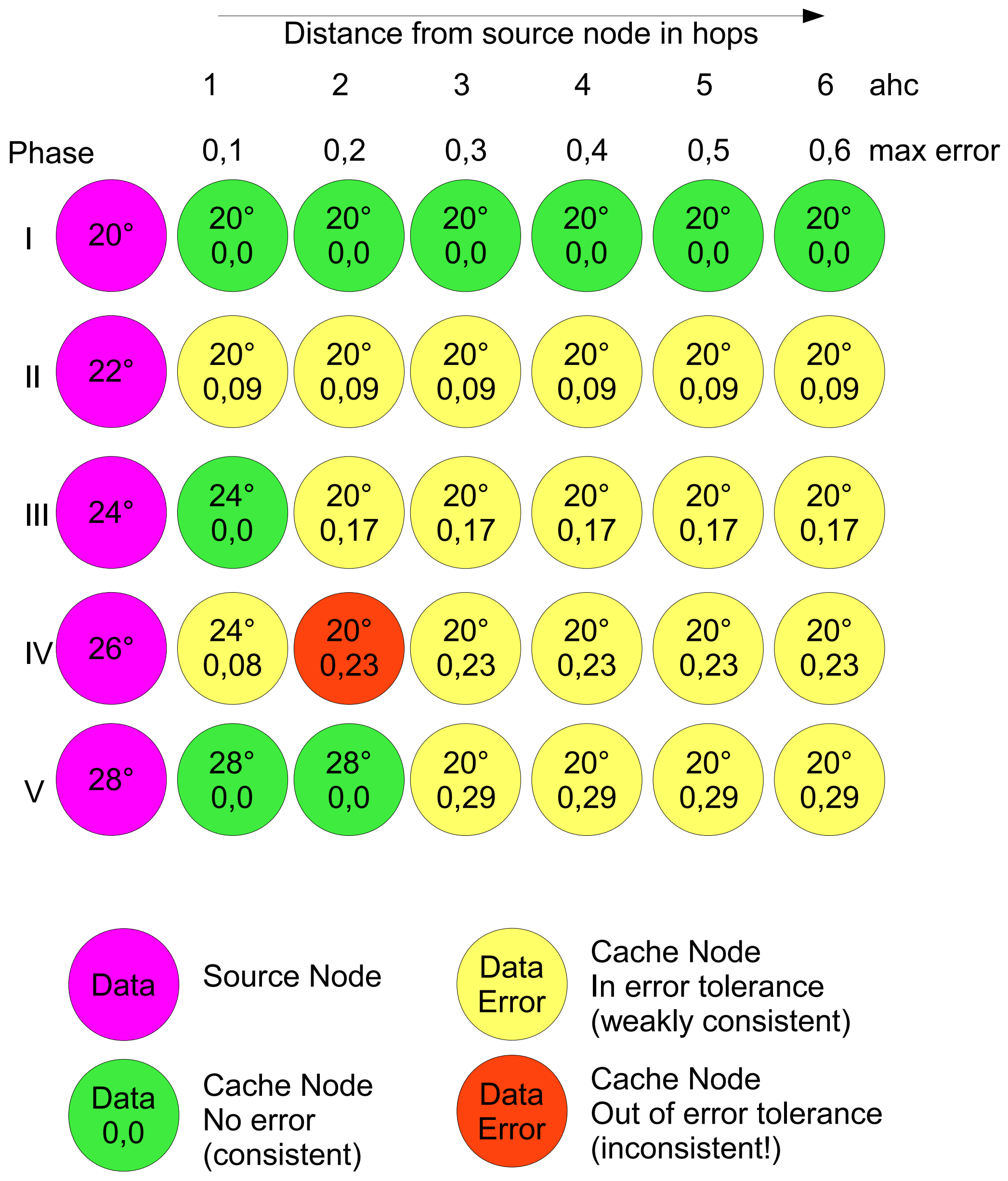

Distance from source node in hops
1 2 3 4 5 6 ahc
Phase
0,1 0,2 0,3 0,4 0,5 0,6 max error
I 20° | 20° 0,0 | 20° 0,0 | 20° 0,0 | 20° 0,0 | 20° 0,0 | 20° 0,0
II 22° | 20° 0,09 | 20° 0,09 | 20° 0,09 | 20° 0,09 | 20° 0,09 | 20° 0,09
III 24° | 24° 0,0 | 20° 0,17 | 20° 0,17 | 20° 0,17 | 20° 0,17 | 20° 0,17
IV 26° | 24° 0,08 | 20° 0,23 | 20° 0,23 | 20° 0,23 | 20° 0,23 | 20° 0,23
V 28° | 28° 0,0 | 28° 0,0 | 20° 0,29 | 20° 0,29 | 20° 0,29 | 20° 0,29
Data
Source Node
Data Error
Cache Node
In error tolerance
(weakly consistent)
Data 0,0
Cache Node
No error
(consistent)
Data Error
Cache Node
Out of error tolerance
(inconsistent!)

Figure 6.2: Forward If Updated Error Intolerance

cache layer that fulfils the quality requirements. This entire process is hidden to the query issuer which makes DACS an optimal solution for non expert sensor network users.
As described in the previous section, cache nodes are classified by their membership to cache layers, e.g. the distance d_{GC} between a gateway and the cache nodes. In DACS, the user defines an overall maximum average error requirement ϵ along his query, e.g. get all temperature values with an overall maximum average error of ϵ. DACS automatically retrieves a corresponding cache layer, e.g. the maximum hop for the query messages, so that the requirement can be satisfied.

We therefore define a function $C(\epsilon) \to c \in \mathbb{N}$ that looks up a corresponding cache layer c that fulfils ϵ. The resulting cache layer guarantees that the overall average error of the results does not exceed ϵ. Hereby, it is important that not only the maximum deviation of the cache layer is relevant to the overall average error but also the amount of nodes between the cache layer and the gateway that will send exact results. On the other side, DACS chooses the cache layer as close as possible to the gateway without violation of the requirement ϵ to optimally reduce the communication overhead.
The localization of the cache nodes depends on two factors:

1. the error function $\Upsilon(d_{SC})$

2. the distribution $F(X)$ of the nodes on the hop layers of the network starting from the gateway. The distance from the gateway to a node is defined by d_{GN}. The density function is denoted as $f(X)$. In Figure 6.3 we show a possible gaussian distribution which will be used throughout this section as an example and is, as we show later in this section, representative for many network scenarios.

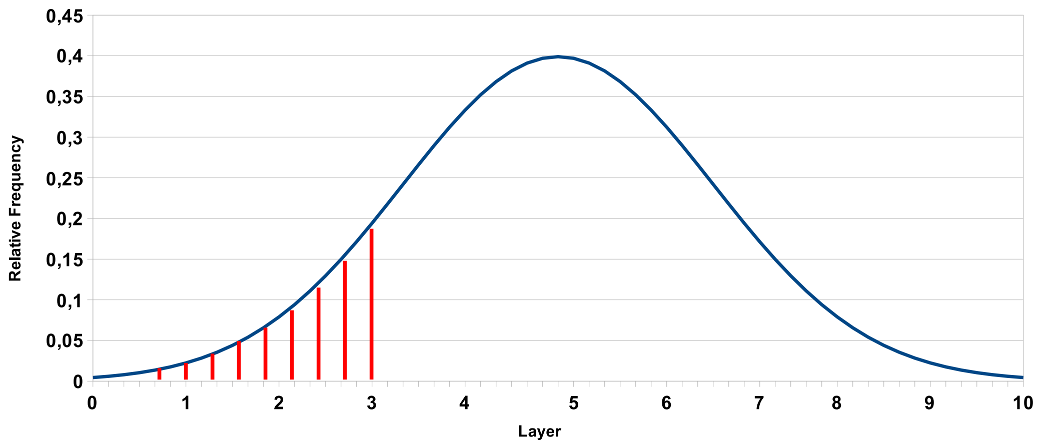


Figure 6.3: Network Node Distribution Density $f(x)$ on Hop Layer

Localization based on Node and Error Distributions

The function $C(\epsilon)$ can be derived inductively over the used cache layers as shown in the following cases:

CASE 1: Cache Layer $C = 0$

In this initial case, we assume that the gateway acts as a cache. By using the approximative update policy the maximum error of the cached value of a node n with distance $x = d_{SC} = d_{GN}$ is defined by $e_x = \Upsilon(d_{SC}) = \Upsilon(d_{GN})$. In example, a node that is 5 hops away from the gateway will be cached by allowing a maximum deviation of $\Upsilon(5)$.

Accordingly, by reviewing all nodes in the network we can derive an error distribution for a given cache layer C denoted as $G(C, e_x)$ (with density $g(C, e_x)$) that determines the amount of nodes that are cached by the cache layer C allowing a maximum error e_x. For the present case (C=0) the error distribution is now directly determined by the given node distribution density $f(X)$:

$$g(0, e_x) = f(\Upsilon^{-1}(e_x)) \tag{6.1}$$

In detail, the amount of nodes that are cached with a maximum error of e_x is the amount of nodes that are on a layer with distance d_{GN} and for that the equation $\Upsilon(d_{GN}) = e_x$ is true. We can retrieve this layer by resolving the function $\Upsilon^{-1}(e_x)$ and get the amount of nodes on this layer from the density function $f(\Upsilon^{-1}(e_x))$.
As shown in Figure 6.4, this theoretical derivation means that for the initial case ($C = 0$) we can directly determine the amount of nodes that are cached with a certain error e_x from the general node distribution.

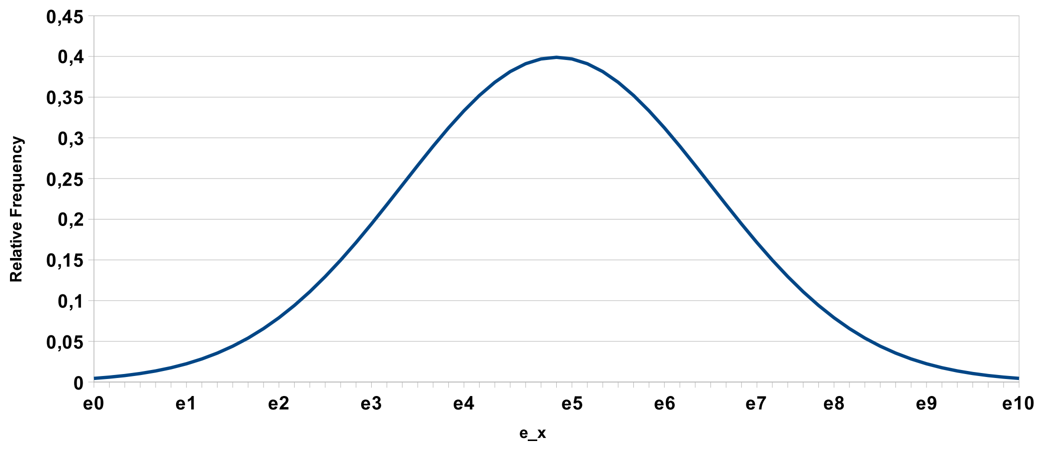


Figure 6.4: Network Error Distribution for Cache Layer 0: $g(0, e_x)$

The estimation value of the derived error distribution for cache layer 0 is the worst case overall average error of DACS. As denoted previously, we now have to review the case that we retrieve results from cache layer deeper in the network whereby nodes between the gateway and the cache will answer with exact results without error.

CASE 2: Cache Layer $C > 0$

We continue the inductive derivation of $C(\epsilon)$ by reviewing the usage of caches deeper in the network ($C > 0$). The overall average error of a query result can again be determined by the expectancy value of an error distribution $g(C, e_x)$. Nevertheless, we have to

take care of the nodes that are positioned between the cache layer C and the gateway, as they will send exact results.
We give an example in Figure 6.3. By using cache layer $C = 3$, all nodes on previous layers will answer with exact results. These nodes are marked by the red area and the number of these nodes is directly determined by $F(3)$.
In general, by using cache layer C, $F(C)$ nodes send exact results ($e_x = 0$) and hence we can derive

$$g(C, 0) = F(C) \tag{6.2}$$

In the following, we need to determine the error distribution of the rest of the nodes with error $e_x > 0$ that are actually cached by the cache layer C, e.g. the nodes from layer $C + 1$. As shown previously, the amount of nodes on layer $C + 1$ is determined by the node distribution $f(C+1)$. Accordingly, because these nodes are only one hop away from the cache layer the maximum error e_1 of the cached values is $\Upsilon(1)$. Based on the node distribution, we then retrieve the density of error e_1 as

$$g(C, e_1) = f(\Upsilon^{-1}(e_1) + C) \tag{6.3}$$

By reviewing all layers of cached nodes, we retrieve the general error distribution for a used cache layer C ($C > 0$):

$$g(C, e_x) = \begin{cases} F(C) & \text{, for } e_x = 0 \\ f(\Upsilon^{-1}(e_x) + C) & \text{, for } e_x > 0 \end{cases} \tag{6.4}$$

In other words, we retrieve the actual error distribution for a cache layer C ($C > 1$) by left shifting the error distribution of Equation 6.1 by C. In example, we show the resulting error distribution for the usage of cache layer $C = 3$ in Figure 6.5.
Equation 6.1 and Equation 6.4 now form the general error distribution of DACS for variable cache layer C:

$$g(C, e_x) = \begin{cases} f(\Upsilon^{-1}(e_x)) & \text{, for } C = 0 \\ F(C) & \text{, for } e_x = 0 \wedge C > 0 \\ f(\Upsilon^{-1}(e_x) + C) & \text{, for } e_x > 0 \wedge C > 0 \end{cases} \tag{6.5}$$

As denoted previously, we can now retrieve the overall average error by calculating the expectancy value E of the error distribution: $E(g(C, e_x))$.

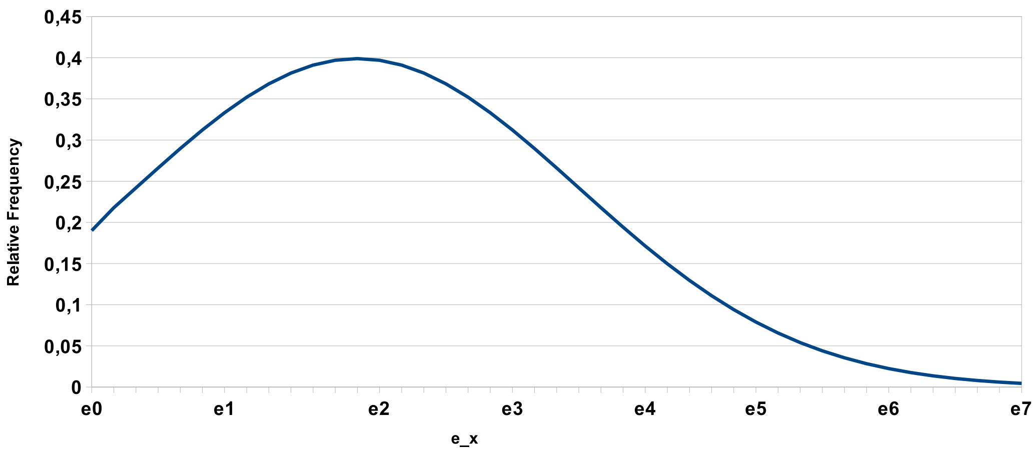


Figure 6.5: Error Distribution in Network for Cache Layer 3

Finally, $C(\epsilon)$ is defined as

$$C(\epsilon) = min(C|(\epsilon - E(g(C, e_x))) \; ; \; \forall \; C : E(g(C, e_x)) > \epsilon) \tag{6.6}$$

This function retrieves the cache layer C whereby the overall average error of this layer $(E(g(C, e_x)))$ is as close as possible to the error requirement ϵ without violating it. In DACS the function $C(\epsilon)$ is solved using interval-valued approximation.

Remarks on the Distribution of Nodes

In the derivation of $C(\epsilon)$ we assume the knowledge of the general node distribution $F(X)$. This distribution of nodes can be known for dedicated deployments, e.g. square deployments with equidistant nodes. However, for general deployments, e.g. deployments out of air, the exact position of nodes and their relative position to each other can not be fully verified. For this purpose, we have investigated the node distribution of randomly distributed networks to find distribution classes that can be used in DACS. We therefore randomly placed nodes of large scale networks consisting of n nodes (n>1000) and issued an analyzation query from a gateway to retrieve the amount of nodes on each hop layer.
As a result, we retrieved that randomly deployed networks tend to be gaussian distributed. In Figure 6.6, we show a histogram of the distribution of randomly placed nodes based on an average of 100 evaluations for 100 nodes. The red curve denotes the density function of the gaussian distribution with a mean / median of 17 and a variance of 8. The gaussian distribution was verified running statistical verifications, e.g. the Kolmogorow-Smirnow Test [25]. The previous described localization strategy in general is independent of the actually used distribution. However, based on these statistical tests, a gaussian distribution can be assumed as a general case. Only an estimation of the network diameter has to be done before using DACS.

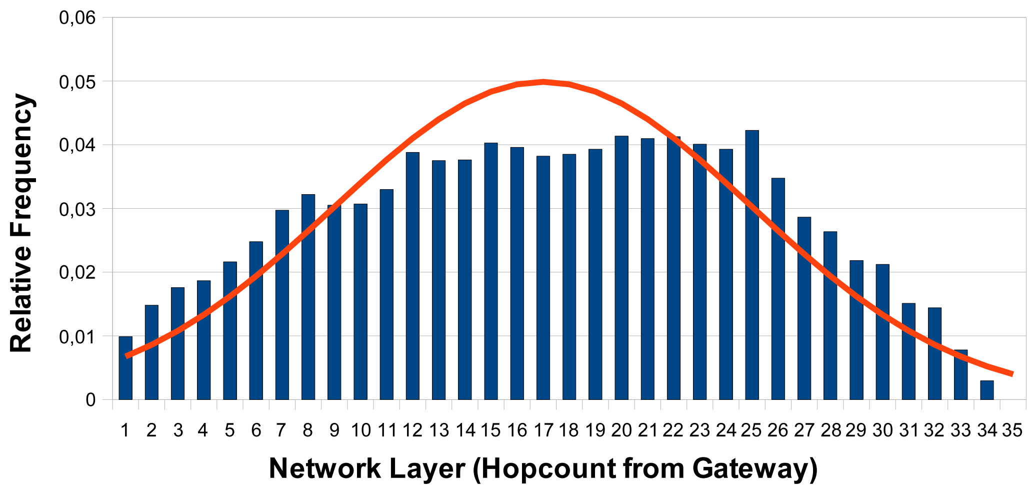


Figure 6.6: Statistical Node Distribution for uniform randomized Deployments

6.3 DACS Implementation

In this section, we discuss the DACS implementation. The actual implementation was done for iSense-based sensor nodes. Nevertheless, we present a general implementation architecture and communication protocol in the following sections. Hence, DACS can easily be adapted on other sensor node platforms.

6.3.1 Implementation Architecture

In Figure 6.7, we show the component diagram of the DACS architecture. The diagram is hereby simplified for better readability. The DACS implementation basically consists of 5 components:

1. *Communication*: A component for analyzing packets that are received and to create result packets and sending them into the network.
2. *Query Engine*: The query engine evaluates queries and cache update messages.
3. *Cache Update Validation*: This component checks if a cache needs to be updated and further decides if an update needs to be forwarded.
4. *Sensing*: The sensing unit collects the sampled sensor data, provides the current data and stores it for historic archiving.
5. *Cache Management*: This component is the central implementation of the data cache including cache lookup routines and a cache write interface.

Like any other sensor network application, DACS is a highly parallel application, e.g. the single components are running autonomously in parallel time. However, for better

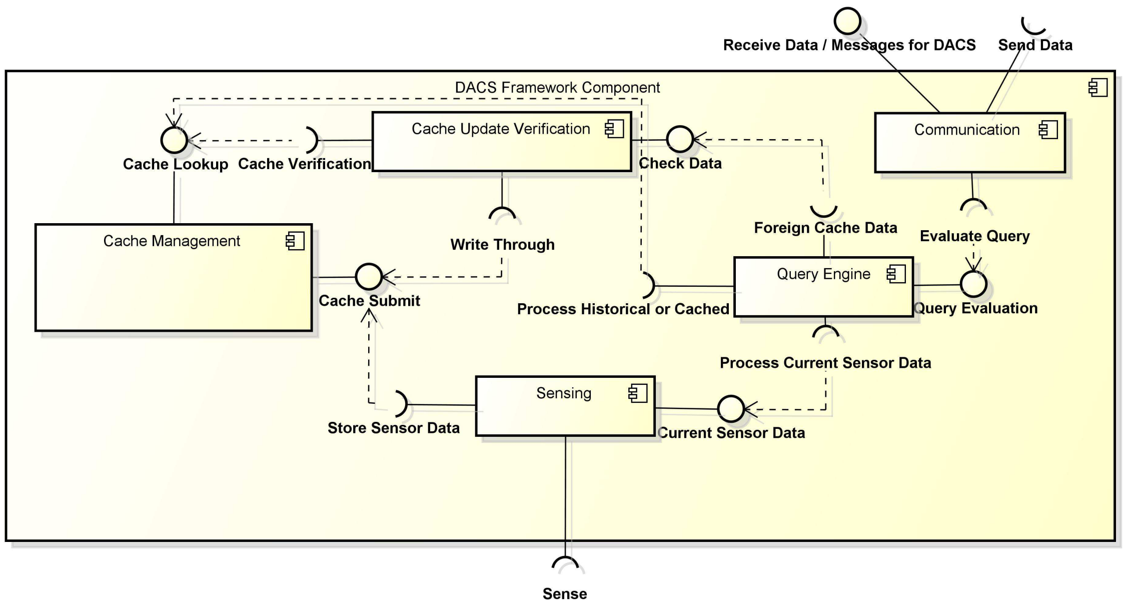


Figure 6.7: DACS Architecture Component Diagram

understanding we discuss the general program flow in a pseudo serialized way in the following. The communication component is responsible for processing query messages, result messages and cache update messages that have been classified for DACS by the central receive function of the sensor node. If one of these message types arrives, it is transformed in an internal query format and forwarded to the query engine. Any query messages that are forwarded by the communication component are clearly targeted for the current sensor node. E.g. the sensor node is responsible for answering the query with cached data representative for other nodes due to its query hop count as proposed by the localization algorithm. The query engine is now responsible for evaluating the query. If the query is targeted for the sensor node itself, the query engine will request either current sensor data from the sensor unit or historical data from the data cache by using the direct required interface. If the query targets another sensor node that is cached by the current sensor node, the query engine also looks the data directly up using the cache management component. In both scenarios, the query engine component will create the query result and send it back to the data sink using the communication component. If the query contains a cache update, the query engine verifies it by forwarding it to the cache update verification component. This component processes the cache coherence algorithm as proposed in Section 6.2.2. If the forward policies demand to forward the query update, the cache update verification component will notify the query engine which forwards the update using the communication unit. As denoted previously, this is just a pseudo serialized program flow. The sensing component and cache management component work in parallel as they sample data continuously.

After discussing the general program flow and architecture of DACS, we will introduce the message protocol and communication strategies, e.g. message dissemination, in the next section

6.3.2 Message Delivery

As proposed previously, DACS desists from using a fixed network topology. According to Nath et al. [170] the ring oriented communication is the most flexible and robust form of data transmission in wireless sensor networks. It uses broadcast communication but avoids endless flooding by directing messages based on rings around the data sink, e.g. messages are attracted by the data sink like a magnet attracts metal. A message hence is routed over several ways analogously which avoids communication bottlenecks like known from fixed topology approaches like TAG [155]. In the following paragraph, we will discuss this in more detail. In the subsequent paragraph we will then introduce the DACS packet structure that is used by the communication component in Figure 6.7 to classify the type of message.

Message Flow

As denoted, DACS uses ring oriented communication according to Synopsis Diffusion by Nath et al. [170]. Thereby the following assumptions are made:

- All sensor nodes use broadcast message delivery.
- Sensor nodes have no further information about their position and their neighbours.
- The only information that is known is the query hop count which signals the estimated distance to the gateway.
- Messages include header information about the progress of the dissemination (for the full header see next paragraph):
  - The query hopcount **qhc** defines the distance between the receiver and the query issuer (gateway). It is increased as long the active query is forwarded for every hop.
  - The answer hopcount **ahc** defines the distance between the data source (result / answer source) and the receiver. The answer hopcount will be increased as long as the answer of a query is forwarded to the data sink (gateway).

The ring oriented communication assumes a full broadcast communication that can result in endless flooding. As denoted previously, to avoid this problem the message number is reduced by using forward rules that represent attraction phenomena. We extended this rule by allowing a message with a unique id to be only forwarded once by a sensor node. We summarize the message forwarding strategy as follows:

1. **Forward only once**
   A sensor node can identify a message uniquely by the packet header. To avoid an endless circulation of messages, each sensor node only forwards a query once. If a query appears for the first time, the query id is stored and marked as forwarded. For every additional time the same query is received as an active unanswered query it is dismissed. Result messages are processed in similar way. To uniquely identify a result message a combination of query id and answer id has to be reviewed.

2. **Forward result messages in attraction of the gateway**
   The forwarding rule, that is an extended form of the Synopsis Diffusion forward policy, demands that an answer message is only forwarded if it is received from an adjacent sensor node with a query hopcount that is one hop higher than the forwarding node. In that way, the answer message is attracted by the gateway using a decreasing query hopcount as the attraction.

We give three examples for the second forward policy in Figure 6.8. K1 and K2 have the same query hopcount after receiving the active query with the same query hopcount and are defined to be on the same ring. K3 is one ring nearer to the gateway and K5 is one ring further away from the gateway. We now review the case that the query has been disseminated and result (answer) messages are sent towards the gateway. An answer message is only allowed to be forwarded if it is received from a ring with higher query hopcount and hence one hop further away from the gateway. We define nhc as the query hopcount of the node that needs to decide if to forward. The forward decision is now based on whether the equation qhc - ahc = nhc is true. In our example K1 is sending an answer

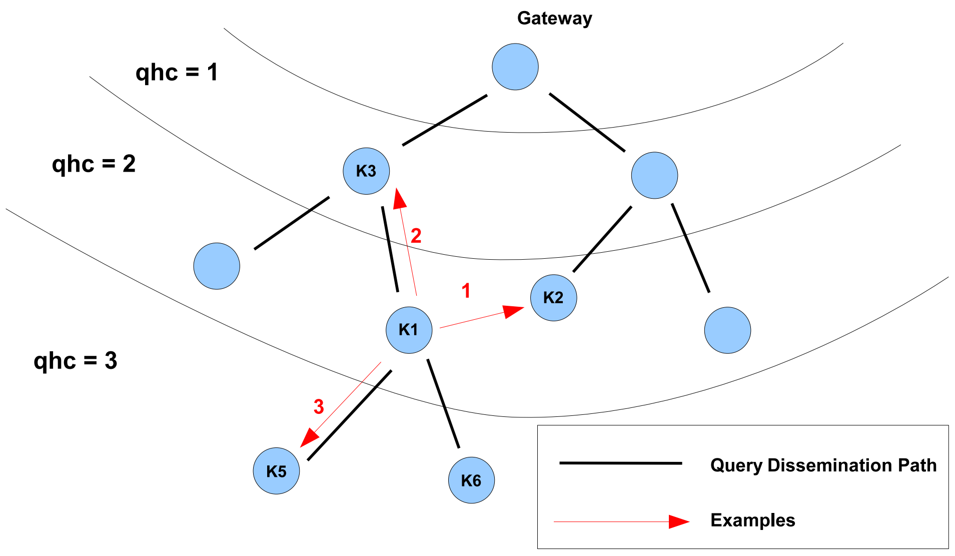


Figure 6.8: DACS Message Forwarding Policy

message via broadcast. K2, K3 and K5 receive this message. K2 is on the same ring than

K1. Hence, K2 will not forward since 2-1 != 2. The message will be dismissed. The second case occurs for K3. This sensor node is one ring nearer to the gateway and hence responsible for forwarding the message. The equation is satisfied (2-1=1) and the message will be forwarded. The last case shows how the problem of endless flooding in the wrong direction is avoided. As K5 is deeper in the network it is not desirable to let the node forward a packet. The forwarding policy takes care of it, as the equation is not satisfied (2-1!=3) and the message will be dismissed.

DACS Packet Structure

In the previous paragraph, we defined the general ring oriented message forward policy. Given this message forward policy, a sensor node accepts and forwards or dismisses a message based on the query id, query hopcount and the sensor node's own hopcount. Finally, we summarize the entire DACS packet header in Table 6.1.

Position	Field Name	Size	Description
0 - 1	qid	2 Byte (uint16)	Queryid of the current packet
2 - 3	qhc	2 Byte (uint16)	Hopcount of the active query
4 - 5	aid	2 Byte (uint16)	Id of the origin node of the query result (AnswerId)
6 - 7	ahc	2 Byte (uint16)	Hopcount of the query result (AnswerHopcount)
8	options	1 Byte (uint8)	Additional packet information determing the type of packet(Init Packet, Result Packet, Update Packet, Cache Query)
9	length	1 Byte (uint8)	Length of the data payload
10 - ...	Payload	Variable	E.g. result data, sensor data

Table 6.1: DACS Packet Header Structure

As denoted in the previous paragraph, the queryid, query hopcount, answer id and answer hopcount are needed to decide if a message needs to be forwarded or not. The optional byte is used for additional packet information. This includes a definition of the packet type and can be extended for quality of service (QoS) purposes. The payload includes the actual sensor data for a query result.

6.3.3 Localization Algorithm

As shown in Section 6.2.3 the localization of the optimal cache layer depends on four factors:

- The error function which is used for forwarding cache update messages as shown in Section 6.2.2.

- The node distribution in the network.
- The maximum deviation in the global result the user is willing to accept.
- The maximum query hop distance (network diameter).

A technical and theoretical calculation of the optimal cache layer has been introduced in Section 6.2.3. As also proposed in Section 6.2.3, the distribution of nodes in general network, e.g. with randomly placed nodes and randomly chosen gateway, can be estimated as the gaussian distribution. The implemented localization of the optimal cache layer hence is shown for the gaussian distribution. Since all of the factors are known from outside the network, the optimal cache layer can be calculated on user side. This process saves energy for processing the estimation and memory that is needed for the distribution approximation represented by tables. We therefore implemented a tool in Java that estimates the optimal cache layer for any given error function, network diameter and maximum deviation demand as parameters. In Listing 6.1, we show the program code of the estimation function. The class *NormalDistributionImpl* is thereby given by the org.apache.commons.math framework [224].

Listing 6.1: Estimation of the Optimal Cache Layer in Java

```java
private static int getOptimalCacheLevel(NormalDistributionImpl
    distribution, double failure,int maxLevel){
    double[] results = new double[maxLevel+1];
    for(int j = 0;j<results.length;j++){
      results[j] = 0.0;
    }
    // Calculate estimated error for every cache layer
    for (int c=0;c<=maxLevel;c++){
      // all layers deeper equal the cache layer
      //(that include an error)
      for(int i=1;i<=(maxLevel-c);i++){
        results[c] = results[c] +
              (distribution.density(new Double(c+i))*errorfunction(i)
                  );
    } }
    // Choose the avg error that is the closest to the deviation
        demand
    int optcachelayer = maxLevel;
    for(int k=0;k<results.length;k++){
      if((results[k] <= failure) && (results[k] > results[
          optcachelayer])){
        optcachelayer = k;
    } }
    return optcachelayer;
}
```

In detail, the function first iterates through all possible scenarios, e.g. choices of cache layers, and calculates the average error according to the cache coherence protocol in Section 6.2.2. As denoted previously, the error only depends on the nodes that are equal or deeper the cache layer. After checking all cache layers and getting all errors, we find the optimal cache layer by comparing the error to the deviation (and hence error) demand of the user. The algorithm proposes a brute force technique to find the optimal cache layer for better understanding. Nevertheless, it can be extended by using computation over nested intervals. After discussing the implementation aspects and the architecture of DACS, we show how DACS performs in simulation and real deployments in the next section.

6.4 Evaluation

In this section, we give an extended overview on evaluation results of running DACS in wireless sensor networks. Hereby, DACS has been evaluated in real sensor node deployments and simulations to test the scalability for very large deployments. The measurement data is based on real temperature measurements in Friedberg, Germany. As proposed throughout this work, the intention of DACS is to save energy on the communication path. Hereby, it is not only important that DACS actually reduces the communication overhead but also that the error requirement ϵ of a given query is never violated. Therefore, the evaluation covers the following most important aspects:

1. *Communication Efficiency*: The impact on the communication demand of using various error functions is shown in Section 6.4.1.

2. *Query per Update Trade-off*: The evaluation in Section 6.4.2 covers aspects on when to use DACS in relation to the ratio between expected updates and queries.

3. *Validity and Robustness*: The error requirement of a given query has to always be guaranteed. Therefore the deviation gradiation on the update layer path that is defined by the used error function needs to be adhered. An evaluation for this aspect is given in Section 6.4.3.

As sensor node hardware we use Pacemate nodes [153], based on a Philips LPC 2136 Processor, and iSense core modules, based on a Jennic 32bit RISC Controller [45]. The available RAM was 96kByte shared for program and data (heap memory was $\approx$15kByte, program memory was $\approx$81kByte). We hereby point out that DACS is fully applicable on real sensor nodes and has been tested in an indoor application scenario as shown in Figure 6.9 whereby each node sends measurements continuously and one node acts as a gateway.

6.4.1 Energy and Communication Efficiency

We first test the communication efficiency of DACS based on two quarter temperature measurements and different error functions. The results are shown in Figure 6.10. Hereby,

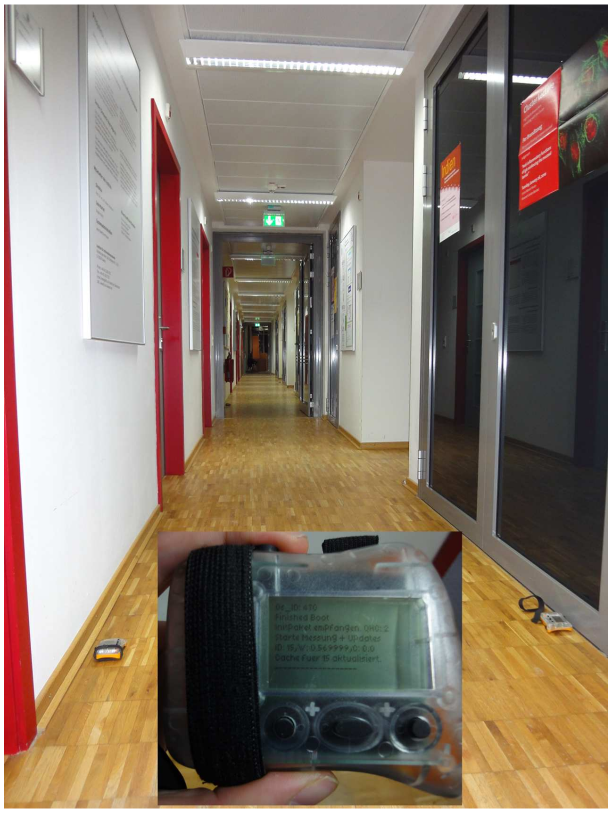


Figure 6.9: DACS Pacemate Indoor Deployment

the x-axis shows the chosen linear error function for the coherence protocol and the y-axis denotes the communication demand in update messages. As a result, linearly increasing the error function significantly reduces the communication demand. Both measurements show a logarithmic decrease. Using DACS in the network therefore significantly reduces the update demand if the user can accept a higher deviation in the cache results.

6.4.2 Query per Update Trade-off

In the next evaluation, the usability of DACS in relation to the expected amount of queries was tested. As denoted previously, the usability of a caching scheme depends on the amount of unique queries that are sent in the network. Generally, caching becomes interesting if a high amount of independent queries is estimated and the update rate is lower. In this evaluation, we show how this trade-off can be determined for DACS. Thereby, we have to compare the cache layers that are actually used. In Figure 6.11, we show the trade-off results for the message demand for the temperature evaluation scenario based on a network of 100 nodes with a diameter of 6 hops over 91 update cycles (one quarter). The x-axis hereby denotes the numer of queries that occur during the 91 update cycles (query per update ratio). The y-axis denotes the overall communication demand in messages. The message demand for direct querying acts as a reference, where all nodes need to be reached

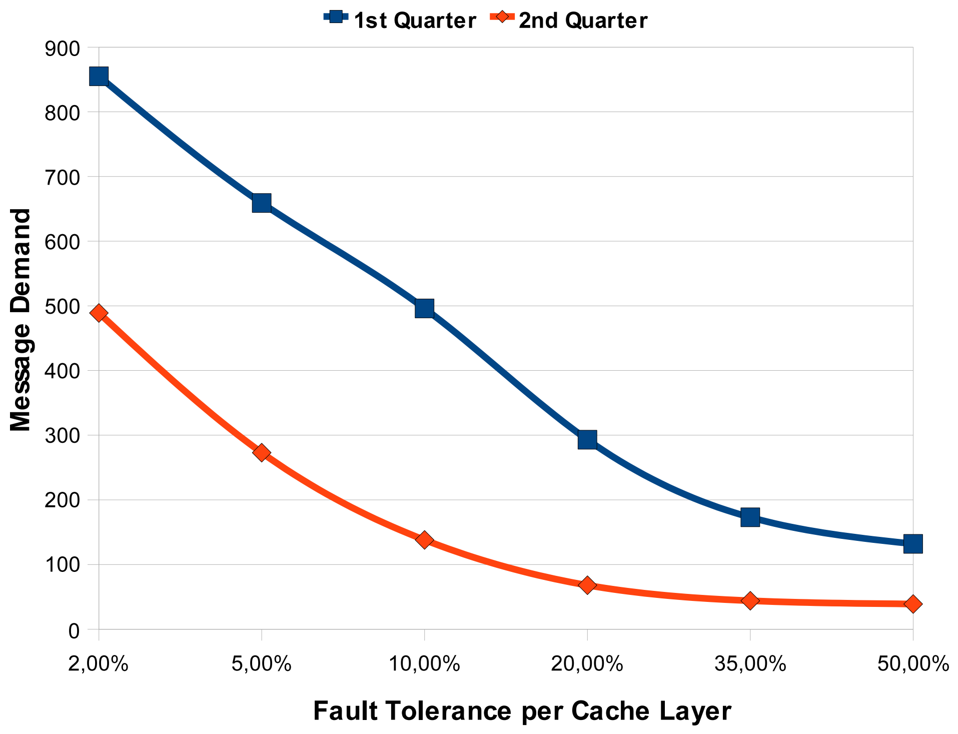


Figure 6.10: Message Demand for varying Error Function Update Policies

without using caches and hence the results need to be forwarded through the entire network. The different curves show the usage of cache layer one to six. The higher the cache layer the deeper it is in the network. As a result, the intersection between the direct querying curve and the cache layer curve determines the point of inflection at which caching reduces the communication demand. We show the inflection points of this scenario in Table 6.2. For caches closer to the gateway the trade-off is significantly small, e.g. for cache layer 1 there are 0.14 queries per update, which shows the benefits of approximative data caching in wireless sensor networks.

Cache Layer	**Ratio Query per Update**
1	0,14
2	0,16
3	0,23
4	0,38
5	0,98
6	36,6

Table 6.2: Point of Efficiency: Query per Update

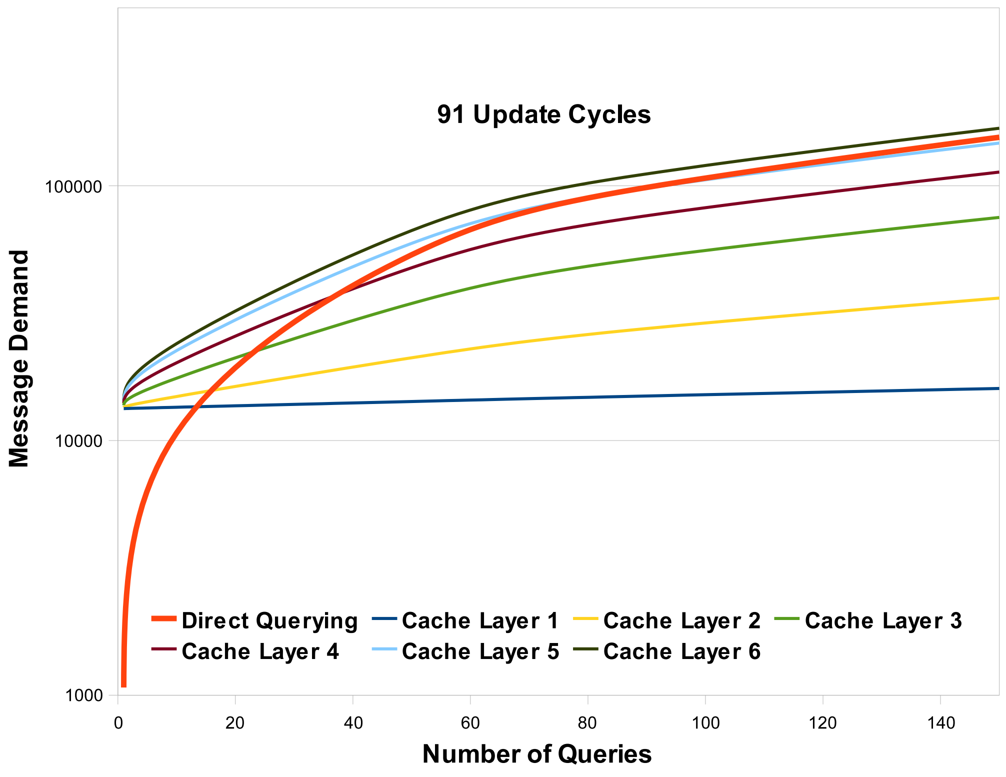


Figure 6.11: Update per Query Trade-Off

6.4.3 Validity and Robustness

In Section 6.2.2, we introduced the approximative cache coherence protocol. We defined an approximative update policy that is also used for localizing an optimal cache layer as proposed in Section 6.2.3. In this evaluation we show that by using DACS the update policy is never violated. Again, this evaluation is based on the temperature scenario in a network with diameter 10, non-robust communication and randomly placed nodes. A linear increasing error function was used (10% per layer) for cache updates. Figure 6.12 shows the actual cache errors based on this linear deviation update policy for the two quarter measurements. Hereby, because of the update policy a maximum deviation of 10% per hop distance from the actual data source is allowed. As a result, the DACS update policy adheres all failure guarantees making it a stable and reliable caching scheme. For the 2nd quarter we tested the *Forward if Updated* forward policy. Hereby, it can be seen that because of non-predictable, monotonic sensor data the error gradiation tends to be more non-linear. Nevertheless, the error reference is never violated which shows that the non stable forward policy can also be used in practice.
To verify the results, we show the corresponding experimental results for using a 5% and a 20% error function in Figure 6.13 whereby an average of both quarters is shown.

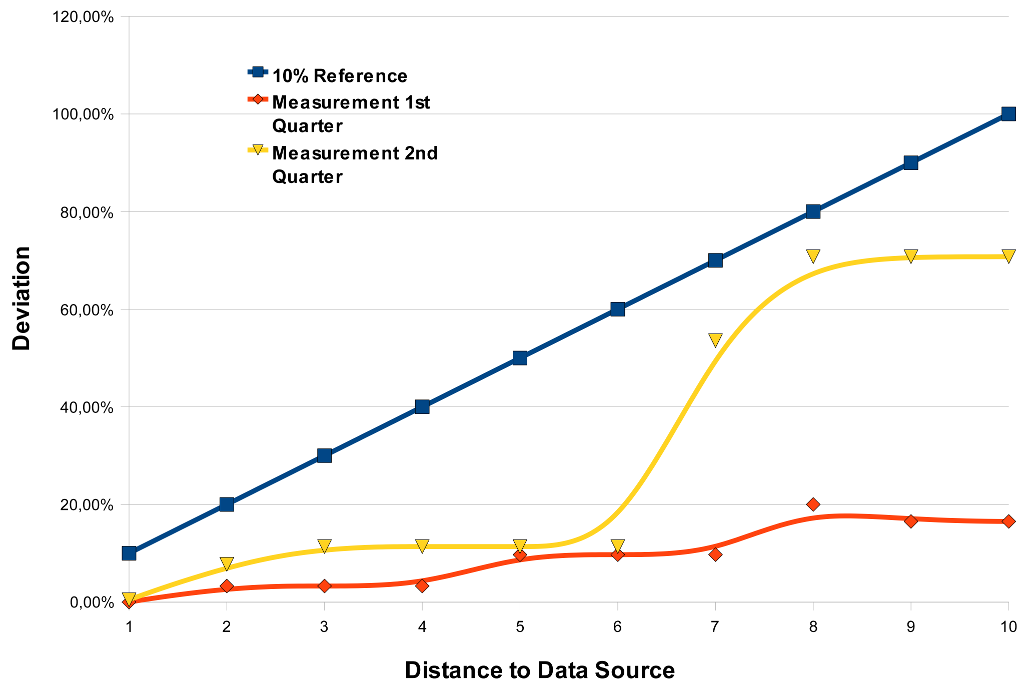


Figure 6.12: Validity of the Update Policy for a 10% per Hop Error Function

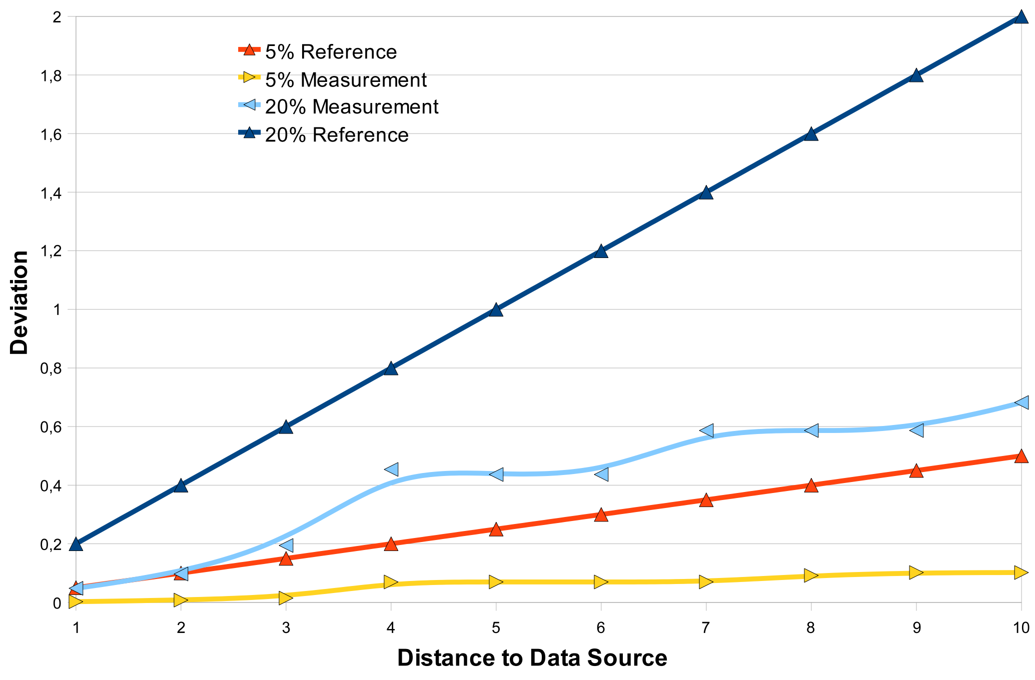


Figure 6.13: Validity of the Update Policy for a 5% / 20% per Hop Error Function

Again, the update policy can be guaranteed for both error functions. Hence, DACS provides a flexible stable update management with stable error gradiation which ensures the correctness of the cache localization process.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this dissertation, we presented the $XOBE_{SensorNetworks}$ framework for efficiently integrating XML data management and query processing in the wireless sensor network development and lifetime cycle. Integrating XML data management into wireless sensor networks is a means towards supporting extended heterogeneity on the application layer and adapting the service oriented paradigm to sensor network application engineering using standardized SOA techniques.

The $XOBE_{SensorNetworks}$ framework provides transparent XML usage during the sensor network application development by extending the Embedded C programming language with XML constructs. This concept enables a high-level and transparent XML usage for sensor network developers. Moreover, XML data can be statically type checked to ensure more stable programs and to avoid costly maintenance of deployed networks.
The most important limitations of XML usage in wireless sensor networks are the scarce resources of the sensor nodes. Strict limitations in energy supply and memory configuration require new techniques in handling XML data during runtime in the network. This can be a reason why XML data management has not been reviewed in the early sensor network data management approaches.

To bridge the limitation gap, we introduced the XML template compression scheme that is suitable for using XML in sensor networks with strict limitations of energy and memory. $XOBE_{SensorNetworks}$ includes two separate implementations of the XML template compression scheme. While the first implementation (XTO) relies on using a light weight object model to allow direct access to the represented XML data the second implementation (XTS) desists from using a dedicated object model and rather processes the encoding as a stream by using a special pushdown automaton on the nodes. Our evaluation of the concepts of this thesis covered a comparison of both implementations concerning the compression ratio and hence the efficiency of in-memory storage. As a result, we showed that both solutions have their application domain. While the XTO approach is used in application scenarios where fast access to the document is more important than highest compression the XTS approach is applicable for sensor networks with strict memory limitations. Nevertheless, both approaches outperform other possible XML data binding solutions and are currently the only concepts that enable dynamic XML data management in wireless sensor networks.

Wireless sensor networks are mainly deployed for data acquisition. Data management solutions therefore need to provide a data centric query evaluation layer. Accordingly, we showed that both compression implementations are feasible to evaluate XPath queries during runtime on the sensor nodes. While the XTO approach uses traditional tree navigation techniques, the XTS approach uses an extended PDA including a new concept for storing query results in a compressed form. We evaluated both approaches using the widespread XPathMark. As a result, the XTO approach is more energy efficient by requiring less processing cycles. While the XTS approach requires extended processing using the PDA, it is the most memory efficient implementation by using the compressed BinType concept. Hence, these results again show the different application domains of both approaches.
Regarding the in-network processing of XML queries, we further discussed the dissemination of XML queries, the in-network aggregation of XML data and the optimized transmission of templates. We pointed out that XPath queries can be syntactically extended to enable continuous query processing.

In the last part of this thesis, we discussed the optimization of dynamic data acquisition using data caches. In Chapter 6, we therefore proposed the dynamic approximative caching scheme DACS for wireless sensor networks to optimize model driven query evaluation. An approximative update policy has been introduced to support weak cache coherence. Based on a deviation tolerance a query is issued to the network and the corresponding caches are used adaptively. Evaluations have shown that this concept performs significantly better than traditional query evaluation when a minimal deviation in the query results can be accepted. The lifetime of networks using DACS can therefore be significantly extended.

In summary, the concepts of this thesis have significant impact on handling complex data formats in wireless sensor networks. The support of XML data management in an energy and memory efficient way opens up many benefits like heterogeneity and the integration in the WWW using standardized XML based protocols. $XOBE_{SensorNetworks}$ encapsulates all concepts to provide developers and users a transparent framework. We believe that this dissertation is a great contribution to inspirit and realize the idea of Smart Dust.

7.2 Future Work

Although the concepts of this work show promising results and had significant impact in the sensor network research community, the XML integration process is still in early stages and can be optimized and extended in various work. In the last chapters, we already pointed out interesting future topics to further optimize the XML data management in wireless sensor networks. We finally summarize and give more detail on these areas of future work.

We pointed out that the XTS approach is currently reviewed as the state-of-the-art XML data management solution. Beside optimizing the processing of XTSs, we imagine a dynamic co-processor for handling XTSs on the sensor nodes. Using integrated circuitry or FPGAs to realize the XTS PDA will optimize the runtime and energy efficiency significantly. Nevertheless, this requires profound technical knowledge and further results in the miniaturization process of electronic circuitry to fulfil the size requirements of Smart Dust. The presented XML support is the vehicle for enabling standardized protocols like SOAP. For future work, we therefore consider the implementation of a SOAP engine for the XTS approach as a next step on adapting the service oriented paradigm for sensor network engineering. We already extended the XTS PDA to be ready for processing and evaluating SOAP headers and bodies. Moreover, we see great potential for optimizing the compression by extending the XTS encoding language to support SOAP based encoding types. What parts of SOAP messages can be represented as XML templates is still an open question. We imagine that great parts of SOAP are not necessary in sensor networks and can therefore been left out for further optimization or be cached as proposed in our introduction into template caching.

Beside the actual management of XML data in wireless sensor networks, we see many areas of optimization and future work in our concepts for evaluating XML queries on compressed XML data. Firstly, the support of other languages like XQuery and XSLT can be realized. Secondly, the support of continuous XML queries should be extended. We already discussed the BCSQ approach as one option. Handling complex data in the network also opens up many new opportunities like heterogeneous in-network aggregation and XML index structures.
Additionally, we see many benefits for related approaches. The *Xenia* approach has been introduced as one of the most efficient compression techniques for XML data [244]. However, the absence of direct query evaluation on fully compressed documents restricts the usage of Xenia [244] in wireless sensor networks. With the introduction of the Bintype concept and the stream oriented PDA XPath evaluation strategy in this dissertation, we showed possible methods that may be adapted to enhance the dynamic processability of Xenia encoded documents. We currently have implemented a prototype that proves this

possibility and are looking confidently into the future that XPath queries can be evaluated on Xenia encoded documents.

Finally, our integrated dynamic approximative caching scheme DACS can be optimized in the area of cache coherence. We already pointed out that other message forward policies can be reviewed and introduced to avoid temporary unstable cache conditions. We also point out that the concepts of DACS have significant impact on realizing XML index structures and index coherence. This is an important field that should be reviewed in the future.

Appendix A

List of Publications and Awards

All publications of the author of this work are listed here in chronological order. The work published in [95, 100, 102] hereby has been explicitly awarded in the conference proceedings. Beside the following publications, the author of this work has also contributed to [84, 85, 132, 133, 154, 171, 172, 198, 199, 200].

Title: Xobe Sensor Networks: Integrating XML in Sensor Network Programming
Authors: Nils Hoeller, Christoph Reinke, Sven Groppe and Volker Linnemann
Abstract: Communication in and with sensor networks often lacks of exchangeability. Furthermore handling communication data formats during sensor node programming is often complex and programming errors can result in unstable programs. In this poster we introduce the easy to use programming framework $XOBE_{SensorNetwork}$, which provides the direct use of XML in a sensor node programming language, while ensuring stable and space-, time- and energy-efficient programs handling XML data.
Published: in [96]

Title: Efficient XML Usage within Wireless Sensor Networks
Authors: Nils Hoeller and Christoph Reinke and Jana Neumann and Sven Groppe and Daniel Boeckmann and Volker Linnemann
Abstract: Integrating wireless sensor networks in heterogeneous net- works is a complex task. A reason is the absence of a standardized data exchange format that is supported in all participating sub networks. XML has evolved to the de facto standard data exchange format between heterogeneous net- works and systems. However, XML usage within sensor networks has not been introduced because of the limited hardware resources. In this paper, we introduce XML template objects making XML usage applicable within sensor networks. This new XML data binding technique provides significant high compression results while still allowing dynamic XML processing and XML navigation. This is a step towards more complex but exchangeable data management in sensor networks and the extension of the service-oriented paradigm to sensor network application engineering.

Published: in [98]

Title: *smartCQ*: Answering and Evaluating Bounded Continuous Search Queries within the WWW and Sensor Networks
Authors: Nils Hoeller, Christoph Reinke, Dirk Kukulenz and Volker Linnemann
Abstract: Continuous Queries (CQ) can be used to keep track of relevant information in the World Wide Web and Sensor Networks over a period of time. However, result sets may become unbounded and notifications can be delayed. A special form of CQ are Bounded Continuous Search Queries (BCSQ), where results are processed immediately and a bounding condition for the number of user notifications may be defined to limit the result set. Based on a theoretical background for answering BCSQ we present smartCQ, a web application for processing BCSQ and evaluating the query results. Based on a transparent search engine a user may execute and evaluate new strategies and methods for continuous search queries. Furthermore because of the positive results of BCSQ we encourage its use for querying Wireless Sensor Networks to reduce the communication demand and hence reduce the energy consumption to enhance the lifetime of a sensor network.
Published: in [97]

Title: Towards Energy Efficient XPath Evaluation in Wireless Sensor Networks
Authors: Nils Hoeller and Christoph Reinke and Jana Neumann and Sven Groppe and Christian Werner and Volker Linnemann
Abstract: Using XML as a standardized data exchange format in wireless sensor networks is a means to support more complex data management and heterogeneous networks. Moreover, XML is a key feature towards service-oriented sensor networks. Recent work has shown that XML can be compressed to meet the general hardware restrictions of sensor nodes while still supporting updates. In this work we outline the vision and benefits of XML usage in wireless sensor networks. We further present first evaluation results of an implemented XPath query engine, that is able to evaluate a large set of XPath queries dynamically on XML using sensor nodes.
Published: in [101]

Title: XML Data Management and XPath Evaluation in Wireless Sensor Networks
Authors: Nils Hoeller and Christoph Reinke and Jana Neumann and Sven Groppe and Christian Werner and Volker Linnemann
Abstract: XML is the defacto standard for data exchange applications like those in the WWW. However, due to the limited hardware resources, wireless sensor networks abstain from using verbose data formats like XML. Nevertheless, XML as a standardized data exchange format in wireless sensor networks is a means to support more complex data management and heterogeneous networks. Moreover, XML is a key feature towards service-oriented sensor networks that exchange structured information by using SOAP. Recent work has shown that XML can be compressed to meet the general hardware restrictions of sensor nodes while still supporting updates. In this work we outline the vision and benefits of XML usage in wireless sensor networks, show how to evaluate XML queries in wireless sensor networks and how query results can be compressed to lower the

comunication overhead. We therefore present an XPath engine on updateable compressed XML data for sensor nodes and an experimental evaluation showing that the performance of our XPath engine fulfills the requirements of today's applications even on sensor nodes.
Published: in [102]
Note: *This paper received the MoMM 2009 Best Paper Award*

Title: Dynamic Approximative Data Caching in Wireless Sensor Networks
Authors: Nils Hoeller
Abstract: Communication in Wireless Sensor Networks generally is the most energy consuming task. Retrieving query results from deep within the sensor network therefore consumes a lot of energy and hence shortens the network's lifetime.
In this work optimizations for processing queries by using adaptive caching structures are discussed. Results can be retrieved from caches that are placed nearer to the query source. As a result the communication demand is reduced and hence energy is saved by using the cached results. To verify cache coherence in networks with non-reliable communication channels, an approximative update policy is presented. A degree of result quality can be defined for a query to find the adequate cache adaptively.
Published: in [95]
Note: *This paper received the MDM 2010 Best PHD Forum Paper Award*

Title: DACS: A Dynamic Approximative Data Caching in Wireless Sensor Networks
Authors: Nils Hoeller and Christoph Reinke and Jana Neumann and Sven Groppe and Florian Frischat and Volker Linnemann
Abstract: Saving energy in Wireless Sensor Networks is essential to extend the lifetime of in-field deployments. Previous research has shown that communication is generally the most energy consuming task and needs to be reduced in order to build resource-efficient long-term applications. The communication demand for retrieving query results from deep within the sensor network is typically high. As a result, frequent non-continuous data acquisition consumes a lot of energy and shortens the lifetime of the sensor network significantly. In this work we discuss optimizations for processing high amounts of unique queries by using a dynamic adaptive caching scheme: **DACS**. In DACS query results can be retrieved from caches that are placed nearer to the query source instead of sending queries deep into the network. The communication demand can be significantly reduced and the entire network lifetime is extended. To verify cache coherence in sensor networks with non-reliable communication channels, an approximative update policy is used. To localize the adequate cache adaptively, model-driven queries including a degree of demanded result quality can be defined. The entire logic is thereby processed by DACS and hidden to the user. The significant energy conservation is proven in evaluations that include real sensor node deployments.
Published: in [99]

Title: Stream-based XML Template Compression for Wireless Sensor Network Data Management

Authors: Nils Hoeller and Christoph Reinke and Jana Neumann and Sven Groppe and Martin Lipphardt and Björn Schütt and Volker Linnemann
Abstract: Using structured data formats like XML in wireless sensor networks to support exchangeability and heterogeneity on application level has become an important research topic in the area of large scale networked sensing systems. Besides, the usage of XML encourages the adaptation of service oriented programming techniques to simplify sensor network application engineering. While the sensor nodes still have significant resource limitations in terms of energy and memory capacity and computational power, recent data management approaches show positive results to bridge this resource gap. Nevertheless, further optimizations are needed to enhance the application range to support larger sets of data within the networks. In this work we present an optimization for a template object compression scheme that is based on a stream-oriented XML compression and supports dynamic data management and query evaluation on the compressed data. We hereby present a complete solution for XML compression, data processing and query evaluation that can be further embedded in the engineering process to support developers. The presented solutions are evaluated and result into significant improvements in comparison to previous approaches, when processing complex large scale XML documents.
Published: in [100]
Note: *This paper received the MUE 2010 Best Paper Award*

Title: Efficient XML Data and Query Integration in the Wireless Sensor Network Engineering Process
Authors: Nils Hoeller, Christoph Reinke, Jana Neumann, Sven Groppe, Christian Werner, Volker Linnemann
Structured Abstract: *Purpose of this paper* In the last decade, XML has become the defacto standard for data exchange in the World Wide Web. The positive benefits of data exchangeability to support system and software heterogeneity on application level and easy WWW integration make XML an ideal data format for many other application and network scenarios like wireless sensor networks. Moreover, the usage of XML encourages using standardized techniques like SOAP to adapt the service oriented paradigm to sensor network engineering. Nevertheless, integrating XML usage in wireless sensor network data management is limited by the low hardware resources that require efficient XML data management strategies suitable to bridge the general resource gap. *Design/methodology/approach* In this work we therefore present two separate strategies on integrating XML data management in wireless sensor networks that both have been implemented and are running on today's sensor node platforms. We further show how XML data can be processed and how XPath queries can be evaluated dynamically. In an extended evaluation we compare the performance of both strategies concerning the memory and energy efficiency and show that both solutions have application domains and are fully applicable on today's sensor node products. *Findings* This work shows that dynamic XML data management and query evaluation is possible on sensor nodes with strict limitations in terms of memory, processing power and energy supply. *What is original/value of paper* In this work we show an optimized stream-based XML compression technique and how XML queries can be evaluated on compressed XML bit streams using generic

pushdown automata. To the best of our knowledge and beside our own preparatory work on XML compression, this is the first complete approach on integrating dynamic XML data management into wireless sensor networks.

Published: in [103]

Bibliography

[1] K. Aberer, M. Hauswirth, and A. Salehi. A middleware for fast and flexible sensor network deployment. In *Proceedings of the 32nd international conference on Very large data bases*, page 1202. VLDB Endowment, 2006.

[2] K. Aberer, M. Hauswirth, and A. Salehi. Infrastructure for data processing in large-scale interconnected sensor networks. In *MDM '07: Proceedings of the 2007 International Conference on Mobile Data Management*, pages 198–205, Washington, DC, USA, 2007. IEEE Computer Society.

[3] S. Agarwal, R. Katz, S. Krishnamurthy, and S. Dao. Distributed power control in ad-hoc wireless networks. In *2001 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 2, 2001.

[4] O. Akan and I. Akyildiz. Event-to-sink reliable transport in wireless sensor networks. *IEEE/ACM Transactions on Networking (TON)*, 13(5):1016, 2005.

[5] I. Akyildiz, T. Melodia, and K. Chowdhury. A survey on wireless multimedia sensor networks. *Computer Networks*, 51(4):921–960, 2007.

[6] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[7] ALERT Systems Organization. *Alert history.* 2010. http://www.alertsystems.org.

[8] Altova. *XMLSpy - XML editor.* 2010. http://www.altova.com.

[9] T. Arampatzis, J. Lygeros, and S. Manesis. A survey of applications of wireless sensors and wireless sensor networks. In *Mediterrean Conference on Control and Automation Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on*, pages 719–724, 2005.

[10] C. Bachman. Summary of current work ANSI/X3/SPARC/study group: database systems. *ACM SIGMOD Record*, 6(3):39, 1974.

[11] S. Banerjee and A. Misra. Minimum energy paths for reliable communication in multi-hop wireless networks. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, page 156. ACM, 2002.

[12] P. Baronti, P. Pillai, V. Chook, S. Chessa, A. Gotta, and Y. Hu. Wireless sensor networks: A survey on the state of the art and the 802.15. 4 and ZigBee standards. *Computer Communications*, 30(7):1655–1695, 2007.

[13] S. Bellis, K. Delaney, B. O'Flynn, J. Barton, K. Razeeb, and C. O'Mathuna. Development of field programmable modular wireless sensor network nodes for ambient systems. *Computer Communications*, 28(13):1531–1544, 2005.

[14] T. Berners-Lee and D. Connolly. Hypertext markup language. *Internet Working Draft*, 13, 1993.

[15] J. Beutel. Metrics for sensor network platforms. In *Proc. ACM Workshop on Real-World Wireless Sensor Networks (REALWSN06)*, 2006.

[16] J. Beutel, P. Blum, M. Dyer, C. Moser, and P. Stadelmann. BTnode Programming-An Introduction to BTnut Applications. *Computer Engineering and Networks Lab, ETH Zuerich, Switzerland*, 1, 2007.

[17] J. Beutel, O. Kasten, F. Mattern, K. Roemer, F. Siegemund, and L. Thiele. Prototyping wireless sensor network applications with BTnodes. *Wireless Sensor Networks*, pages 323–338, 2004.

[18] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann. Wireless sensor networks - new challenges in software engineering. In *Proceedings of the ETFA '03. IEEE Conference*, volume 1, 2003.

[19] A. Bonifati and S. Ceri. Comparative analysis of five XML query languages. *ACM Sigmod Record*, 29(1):79, 2000.

[20] G. Booch, I. Jacobson, and J. Rumbaugh. Unified Modeling Language (UML). *Rational Software Corporation, Santa Clara, CA, version*, 1, 1997.

[21] D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture, W3C Working Group Note 11 February 2004. *World Wide Web Consortium, article available from: http://www. w3. org/TR/ws-arch*, 2004.

[22] I. Botan, D. Kossmann, P. M. Fischer, T. Kraska, D. Florescu, and R. Tamosevicius. Extending xquery with window functions. In *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*, pages 75–86. VLDB Endowment, 2007.

[23] M. Botts and A. Robins. Bringing the sensor web together. *Géosciences*, 6:46–53, 2007.

[24] R. Bourret. *XML data binding resources*. 2002. http://www. rpbourret. com/xml/XMLDataBinding.htm.

[25] I. N. Bronshtein and K. A. Semendyayev. *Handbook of mathematics (3rd ed.)*. Springer-Verlag, London, UK, 1997.

[26] M. Brundage. *XQuery: the XML query language*. Pearson Higher Education, 2004.

[27] D. Brunelli, L. Benini, C. Moser, and L. Thiele. An efficient solar energy harvester for wireless sensor nodes. In *Proceedings of the conference on Design, automation and test in Europe*, pages 104–109. ACM, 2008.

[28] BTnode Project @ ETH Zurich. *BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks*. 2007. http://www.btnode.ethz.ch/.

[29] P. Buneman, M. Grohe, and C. Koch. Path queries on compressed xml. In *vldb'2003: Proceedings of the 29th international conference on Very large data bases*, pages 141–152. VLDB Endowment, 2003.

[30] Q. Cao, T. Abdelzaher, T. He, and J. Stankovic. Towards optimal sleep scheduling in sensor networks for rare-event detection. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 4. IEEE Press, 2005.

[31] K. C. Carreras I., De Pellegrini F. and C. i. *Data Management in Wireless Sensor Networks*. IOS P, 2006.

[32] J. Cartigny, D. Simplot, and I. Stojmenovic. Localized minimum-energy broadcasting in ad-hoc networks. In *IEEE Societies INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications*, pages 2210–2217, 2003.

[33] A. Cerpa, J. Elson, D. Estrin, L. Girod, M. Hamilton, and J. Zhao. Habitat monitoring: Application driver for wireless communications technology. *ACM SIGCOMM Computer Communication Review*, 31(2 supplement):41, 2001.

[34] H. Chan and A. Perrig. Security and privacy in sensor networks. *Computer*, 36(10):103–105, 2003.

[35] N. Chand, R. C. Joshi, and M. Misra. Cooperative caching in mobile ad hoc networks based on data utility. *Mob. Inf. Syst.*, 3(1):19–37, 2007.

[36] J. Chang and L. Tassiulas. Energy conserving routing in wireless ad-hoc networks. In *IEEE INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, volume 1, 2000.

[37] C. Charalambous and S. Cui. A bio-inspired distributed clustering algorithm for wireless sensor networks. In *WICON '08: Proceedings of the 4th Annual International Conference on Wireless Internet*, pages 1–8, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[38] B. Charny. Wireless research senses the future. *ZDNet News*, pages 2100–1105, 2002.

[39] J. Cheney. Compressing xml with multiplexed hierarchical ppm models. In *Proceedings of DCC '01*, page 163, Washington, DC, USA, 2001. IEEE Computer Society.

[40] S. Chien, B. Cichy, A. Davies, D. Tran, G. Rabideau, R. Castano, R. Sherwood, D. Mandl, S. Frye, S. Shulman, J. Jones, and S. Grosvenor. An autonomous earth-observing sensorweb. *IEEE Intelligent Systems*, 20(3):16–24, 2005.

[41] C. Chong, S. Kumar, and B. Hamilton. Sensor networks: Evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, 2003.

[42] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web service definition language (WSDL). *W3C, http://www. w3. org/TR/wsdl, Accessed*, 18(2):2005, 2001.

[43] D. C. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica. The design and implementation of a declarative sensor network system. Technical Report UCB/EECS-2006-132, University of California, Berkeley, Oct 2006.

[44] J. Clark, M. Murata, et al. Relax NG specification. *OASIS Committee Specification*, 3, 2001.

[45] Coalesenses. *Coalesenses iSense Core Module*. 2010. http://www.coalesenses.com.

[46] R. Cole and G. Graefe. Optimization of dynamic query evaluation plans. In *Proceedings of the 1994 ACM SIGMOD international conference on Management of data*, pages 150–160. ACM, 1994.

[47] P. Corke, P. Valencia, P. Sikka, T. Wark, and L. Overs. Long-duration solar-powered wireless sensor networks. In *Proceedings of the 4th workshop on Embedded networked sensors*, page 37. ACM, 2007.

[48] A. Das, R. Marks, M. El-Sharkawi, P. Arabshahi, and A. Gray. Minimum power broadcast trees for wireless networks: integer programming formulations. In *IEEE INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 2, 2003.

[49] R. Data. W3Schools Online Web Tutorials. *Online Document available on: http://www. w3schools. com, Last visited: 23rd July*, 2008.

[50] F. C. Delicato, P. F. Pires, L. Pirmez, and L. F. Carmo. A service approach for architecting application independent wireless sensor networks. *Cluster Computing*, 8(2-3):211–221, 2005.

[51] F. C. Delicato, P. F. Pires, L. Pirmez, and L. F. R. da Costa Carmo. A flexible web service based architecture for wireless sensor networks. *International Conference on Distributed Computing Systems Workshops*, 00:730, 2003.

[52] K. Delin. The Sensor Web: A macro-instrument for coordinated sensing. *Sensors*, 2(1):270–285, 2002.

[53] K. Delin, S. Jackson, D. Johnson, S. Burleigh, R. Woodrow, J. McAuley, J. Dohm, F. Ip, T. Ferré, D. Rucker, et al. Environmental studies with the sensor web: Principles and practice. *Sensors*, 5(1-2):103–117, 2005.

[54] I. Demirkol, C. Ersoy, and F. Alagoz. MAC protocols for wireless sensor networks: a survey. *IEEE Communications Magazine*, 44(4):115–121, 2006.

[55] A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, page 599. VLDB Endowment, 2004.

[56] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 588–599. VLDB Endowment, 2004.

[57] A. Deshpande, S. Nath, P. B. Gibbons, and S. Seshan. Cache-and-query for wide area sensor databases. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 503–514, New York, NY, USA, 2003. ACM.

[58] Y. Diao, D. Ganesan, G. Mathur, and P. Shenoy. Rethinking data management for storage-centric sensor networks. In *Proceedings of the Third Biennial Conference on Innovative Data Systems Research (CIDR)*. Citeseer, 2007.

[59] T. Downing and R. Java. Remote Method Invocation. *Foster City, Calif.: IDG Books Worldwide*, 1998.

[60] F. Dressler, M. Struebe, R. Kapitza, and W. Schroeder-Preikschat. Dynamic Software Management on BTnode Sensors. In *4th IEEE/ACM International Conference on Distributed Computing in Sensor Systems (IEEE/ACM DCOSS 2008): IEEE/ACM International Workshop on Sensor Network Engineering (IWSNE 2008), Santorini Island, Greece (June 2008)*, pages 9–14. Citeseer, 2008.

[61] M. Droop, M. Flarer, J. Groppe, S. Groppe, V. Linnemann, J. Pinggera, F. Santner, M. Schier, F. Schoepf, H. Staffler, and S. Zugal. Embedding XPATH Queries into SPARQL Queries. In J. Cordeiro and J. Filipe, editors, *Proceedings of the 10th International Conference on Enterprise Information Systems, Volume DISI, (ICEIS 2008)*, pages 5–14, Barcelona, Spain, June 12 - 16 2008. INSTICC.

[62] M. Dubinin, A. Lushchekina, and V. Radeloff. Performance and accuracy of Argos transmitters for wildlife monitoring in Southern Russia. *European Journal of Wildlife Research*, 56(3):459–463, 2010.

[63] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *Proceedings of 2001 International Parallel and Distributed Processing Symposium (IPDPS)*, San Francisco, CA, USA, 2001. Published by the IEEE Computer Society.

[64] J. Elson, L. Girod, and D. Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[65] J. Elson and K. Roemer. Wireless sensor networks: A new regime for time synchronization. *ACM SIGCOMM Computer Communication Review*, 33(1):154, 2003.

[66] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, page 270. ACM, 1999.

[67] ExoLab Group. *The Castor Project*. 2010. http://www.castor.org/.

[68] C. Fok, G. Roman, and C. Lu. Mobile agent middleware for sensor networks: An application case study. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 51. IEEE Press, 2005.

[69] C. Fok, G. Roman, and C. Lu. Rapid development and flexible deployment of adaptive wireless sensor network applications. In *25th IEEE International Conference on Distributed Computing Systems, 2005. ICDCS 2005. Proceedings*, pages 653–662, 2005.

[70] M. Franceschet. *XPathMark: An XPath Benchmark for the XMark Generated Data*. 2005.

[71] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla. The impact of multihop wireless channel on TCP throughput and loss. In *IEEE INFOCOM*, volume 3, pages 1744–1753. Citeseer, 2003.

[72] D. Ganesan, B. Greenstein, D. Estrin, J. Heidemann, and R. Govindan. Multiresolution storage and search in sensor networks. *ACM Transactions on Storage (TOS)*, 1(3):315, 2005.

[73] J. Gao, D. Yang, S. Tang, and T. Wang. Tree automata based efficient XPath evaluation over XML data stream. *Ruan Jian Xue Bao(J. Softw.)*, 16(2):223–232, 2005.

[74] 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 *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, page 11. ACM, 2003.

[75] GCC. *GCC, the GNU Compiler Collection*. 2010. http://gcc.gnu.org/.

[76] A. Ghose, J. Grossklags, and J. Chuang. Resilient data-centric storage in wireless ad-hoc sensor networks. In *Mobile Data Management*, pages 45–62. Springer, 2003.

[77] GlobalSecurity.org. Sound surveillance system (sosus), 2005.

[78] A. Gokhale, B. Kumar, and A. Sahuguet. Reinventing the wheel? CORBA vs. Web services. In *International WWW Conference*, 2002.

[79] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of xpath query evaluation and xml typing. *J. ACM*, 52(2):284–335, 2005.

[80] G. Gou and R. Chirkova. Efficient algorithms for evaluating XPath over streams. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 269–280. ACM, 2007.

[81] W. Gould. From Swallow floats to Argo–the development of neutrally buoyant floats. *Deep Sea Research Part II: Topical Studies in Oceanography*, 52(3-4):529–543, 2005.

[82] R. Govindan, J. Hellerstein, W. Hong, S. Madden, M. Franklin, and S. Shenker. The sensor network as a database. Technical report, Citeseer, 2002.

[83] G. Graefe. Query evaluation techniques for large databases. *ACM computing Surveys*, 25(2):73–170, 1993.

[84] S. Groppe, J. Groppe, V. Linnemann, D. Kukulenz, N. Hoeller, and C. Reinke. Embedding SPARQL into XQuery / XSLT. In R. L. Wainwright and H. Haddad, editors, *Proceedings of the 23rd ACM Symposium on Applied Computing (ACM SAC 2008)*, pages 2271–2278, Fortaleza, Ceara, Brasilien, March 16 - 20 2008. ACM.

[85] S. Groppe, J. Groppe, C. Reinke, N. Hoeller, and V. Linnemann. *Open and novel issues in XML database applications: future directions and advanced technologies*, chapter XSLT: Common Issues with XQuery and Special Issues of XSLT, pages 108–135. IGI Global, Information Science Reference, USA/UK, 2009.

[86] L. Gurgen, C. Roncancio, C. Labbé, A. Bottaro, and V. Olive. Sstreamware: a service oriented middleware for heterogeneous sensor data management. In *Proceedings of ICPS '08*, pages 121–130, New York, NY, USA, 2008. ACM.

[87] Gzip. 2010. http://www.gzip.org/.

[88] V. Handziski, J. Polastre, J. Hauer, C. Sharp, A. Wolisz, and D. Culler. Flexible hardware abstraction for wireless sensor networks. In *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005*, pages 145–157, 2005.

[89] W. Heinzelman, A. Chandrakasan, and H. Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences, 2000*, page 10, 2000.

[90] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The gator tech smart house: A programmable pervasive space. *Computer*, pages 50–60, 2005.

[91] J. Hill and D. Culler. A wireless embedded sensor architecture for system-level optimization. Technical report, 2001.

[92] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *ACM Sigplan Notices*, 35(11):104, 2000.

[93] J. L. Hill and D. E. Culler. Mica: A wireless platform for deeply embedded networks. *IEEE Micro*, 22:12–24, 2002.

[94] R. Hills. Sensing for danger. *Sci. Technol. Rep*, 2001.

[95] N. Hoeller. Dynamic Approximative Data Caching in Wireless Sensor Networks. In *Proceedings of the 11th International Conference on Mobile Data Management (MDM 2010)*, pages 291–292, Kansas City, Missouri, USA, May 23 - 26 2010. IEEE. This paper received the BEST PHD FORUM PAPER AWARD MDM 2010.

[96] N. Hoeller, C. Reinke, S. Groppe, and V. Linnemann. Xobe Sensor Networks: Integrating XML in sensor network programming. In *Proceedings of INSS 2008*, Kanazawa, Japan, June 17 - 19 2008. IEEE.

[97] N. Hoeller, C. Reinke, D. Kukulenz, and V. Linnemann. smartCQ: Answering and Evaluating Bounded Continuous Search Queries within the WWW and Sensor Networks. In *Fifth International Conference on Innovations in Information Technology (Innovations 2008)*, pages 160–164, Al Ain, United Arab Emirates, December 16 - 18 2008. IEEE.

[98] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, D. Boeckmann, and V. Linnemann. Efficient XML Usage within Wireless Sensor Networks. In X. Wang and N. B. Shroff, editors, *Proceedings of the Fourth International Wireless Internet Conference (WICON 2008)*, ACM International Conference Proceeding Series (AICPS), page Article No: 74, Maui, Hawaii, USA, November 17 - 19 2008. ACM.

[99] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, F. Frischat, and V. Linnemann. DACS: A Dynamic Approximative Data Caching in Wireless Sensor Networks. In *Proceedings of the 5th International Conference on Digital Information Management (ICDIM 2010)*, pages 339–346, Thunder Bay, Canada, July 5 - 8 2010. IEEE.

[100] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, M. Lipphardt, B. Schütt, and V. Linnemann. Stream-based XML Template Compression for Wireless Sensor Network Data Management. In *Proceedings of the 4th International Conference on Multimedia and Ubiquitous Engineering (MUE 2010)*, Cebu, Philippines, August 11 - 13 2010. IEEE. This paper received the BEST PAPER AWARD MUE 2010.

[101] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Towards Energy Efficient XPath Evaluation in Wireless Sensor Networks. In *ACM*

International Conference Proceeding Series - Proceedings of the 6th International Workshop on Data Management for Sensor Networks (DMSN 2009), pages 1–2, Lyon, Frankreich, August 24 - 28 2009. ACM.

[102] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. XML Data Management and XPath Evaluation in Wireless Sensor Networks. In G. Kotsis, D. Taniar, E. Pardede, and I. Khalil, editors, *Proceedings of the 7th International Conference on Advances in Mobile Computig & Multimedia (MoMM 2009), held in conjunction with iiWAS 2009 conference*, pages 218–227, Kuala Lumpur, Malaysia, December 14 - 16 2009. ACM. This paper received the BEST PAPER AWARD MoMM 2009.

[103] N. Hoeller, C. Reinke, J. Neumann, S. Groppe, C. Werner, and V. Linnemann. Efficient xml data and query integration in the wireless sensor network engineering process. *International Journal of Web Information Systems*, 6(4), 2010.

[104] H. Hof, E. Blaß, and M. Zitterbart. Secure overlay for service centric wireless sensor networks. *Security in Ad-hoc and Sensor Networks*, pages 125–138, 2005.

[105] C. Hsu, D. Levermore, C. Carothers, and G. Babin. Enterprise collaboration: On-demand information exchange using enterprise databases, wireless sensor networks, and RFID systems. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 37(4):519–532, 2007.

[106] L. Hu and D. Evans. Localization for mobile sensor networks. In *Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 45–57. ACM, 2004.

[107] D. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[108] M. hun Lee, K. bok Eom, H. joong Kang, C. sun Shin, and H. Yoe. Design and implementation of wireless sensor network for ubiquitous glass houses. *icis*, 0:397–400, 2008.

[109] International Telecommunication Union (ITU). Recommendation x.891: Generic applications of asn.1 – fast infoset, May 2005.

[110] K. Jamieson, H. Balakrishnan, and Y. Tay. Sift: A MAC protocol for event-driven wireless sensor networks. *Wireless Sensor Networks*, pages 260–275, 2006.

[111] D. Jardine. *The ansi/sparc dbms model*. North-Holland, 1977.

[112] D. Jensen. SIVAM: Communication, navigation and surveillance for the Amazon. *Avionics Mag*, 2002.

[113] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, and D. Rubenstein. Energy-efficient computing for wildlife tracking: Design tradeoffs and early experiences with zebranet. *ACM SIGOPS operating systems review*, 36(5):96–107, 2002.

[114] E.-H. Jung and Y.-J. Park. Tinyonet: A cache-based sensor network bridge enabling sensing data reusability and customized wireless sensor network services. *Sensors*, 8(12):7930–7950, 2008.

[115] J. M. Kahn, R. H. Katz, and K. S. J. Pister. Next century challenges: mobile networking for "smart dust". In *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 271–278, New York, NY, USA, 1999. ACM.

[116] K. Kalpakis, K. Dasgupta, and P. Namjoshi. Efficient algorithms for maximum lifetime data gathering and aggregation in wireless sensor networks* 1. *Computer Networks*, 42(6):697–716, 2003.

[117] K. Kalpakis and S. Tang. A combinatorial algorithm for the maximum lifetime data gathering with aggregation problem in sensor networks. *Computer Communications*, 32(15):1655–1665, 2009.

[118] A. Kamra, V. Misra, J. Feldman, and D. Rubenstein. Growth codes: Maximizing sensor network data persistence. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, page 266. ACM, 2006.

[119] H. Karl and A. Willig. A short survey of wireless sensor networks. *Telecommunication Networks group, Technical Report*, 2003.

[120] H. Katz and D. Chamberlin. *XQuery from the experts: a guide to the W3C XML query language*. Addison-Wesley Professional, 2004.

[121] M. Kay. *XPath 2.0 programmer's reference*. Wrox, 2004.

[122] M. Kempa and V. Linnemann. Towards Valid XML Applications. In V. Milutinovic, editor, *Proceedings of the Third International Conference on Advances in Infrastructure for Electronic Business, Education, Science, and Medicin on the Internet, SSGRR 2002w*, L'Aquila, Italien, January 2002. Morgan Kaufmann. nur erhältlich als CD mit ISBN: 88-85280-62-5.

[123] J. Kent and H. Brumbaugh. autosql and autoxml: code generators from the genome project. *Linux J.*, 2002(99):1, 2002.

[124] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Wireless sensor networks for structural health monitoring. In *Proceedings of the 4th international conference on Embedded networked sensor systems*, page 428. ACM, 2006.

[125] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon. Health monitoring of civil infrastructures using wireless sensor networks. In *Proceedings of the 6th international conference on Information processing in sensor networks*, page 263. ACM, 2007.

[126] P. Kinney et al. Zigbee technology: Wireless control that simply works. In *Communications design conference*, volume 2, 2003.

[127] D. Knuth. *Computers & Typesetting*. Addison-Wesley Reading, MA, 1984.

[128] D. Knuth. Dynamic huffman coding. *Journal of Algorithms*, 6(2):163–180, 1985.

[129] D. Kotz, C. Newport, and C. Elliot. The mistaken axioms of wireless-network research. Technical report, Dartmouth College Computer Science, 2003.

[130] E. Kranakis, H. Singh, and J. Urrutia. Compass routing on geometric networks. In *Proc. 11th Canadian Conference on Computational Geometry*, pages 51–54, 1999.

[131] A. Kroeller, D. Pfisterer, C. Buschmann, S. Fekete, and S. Fischer. Shawn: A new approach to simulating wireless sensor networks. *Arxiv preprint cs.DC/0502003*, 2005.

[132] D. Kukulenz, N. Hoeller, S. Groppe, and V. Linnemann. Optimization of Bounded Continuous Search Queries based on Ranking Distributions. In *Proceedings 8th International Conference on Web Information Systems Engineering (WISE 2007)*, volume 4831 of *Lecture Notes in Computer Science (LNCS)*, pages 26–37, Nancy, Frankreich, December 3 - 7 2007. Springer-Verlag.

[133] D. Kukulenz, C. Reinke, and N. Hoeller. Web Contents Tracking by Learning of Page Grammars. In *Proceedings of the Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 416–425, Athens, Greece, June 8 - 13 2008. IEEE.

[134] S. Kurkowski, T. Camp, and M. Colagrosso. Manet simulation studies: the incredibles. *SIGMOBILE Mob. Comput. Commun. Rev.*, 9(4):50–61, 2005.

[135] M. Kushwaha, I. Amundson, X. Koutsoukos, S. Neema, and J. Sztipanovits. Oasis: A programming framework for service-oriented sensor networks. In *In IEEE/Create-Net COMSWARE 2007*, January 2007.

[136] L. Lamport. *LATEX: A document preparation system. User's guide and reference manual*. Addison-Wesley Publishing Company, Reading, Ma, 1994.

[137] K. Langendoen and N. Reijers. Distributed localization in wireless sensor networks: a quantitative comparison. *Computer Networks*, 43(4):499–518, 2003.

[138] D. Lee and W. Chu. Comparative analysis of six XML schema languages. *ACM Sigmod Record*, 29(3):87, 2000.

[139] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan. Service oriented architecture for heterogeneous and dynamic sensor networks. In *Proceedings of DEBS '08*, pages 309–312, New York, NY, USA, 2008. ACM.

[140] M. Leopold, M. Dydensborg, and P. Bonnet. Bluetooth and sensor networks: A reality check. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, page 113. ACM, 2003.

[141] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, et al. Tinyos: An operating system for sensor networks. *Ambient Intelligence*, pages 115–148, 2005.

[142] D. Li, K. Wong, Y. Hu, and A. Sayeed. Detection, classification and tracking of targets in distributed sensor networks. *IEEE signal processing magazine*, 19(2):17–29, 2002.

[143] S. Li, Y. Lin, S. Son, J. Stankovic, and Y. Wei. Event detection services using data service middleware in distributed sensor networks. *Telecommunication Systems*, 26(2):351–368, 2004.

[144] Y. Li, M. V. Ramakrishna, and S. W. Loke. Approximate query answering in sensor networks with hierarchically distributed caching. *Advanced Information Networking and Applications, International Conference on*, 2:281–285, 2006.

[145] Y. Li, M. V. Ramakrishna, and S. W. Loke. An optimal distribution of data reduction in sensor networks with hierarchical caching. In *EUC*, pages 598–609, 2007.

[146] H. Liefke and D. Suciu. Xmill: an efficient compressor for xml data. In *SIGMOD '00: Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 153–164, New York, NY, USA, 2000. ACM.

[147] S. Lin, B. Arai, and D. Gunopulos. Reliable hierarchical data storage in sensor networks. In *19th International Conference on Scientific and Statistical Database Management, 2007. SSBDM'07*, pages 26–26, 2007.

[148] Y. Lin, B. Liang, and B. Li. Data persistence in large-scale sensor networks with decentralized fountain codes. In *IEEE INFOCOM 2007. 26th IEEE International Conference on Computer Communications*, pages 1658–1666, 2007.

[149] V. Linnemann, S. Fischer, and C. Werner. Aesop's tale - applying and extending the service-oriented paradigm to sensor network application engineering. Technical report, University of Luebeck, 2006.

[150] V. Linnemann and M. Kempa. XML-Objekte: das Projekt XOBE. Schriftenreihe der Institute für Informatik/Mathematik A-02-10, Technisch-Naturwissenschaftliche Fakultät, Medizinische Universität zu Lübeck, Mai 2002.

[151] M. Lipphardt. *Service-orientierte Infrastrukturen und Algorithmen fuer praxistaugliche Sensornetzanwendungen*. Institute of Telematics, University of Luebeck, 2010.

[152] M. Lipphardt, N. Glombitza, J. Neumann, and C. Werner. A service-oriented operating system and an application development infrastructure for wireless sensor networks. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 309–310. ACM, 2009.

[153] M. Lipphardt, H. Hellbrück, D. Pfisterer, S. Ransom, and S. Fischer. Practical experiences on mobile inter-body-area-networking. In *BodyNets '07*, pages 1–8. ICST, 2007.

[154] M. Lipphardt, J. Neumann, N. Hoeller, C. Reinke, S. Groppe, V. Linnemann, and C. Werner. XML und SOA als Wegbereiter fuer Sensornetze in der Praxis. *Praxis der Informationsverarbeitung und Kommunikation (PIK)*, 31(3):146–152, Juli - September 2008.

[155] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[156] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Tinydb: an acquisitional query processing system for sensor networks. *ACM Trans. Database Syst.*, 30(1):122–173, 2005.

[157] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM.

[158] S. Mangano and S. Laurent. *XSLT cookbook*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2002.

[159] G. Mao, B. Fidan, and B. Anderson. Wireless sensor network localization techniques. *Computer Networks*, 51(10):2529–2553, 2007.

[160] P. Marchal. Field-programmable gate arrays. *Communications of the ACM*, 42(4):59, 1999.

[161] M. Marin-Perianu, N. Meratnia, P. Havinga, L. de Souza, J. Muller, P. Spiess, S. Haller, T. Riedel, C. Decker, and G. Stromberg. Decentralized enterprise systems: a multiplatform wireless sensor network approach. *IEEE Wireless Communications*, 14(6):57–66, 2007.

[162] R. Marin-Perianu, H. Scholten, and P. Havinga. Prototyping service discovery and usage in wireless sensor networks. *IEEE Conference on Local Computer Networks (LCN)*, 0:841–850, 2007.

[163] M. Maroti, G. Simon, A. Ledeczi, and J. Sztipanovits. Shooter localization in urban terrain. *Computer*, 37(8):60–61, 2004.

[164] S. Microsystems. Rpc: Remote procedure call protocol specification, 1988.

[165] E. Miluzzo, X. Zheng, K. Fodor, and A. Campbell. Radio characterization of 802.15. 4 and its impact on the design of mobile sensor networks. In *Proceedings of the 5th European conference on Wireless sensor networks*, pages 171–188. Springer-Verlag, 2008.

[166] J.-K. Min, M.-J. Park, and C.-W. Chung. Xpress: a queriable compression for xml data. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 122–133, New York, NY, USA, 2003. ACM.

[167] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[168] R. Mueller, G. Alonso, and D. Kossmann. Swissqm: Next generation data processing in sensor networks. In *CIDR*, pages 1–9. www.crdrdb.org, 2007.

[169] L. Nachman, R. Kling, R. Adler, J. Huang, and V. Hummel. The Intel® Mote platform: a Bluetooth-based sensor network for industrial monitoring. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 61. IEEE Press, 2005.

[170] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. In *Proceedings of SenSys '04*, pages 250–262, New York, NY, USA, 2004. ACM.

[171] J. Neumann, N. Hoeller, C. Reinke, and V. Linnemann. Redundancy Infrastructure for Service-oriented Wireless Sensor Networks. In *Proceedings of the 9th International Symposium on Network Computing and Applications (NCA 2010)*, pages 269–274, Cambridge, Massachusetts, USA, July 15 - 17 2010. IEEE.

[172] J. Neumann, C. Reinke, N. Hoeller, and V. Linnemann. Adaptive Quality-Aware Replication in Wireless Sensor Networks. In *Communication and Networking, Proceedings of the 2009 International Workshop on Wireless Ad Hoc, Mesh and Sensor Networks (WAMSNET09), held in conjunction with the 2009 International Conference on Future Generation Communication and Networking(FGCN 2009)*, volume 56 of *Communications in Computer and Information Science (CCIS)*, pages 413–420, Jeju Island, Korea, Dezember 10 - 12 2009. Springer.

[173] W. Ng, W.-Y. Lam, P. T. Wood, and M. Levene. Xcq: A queriable xml compression system. *Knowl. Inf. Syst.*, 10(4):421–452, 2006.

[174] C. Nishimura and D. Conlon. IUSS dual use: Monitoring whales and earthquakes using SOSUS. *Marine Technology Society Journal*, 27:13–13, 1994.

[175] Object Management Group. *OMG Interface Definition Language (IDL) specification*. OMG, United States, 1991.

[176] B. O'Flynn, S. Bellis, K. Delaney, J. Barton, S. O'mathuna, A. Barroso, J. Benson, U. Roedig, and C. Sreenan. The development of a novel minaturized modular

platform for wireless sensor networks. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 49. IEEE Press, 2005.

[177] T. Okoshi, S. Wakayama, Y. Sugita, S. Aoki, T. Iwamoto, J. Nakazawa, T. Nagata, D. Furusaka, M. Iwai, A. Kusumoto, et al. Smart space laboratory project: Toward the next generation computing environment. In *IEEE Third Workshop on Networked Appliances (IWNA 2001)*. Citeseer, 2001.

[178] D. Olteanu, H. Meuss, T. Furche, and F. Bry. XPath: looking forward. In *XML-Based Data Management and Multimedia Engineering—EDBT 2002 Workshops*, pages 892–896. Springer, 2002.

[179] E. Ort and B. Mehta. Java architecture for xml binding (jaxb), 2010. http://java.sun.com/developer/earlyAccess/xml/jaxb/.

[180] S. Pakzad, G. Fenves, S. Kim, and D. Culler. Design and implementation of scalable wireless sensor network for structural monitoring. *Journal of infrastructure systems*, 14:89, 2008.

[181] S. Pakzad, S. Kim, G. Fenves, S. Glaser, D. Culler, and J. Demmel. Multi-Purpose Wireless Accelerometers for Civil Infrastructures Monitoring. *Structural health monitoring, 2005: advancements and challenges for implementation*, page 125, 2005.

[182] J. Paradiso, G. Borriello, L. Girod, and R. Han. Applications: Beyond dumb data collection (panel). EmNets, 2006.

[183] P. Parys. XPath evaluation in linear time with polynomial combined complexity. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 55–64. ACM, 2009.

[184] S. Pemberton et al. XHTML (TM) 1.0 The Extensible HyperText Markup Language. 2000.

[185] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Communications of the ACM*, 47(6):53–57, 2004.

[186] A. Perrig, R. Szewczyk, J. Tygar, V. Wen, and D. Culler. SPINS: Security protocols for sensor networks. *Wireless networks*, 8(5):534, 2002.

[187] J. Polastre, R. Szewczyk, and D. Culler. Telos: enabling ultra-low power wireless research. In *Fourth International Symposium on Information Processing in Sensor Networks, 2005. IPSN 2005*, pages 364–369, 2005.

[188] J. Postel et al. User datagram protocol, 1980.

[189] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Commun. ACM*, 43(5):51–58, 2000.

[190] K. S. Prabh and T. F. Abdelzaher. Energy-conserving data cache placement in sensor networks. *ACM Trans. Sen. Netw.*, 1(2):178–203, 2005.

[191] J. M. Prinsloo, C. L. Schulz, D. G. Kourie, W. H. M. Theunissen, T. Strauss, R. V. D. Heever, and S. Grobbelaar. A service oriented architecture for wireless sensor and actor network applications. In *SAICSIT '06*, pages 145–154, , Republic of South Africa, 2006. South African Institute for Computer Scientists and Information Technologists.

[192] B. Priyantha, A. Kansal, M. Goraczko, and F. Zhao. Tiny web services for sensor device interoperability. In *Proceedings of IPSN '08*, pages 567–568, Washington, DC, USA, 2008. IEEE Computer Society.

[193] B. Przydatek, D. Song, and A. Perrig. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, page 265. ACM, 2003.

[194] QUALCOMM, CDMA Technologies GmbH. *Snapdragon - Technical Specifications*. 2010.

[195] M. A. Rahman and S. Hussain. Effective caching in wireless sensor network. *Advanced Information Networking and Applications Workshops, International Conference on*, 1:43–47, 2007.

[196] V. Rajesh, J. Gnanasekar, R. Ponmagal, and P. Anbalagan. Integration of Wireless Sensor Network with Cloud. In *Proceedings of the 2010 International Conference on Recent Trends in Information, Telecommunication and Computing*, pages 321–323. IEEE Computer Society, 2010.

[197] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker. GHT: a geographic hash table for data-centric storage. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, page 87. ACM, 2002.

[198] C. Reinke, N. Hoeller, and V. Linnemann. Adaptive Atomic Transaction Support for Service Migration in Wireless Sensor Networks. In *Proceedings of the Seventh IEEE and IFIP International Conference on Wireless and Optical Communications Networks (WOCN2010)*, Colombo, Sri Lanka, September 6 - 8 2010. IEEE.

[199] C. Reinke, N. Hoeller, M. Lipphardt, J. Neumann, S. Groppe, and V. Linnemann. Integrating Standardized Transaction Protocols in Service Oriented Wireless Sensor Networks. In *Proceedings of the 24th ACM Symposium on Applied Computing (ACM SAC 2009)*, pages 2202–2203, Honolulu, Hawaii, USA, March 8 - 12 2009. ACM.

[200] C. Reinke, N. Hoeller, J. Neumann, S. Groppe, S. Werner, and V. Linnemann. Analysis and Comparison of Atomic Commit Protocols for Adaptive Usage in Wireless Sensor Networks. In *Proceedings of the 2010 IEEE International Conference on Sensor Networks, Ubiquitous and Trustworthy Computing (SUTC 2010)*, pages 138–145, Newport Beach, California, USA, June 7 - 9 2010. IEEE.

[201] K. Roemer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, 2004.

[202] S. Roundy, P. Wright, and J. Rabaey. A study of low level vibrations as a power source for wireless sensor nodes. *Computer Communications*, 26(11):1131–1144, 2003.

[203] J. Sandhu, A. Agogino, A. Agogino, et al. Wireless sensor networks for commercial lighting control: decision making with multi-agent systems. In *AAAI workshop on sensor networks*, pages 131–140. Citeseer, 2004.

[204] P. Santi. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys (CSUR)*, 37(2):194, 2005.

[205] T. Schmid, H. Dubois-Ferriere, and M. Vetterli. Sensorscope: Experiences with a wireless building monitoring sensor network. In *Workshop on Real-World Wireless Sensor Networks*, 2005.

[206] A. Schmidt, F. Waas, M. L. Kersten, M. J. Carey, I. Manolescu, and R. Busse. Xmark: A benchmark for xml data management. In *VLDB*, pages 974–985, 2002.

[207] H. Schuhart, D. Pietzsch, and V. Linnemann. Framework of the XOBE Database Programming Language. In *Proceedings of the IADIS International Conference Applied Computing (IADIS-AC 2005)*, volume I, pages 193–200, Carvoeiro, Portugal, February 22-25, 2005. IADIS Press.

[208] M. Schwabl-Schmidt. Programmiertechniken fuer AVR-Mikrocontroller. *Aachen: Elektor-Verlag*, 2007.

[209] K. Seada, M. Zuniga, A. Helmy, and B. Krishnamachari. Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 108–121. ACM, 2004.

[210] V. Shankar, A. Natarajan, S. Gupta, and L. Schwiebert. Energy-efficient protocols for wireless communication in biosensornetworks. In *2001 12th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, volume 1, 2001.

[211] S. Sharples, V. Callaghan, and G. Clarke. A multi-agent architecture for intelligent building sensing and control. *Sensor Review*, 19(2):135–140, 1999.

[212] J. Shi and W. Liu. A service-oriented model for wireless sensor networks with internet. In *CIT '05: Proceedings of the The Fifth International Conference on Computer and Information Technology*, pages 1045–1049, Washington, DC, USA, 2005. IEEE Computer Society.

[213] N. Shrivastava, C. Buragohain, D. Agrawal, and S. Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249. ACM, 2004.

[214] G. Simon, M. Maróti, A. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai, and K. Frampton. Sensor network-based countersniper system. In *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 1–12, New York, NY, USA, 2004. ACM.

[215] F. Sivrikaya and B. Yener. Time synchronization in sensor networks: a survey. *IEEE network*, 18(4):45–50, 2004.

[216] J. Snell, D. Tidwell, and P. Kulchenko. *Programming Web services with SOAP*. O'Reilly Media, 2002.

[217] Software AG. *XML Benefits*. 2010. www.softwareag.com.

[218] F. Stann and J. Heidemann. RMST: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protocols and Applications*, pages 102–112. Citeseer, 2003.

[219] D. Steere, A. Baptista, D. McNamee, C. Pu, and J. Walpole. Research challenges in environmental observation and forecasting systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, page 299. ACM, 2000.

[220] M. Steffen and I. UiO. XML Query Languages. 2007.

[221] P. Steggles and S. Gschwind. The Ubisense smart space platform. In *Adjunct Proceedings of the Third International Conference on Pervasive Computing*, volume 191, 2005.

[222] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 214–226. ACM, 2004.

[223] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Commun. ACM*, 47(6):34–40, 2004.

[224] The Apache Software foundation. *Math Framework*. 2003. http://jakarta. apache. org/bsf.

[225] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. *Distributed Computing in Sensor Systems*, pages 307–321, 2005.

[226] M. Tubaishat and S. Madria. Sensor networks: an overview. *IEEE potentials*, 22(2):20–23, 2003.

[227] D. Veillard. The XML C parser and toolkit of Gnome: libxml. *System Home page at http://xmlsoft. org*, 2010.

[228] M. Vieira, C. Coelho Jr, D. da Silva Jr, and J. da Mata. Survey on wireless sensor network devices. In *IEEE Conference Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA'03*, pages 537–544, 2003.

[229] S. Vinoski. CORBA: Integrating diverse applications within distributed heterogeneous environments. *IEEE Communications Magazine*, 35(2):46–55, 1997.

[230] W3C. Xml specification W3C recommendation, 1998.

[231] W3C. Xsl transformations (xslt) version 1.0, 1999. http://www.w3.org/TR/xslt.

[232] W3C. Soap version 1.2 part 1: Messaging framework, 2001. http://www.w3.org/TR/soap/.

[233] W3C. W3C XML Schema. *W3C Recommendation*, 2001.

[234] W3C. Xml path language (xpath) 2.0, 2007. http://www.w3.org/TR/xpath20/.

[235] W3C. Xquery 1.0: An xml query language, 2007. http://www.w3.org/TR/xquery/.

[236] W3C. Working draft: Efficient xml interchange (exi) format 1.0, Sept. 2008.

[237] W3C. Technical reports and recommendations, 2010. http://www.w3.org/TR.

[238] A. Walsh. UDDI, SOAP, and WSDL: The Web Services Specification Reference Book. 2002.

[239] N. Walsh. A technical introduction to XML. *World Wide Web Journal*, 1998.

[240] C. Wan, A. Campbell, and L. Krishnamurthy. PSFQ: a reliable transport protocol for wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, page 11. ACM, 2002.

[241] Y. Wang. Topology control for wireless sensor networks. *Wireless Sensor Networks and Applications*, pages 113–147, 2008.

[242] B. Warneke, M. Scott, B. Leibowitz, L. Zhou, C. Bellew, J. Chediak, J. Kahn, B. Boser, and K. Pister. An autonomous 16 mm3 solar-powered node for distributed wireless sensor networks. In *Proc. IEEE Sensors*, volume 1, pages 1510–1515. Citeseer, 2002.

[243] M. Weiser. The computer for the 21st century. *Scientific American*, 272(3):78–89, 1991.

[244] C. Werner, C. Buschmann, Y. Brandt, and S. Fischer. Compressing soap messages by using pushdown automata. *icws*, 0:19–28, 2006.

[245] C. Werner, C. Buschmann, Y. Brandt, and S. Fischer. XML Compression for Web Services on Resource-Constrained Devices. *International Journal of Web Services Research*, 5(3), 2008.

[246] G. Werner-Allen, K. Lorincz, M. Welsh, O. Marcillo, J. Johnson, M. Ruiz, and J. Lees. Deploying a wireless sensor network on an active volcano. *IEEE Internet Computing*, 10(2):18–25, 2006.

[247] J. Wu and I. Stojmenovic. Ad hoc networks. *IEEE COMPUTER SOCIETY*, 37(2):29–31, 2004.

[248] N. Xu. A survey of sensor network applications. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[249] N. Xu, S. Rangwala, K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin. A wireless sensor network for structural monitoring. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 13–24. ACM, 2004.

[250] Y. Yang, X. Wang, S. Zhu, and G. Cao. SDAP: A secure hop-by-hop data aggregation protocol for sensor networks. *ACM Transactions on Information and System Security (TISSEC)*, 11(4):18, 2008.

[251] S. Yao. Optimization of query evaluation algorithms. *ACM Transactions on Database Systems (TODS)*, 4(2):133–155, 1979.

[252] Y. Yao, S. Alam, J. Gehrke, and S. Servetto. Network scheduling for data archiving applications in sensor networks. In *Proceedings of the 3rd workshop on Data management for sensor networks: in conjunction with VLDB 2006*, page 25. ACM, 2006.

[253] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[254] J. Yick, B. Mukherjee, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[255] Y. Yu, R. Govindan, and D. Estrin. Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. *UCLA Computer Science Department Technical Report, UCLA-CSD TR-01-0023*, 2001.

[256] P. Zhang, C. Sadler, S. Lyon, and M. Martonosi. Hardware design experiences in zebranet. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 227–238. ACM, 2004.

[257] H. Zimmermann. *OSI reference model—The ISO model of architecture for open systems interconnection*. Artech House, Inc., Norwood, MA, USA, 1988.