Efficient XML Usage within Wireless Sensor Networks

Nils Hoeller, Christoph Reinke, Jana Neumann, Sven Groppe, Daniel Boeckmann,
Volker Linnemann
Institute of Information Systems
Ratzeburger Allee 160
23538 Luebeck, Germany
{hoeller, reinke, neumann, groppe, boeckmann, linnemann}@ifis.uni-luebeck.de

ABSTRACT

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. XML has evolved to the de facto standard data exchange format between heterogeneous networks 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.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications, Distributed databases*; H.2.4 [**Database Management**]: Systems—*Distributed Databases, Query Processing*

General Terms

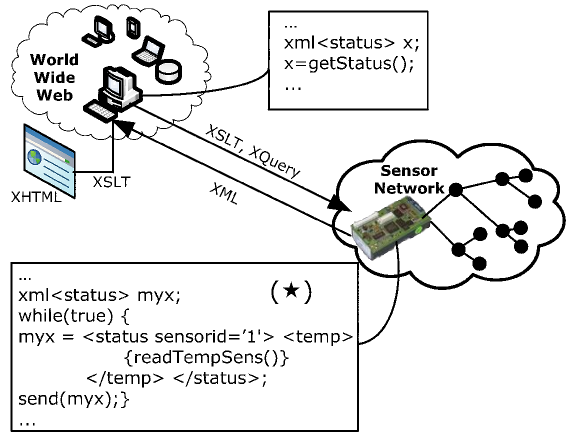
Languages, Management, Design

Keywords

XML, Wireless Sensor Networks, Database Management, Programming Language

1. INTRODUCTION

With the rapidly advancing development in the field of microprocessor technology, which results into smaller and more powerful microprocessors, wireless sensor networks are becoming increasingly important. Wireless sensor networks consist of single sensor nodes that combine regular computing devices with different sensors for monitoring environmental conditions and events. Recently, many sensor network deployments have been reported for different application fields from geological environment monitoring to ubiquitous computing applications [17, 30, 12]. Nevertheless, sensor network programming remains a highly complex development task and the integration of sensor networks in heterogeneous networks has only been proposed inchoately. Besides, recent sensor network data management approaches only support very simple data structures, like one table per network [31, 16], while there is a need for more complex data structures like those in existing database systems. Hence, the use of a standardized exchange format is desirable while handling this format during programming should be easy and transparent for the developer. A highly exchangeable and extensible data format is XML, which has become the de facto standard for data exchange over the Web. Different query languages, like XPath [26], XSLT [28] and XQuery [27], have been introduced. Using XML in sensor networks encourages the interchangeability of different types of sensors and systems, e.g. making it easy to interconnect a sensor network to the WWW. In Figure 1 we show a possible scenario. A sensor network can directly be queried using XML query languages by any client that is able to process XML, while the XML result can be further processed, e.g. in order to present it on a webpage.



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

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

Figure 1: Sensor Networks <-> WWW Integration

Digital Object Identifier: 10.4108/ICST.WICON2008.4787
http://dx.doi.org/10.4108/ICST.WICON2008.4787

However, due to the limited hardware resources of sensor nodes, 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. 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.

In summary, a good XML data binding solution for sensor networks has to fulfill the following criteria:

- *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.

In this paper, we present such an XML data binding technique for XML processing in sensor networks. This technique is based on the usage of XML template objects. XML template objects are defined as XML data structures that combine similar XML structures and further compress them while still allowing dynamic processing of the XML contents. By statically analyzing the usage and definition of XML documents within sensor network programming, XML template objects can be automatically defined without letting the developer handle the complex XML compression and data binding task.

In summary, our contributions in this paper are:

- $XOBE_{SensorNetwork}$*:* We propose an XML programming environment to simplify the XML integration in sensor network programming. $XOBE_{SensorNetwork}$ (**X**ML **Obj**ects for **Sensor Networks**) provides the direct use of XML in a sensor node programming language while ensuring stable and space-, time- and energy-efficient programs.
- *XML template objects:* XML template objects represent a memory and energy efficient way to process XML natively in wireless sensor networks.
- *Implementation:* We present the integration of XML template objects by using the Embedded C programming language.
- *Evaluation:* The efficiency of XML template objects is proven by evaluating typical XML usage scenarios in sensor networks. Furthermore, we show encouraging results based on the XMark benchmark.

The remainder of this paper is organized as follows: Section 2 explains how to integrate the presented XML processing techniques in sensor network programming by using the $XOBE_{SensorNetwork}$ programming environment. Thereby we introduce the proposed approach on memory and energy efficient XML processing within sensor networks. In Section 3, we present the evaluation results of running typical XML usage scenarios and an XMark benchmark based application. We give an overview of related work in Section 4 and conclude this paper and suggest future research directions in Section 5 and 6.

2. XML USAGE IN SENSOR NETWORKS

In this section, we describe our approach on managing XML data within wireless sensor networks. Starting with the given limitations of XML usage on sensor nodes in Section 2.1, we present a complete solution fo processing XML data under the present energy and memory restrictions in sensor networks. In detail we describe the integration of XML in sensor network programming in Section 2.2. Integrating XML usage in sensor network programming requires transforming XML into internal data structures that are supported by the sensor network programming languages. In Section 2.3 we introduce our new approach of binding XML data by using XML template structures. The implementation of this new XML data binding technique is presented in Section 2.4.

2.1 Limitations & Goals

XML is verbose, i.e. using the textual representation of XML can increase the space cost for representing the same information up to 400%. Higher space costs slow down computation as more processor cycles are needed for generating, copying and transmitting data over a network. Whenever we need more processor cycles for computation, we also need more energy. Transmitting data is one of the most energy-consuming operations in wireless sensor networks. Thus, the biggest waste of energy is whenever more data than necessary is transmitted. Therefore, saving space costs for used data leads to increasing performance and saving energy, which extends the lifetime of each single sensor node and the whole sensor network. Table 1 presents an overview over existing sensor node platforms. In summary, the memory capacity of current sensor nodes is quite limited while the energy consumption hits its peak at transmission operations, as described in [16]. These hardware resource limitations therefore conflict with the general verbosity of XML. 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 serious for integrating XML in sensor network programming. While recent sensor network applications use only simple data structures, e.g. only lists of sensor values, using XML will require consolidated knowledge on how to represent XML with the language constructs of sensor node programming languages like Embedded C [3]. Hence, besides solving the verbosity problem of XML, the XML programming language integration should be transparent and usable without any consolidated knowledge. The motivation of this paper is to show a flexible API providing XML storage and XML navigation methods that are the key feature for future web service solutions integrated in sensor networks.

2.2 Our XML Programming Environment

In this subsection, we propose our XML programming environment to simplify the XML integration in sensor network programming, $XOBE_{SensorNetwork}$ [11, 24]. It provides the direct use of XML in a sensor node programming language while ensuring stable and space-, time- and energy-efficient programs. This is done by integrating XML con-

Digital Object Identifier: 10.4108/ICST.WICON2008.4787
http://dx.doi.org/10.4108/ICST.WICON2008.4787

Table 1: An Overview of Common Sensor Nodes

Name	pacemate	BTnode	Tmote Sky (Telos B)	MICAz Mote	IMote	XYZ Sensor Node
Producer	Uni Luebeck	ETH Zuerich	Moteiv	Crossbow	Intel	Yale
RAM [kB]	32	64+180	10	4	64	32
Flash [kB]	256	128	48	128	512	256

structs directly into the used programming language Embedded C [3]. In this section, we also outline the advantages of our approach, which are transparency to the user and guaranteed valid XML expressions which enable stable programs.

2.2.1 Transparency

Programming sensor nodes may be tedious when you have to deal with different exchange formats, even with XML. Providing internal send and receive methods might simplify the handling of messages but still does not ensure full transparency of communication to the developer. With $XOBE_{SensorNetwork}$ we ensure transparency by integrating the possibility of directly using XML within the programming language. In Figure 1 we show an example of programming a node using $XOBE_{SensorNetwork}$. The PC is asking for the status of the sensor network and is further processing the result using XML. The sensor node is programmed to gather its status information using XML, e.g. read the temperature sensor value, and send it as a result to the PC. $XOBE_{SensorNetwork}$ integrates XML into Embedded C, which is a wide spread programming language for sensor nodes based on AVR controllers [3]. By using the $XOBE_{SensorNetwork}$ precompiler, the developer does not have to handle XML within the Embedded C Code manually. Using the keyword *xml* at variable declaration defines using an XML variable. From now on, plain XML can be assigned to the XML variable, which is shown in Figure 1($\star$). The $XOBE_{SensorNetwork}$ precompiler automatically transforms the assignment into correct C code using our efficiency optimized transformation rules.

2.2.2 Stable Programs

Besides providing a transparent use of XML within programs, $XOBE_{SensorNetwork}$ code ensures more stable programs. If a program generates an XML document, it is often necessary that the document is not only well-formed but also valid. This means it conforms to a schema, generally given as a DTD or an XML Schema document. To verify that a given program produces a valid XML document it is normally necessary to perform comprehensive tests at runtime. This process is called dynamic type checking. $XOBE_{SensorNetwork}$ instead offers the possibility of static type checking: by a program analysis at compile time it can be decided if the given program produces a valid XML document or not. If e. g. in the previous code example the right hand side of the assignment does not conform to the type of `status`, the static type checking procedure reports an error.

In this way even the return values of queries, e. g. formulated in XQuery, 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. In the next subsection, we describe our XML template object based approach.

2.3 XML Template Objects

Section 2.1 points out that there are limitations on representing a high amount of XML data on sensor nodes. However, if XML is used as a centralized data format, the size of data over long time running applications may become very high. For this reason, XML data binding techniques without any compression technique are not suitable for representing XML in sensor node programs. An example for extensive usage of XML in sensor network programming is shown in Listing 1

Listing 1: $XOBE_{SensorNetwork}$ program processing a rapidly growing XML object structure

```
xml<btsysinfo> sensor;
sensor = <btsysinfo>
                <timestamp>0</timestamp>
                <bat>10</bat>
              </btsysinfo>;
for (int i=1; i<=n; i++) {
  int time = getTime();
  int battery = getBat();

  sensor = <btsysinfo>
                <timestamp>{time}</
                    timestamp>
                <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 at transfering 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.

Beside the string representation of XML, there have been other XML data binding frameworks introduced. For C based sensor network programming, autoXML [13] offers XML to C transformation methods. However, as is the rule for XML data binding frameworks, autoXML generates structs and functions for every XML element according to a given schema file. If the schema file is big, much memory is spent for this transformation method including the function library overhead.

To save memory and hence be able to use XML within sensor networks, we propose the new XML template object (XTO) framework. This data binding framework is based on the idea of splitting up XML fragments into dynamic and static parts, which can be compressed individually. In a program like in Listing 1, resulting XML documents consist mostly of repeating structures. These repeating structures are defined as static parts, because they do not change over iterations. All other parts are supposed to be dynamic and can be processed separately. The only consideration that needs to be made is to link static and dynamic parts. As a result for Listing 1, we define the XML template objects by the static part consisting of the tags <*btsysinfo*>, <*timestamp*> and <*bat*>. The dynamic part is linked by placing insertion markers for the integer variables *time*, *battery* and the XML element variable *btnodeinfo*. The resulting XTO is shown in Listing 2.

Listing 2: XML Template Object with Three Insertion Places @1, @2 and @3

```
<btsysinfo>
  <timestamp>@1</timestamp>
  <bat>@2</bat>
  @3
</btsysinfo>
```

For inverse transformation from XTO to XML it suffices to store static parts one time globally while only for dynamic parts memory is allocated every time they are used during runtime.

2.3.1 Representing XML Template Objects

While the idea of XTO promises good compression results, there is still the question how to represent XTO in the C programming language. The obvious thing to do would be to use strings and char arrays respectively as they are the natural representation of XML. Insertion markers are represented by using special characters. Generic structs can be introduced for XTO information to save further memory. This result is shown in Listing 3.

Listing 3: Generic String Representation of XML Templates

```
struct _XmlObject {
        char * template;
        void ** dE;
};
```

In the variable *dE*, the dynamic part of the XTO is saved, whereby the static part is stored in the char array variable *template*. However, by accessing the dynamic part during runtime, a typecast to the original type of the dynamic variable has to be done, e.g. to integer, string or another XTO. 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 everytime 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 tree structures for representing the static template instead of string representation because:

- subsequent parsing of the template for accessing dynamic elements is avoided
- accessing dynamic elements becomes faster
- dynamic element types can be stored within the template
- evaluating XML queries is simplified and more energy efficient

In Listing 4, we show the resulting C structs that are the basis for further enhancements on XTO.

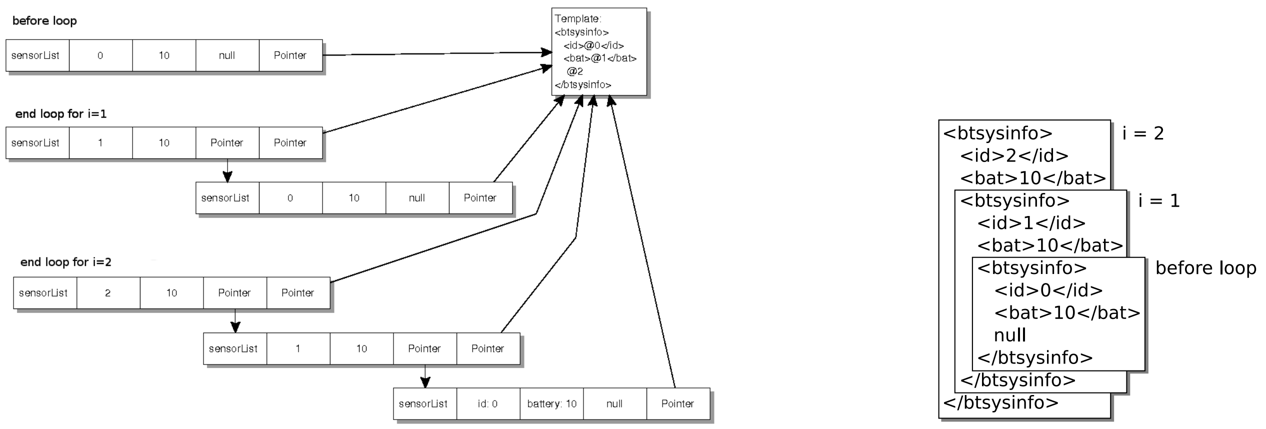
Listing 4: Generic XML Template Tree Structure Representation in Embedded C

```
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 anzel; // #elements
        int anzAtt; // #attributes
};

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

In summary, Figure 2(a) shows the resulting XML template objects for the given example in Listing 1 after two iterations. The corresponding XML document is shown in Figure 2(b).

2.3.2 Optimizing XML Template Objects

XML template objects 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 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



(a) XML Template Object (b) XML

Figure 2: Example for growing XML Template Objects

often has good 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 5 shows this simple but effective approach by an example.

Listing 5: XML Query Rewriting after Identifier Optimization

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

XML Template for <btsysinfo>:
<1>
        <2>010006</2>
        <3>15</3>
        <1></1>
</1>

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

2.4 Implementation

After presenting the idea of XML template objects and proposing techniques on how to represent the structure of XML templates, this section shows how information of XML documents is processed. In detail, we discuss the generation of XML template objects for concrete XML fragments and documents respectively. In this process the static parts of the XML fragments are collected and represented using the XML templates technique while the dynamic parts are stored dynamically like shown in Figure 2(a). This section finally ends up with an introduction on how to process XML queries on XML template objects.

2.4.1 Generating XML Templates

In this paper we consider using XML during programming, e.g. writing sensor network applications that make extensive use of a stored XML document, like saving sensor values in it. 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. Listing 1 gives an example for an XML assignment and the integration of those assignments in general programming is discussed later in Section 2. In this section, the only assumption is that there are variables refering to XML fragments and documents respectively.

The generation of XML template objects always depends on where and what XML fragments are used in the program code. To avoid generating unnecessary XML templates we look for XML assignments during compile time to generally generate an XML template object consisting of the static XML template and the dynamic parts of the assignments. This method generates a XML template for every assignment and hence the instance of the XML template object 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 XML template object 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 XML template object 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 beeing represented by empty pointers, which actually also need to be stored and hence require memory space. In Listing 6, we show an extension of the schema file Listing 1 is based on. During compile time it cannot be determined how many *btsysinfo* elements will be inserted in each iteration.

Listing 6: Extended DTD for BTstatus Example

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

For this purpose we check for changing XML structures during runtime and adapt XML templates if changes are detected, e.g. the size of the *btsysinfo* list is changed. To avoid destroying cross referenced XML templates, we log the number of assignments to each XML template. Only if there is no other assignment the XML template is changed. Otherwise, we need to create a new XML template for the actual assignment that may use the old XML template for optimization reasons.

2.4.2 XML to XML Template Object Transformation

In the last subsection, we discussed when to create XML templates. In this subsection, we will present how to create XML template objects consisting of XML templates and the dynamic part of the assignment. We refer again to the example given in Listing 1. For simplification, we will leave the loop construct out. The reassignment has been discussed in the previous subsection and will be summarized in Subsection 2.4.3. We assume that all generic template structures from Listing 4 have been generated.

For the transformation of an assignment as shown in Listing 1 first the definition needs to be processed. We therefore define a new XML template object *sensor* as follows:

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

The remaining part of the assignment is the right part. This part represents the actual XML fragment and is then parsed for two times. 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 XML template object. For both parts, we use an internal representation to let the XML template object be initialized by a central constructor during runtime to save unnecessary program code. The resulting code in an abbreviated form is shown in Listing 7.

Listing 7: 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.

2.4.3 Summary of the XML Template Object Transformation Process

In the last two subsections we discussed how and when to generate XML template objects. Beyond the scope of initial instantiation of new XML template objects, the reassignment of these types of variables will result into an XML template update process.

We now propose a complete transformation process for transformating XML to XML template objects. Figure 3 shows the complete process.

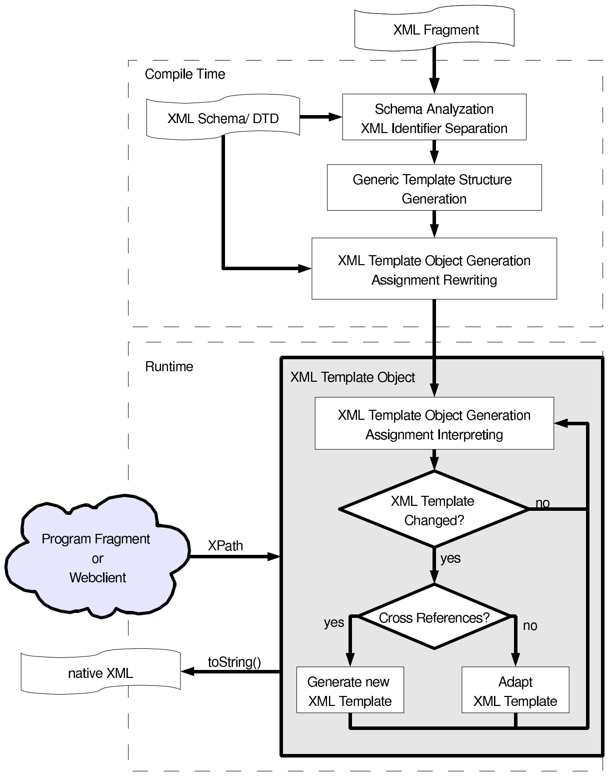


Figure 3: XML to XML Template Object Transformation Process

Starting with the assigned XML Fragment and a given XML schema file, the Schema Analyzer separates XML identifiers as described in Section 2.3.2. The next step is to generate the generic XML template structs, which have been introduced in Section 2.3.1. This includes the integration of further header files including the XML template object framework. For all XML assignments the transformator will then define XML template objects and rewrite the assignment as shown in Section 2.4.2. The result of these compile time processes is a compilable Embedded C program.

During runtime, blocks using the XML template objects will be reached and the constructor of them is initiated like described in Section 2.4.2. This process is called *Assignment Interpreting*, because the constructor interprets the internal representations of XML templates and dynamic XML parts before instantiating the XML template object. As also mentioned in Section 2.4.2, 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 XML template objects 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.

2.4.4 Evaluating XPath Queries on XML Template Objects

In this last part of Section 2 we discuss the evaluation of XPath queries. Evaluating XML Queries dynamically is a key feature for an XML data binding framework. As shown in Section 2.3.1 representing XML with strings lacks of this functionality without any further parsing step. With the usage of XML templates based on tree structures we are able to navigate XML template objects in order to process XPath queries.

To evaluate a XPath query first the root XML template object is chosen as the first context node. For each XPath navigation step new nodes are found. In order to collect nodes on the navigation path we have implemented two structures *node* and *nodeset*. Each node is a pointer to a specific location in the XML template object. This can be a part in the XML template or a dynamical value. We therefore can directly access parts of XML template object if they are in the resultset after evaluating the XPath query.

Until now we have implemented all important axis and the most important predicate functions of XPath. We have collected these functions in a library that will be used by our future continuous XML query engine dynamically within the sensor network. The implemented axis are

- *self, child, descendant, descendant-or-self, parent, ancestor, ancestor-or-self, following-sibling, preceding-sibling, following, preceding, attribute*

In extension we have implemented most abbreviations in common use of XPath.

3. EVALUATION

In this section we present results of different representative sensor network applications using XML natively on sensor nodes. As proposed in the introduction and in Section 2.1, 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 for the criteria as already shown in the introduction:

- *Memory Efficiency:* An XML representation must be memory efficient. For a fixed amount of allocated memory it should represent a high amount of native XML data. Besides, less allocated memory means 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.

- *Processability:* Using XML in wireless sensor networks requires dynamic accessability. XML data will change over the time, e.g. sensor data will be updated, and whole XML trees 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.

The evaluation tests are based on applications which make extensive use of XML. In these applications XML documents grow rapidly over the time by dynamically linking. We compare the following transformation methods for their capability of representing XML on sensor nodes:

- *XML string representation:* Representing XML in a native form as a string has been discussed in Section 2.3. 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:* 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 for evaluation.

- *autoXML:* As presented in Section 2.3 autoXML [13] is a state-of-the-art C data binding framework. However, no compression is focussed so that it is to expect that this approach is not memory efficient. Navigating XML for answering XPath queries would also demand an implemented framework on top of autoXML.

- *XML Template Objects:* As proposed throughout this paper XML template objects are a data binding approach with a high processability. Furthermore, the memory efficiency has been the main development goal to reach the limited hardware restrictions of sensor nodes.

For evaluation we use the BTnode sensor node platform [32] as target platform. The criteria for this sensor node environment are that programs are limited to 128 kByte of program memory and 64 kByte in total of main memory. The BTnode sensor node platform is based on the Atmel ATmega128l Controller and therefore represents a high amount of other sensor node platforms like the Crossbow MicaZ sensor node. 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 [2].

3.1 BTsys Benchmark

We firstly tested a typical scenario on gathering status information in the sensor network. Therefore, each sensor node runs an extended version of the application shown in Listing 1. This application has a high demand on using compression techniques, because the XML fragment grows with every iteration. The results are shown in Figure 4.

As a result, XML template objects are the most memory efficient way to represent XML within this application.

Digital Object Identifier: 10.4108/ICST.WICON2008.4787
http://dx.doi.org/10.4108/ICST.WICON2008.4787

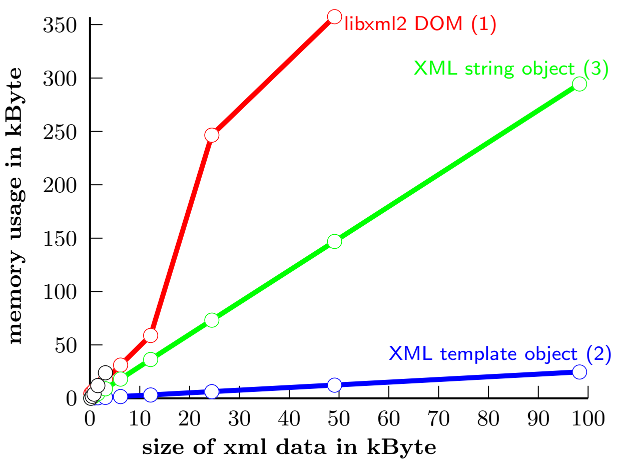


Figure 4: XML Memory Usage for Different Methods of Data Binding

By using XML template structures 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 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, which are shown in Figure 5. 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

We were then interested in the runtime efficiency. We tested the string representation and the XML template objects, as both beeing the most memory efficient approaches, for the amount of needed processor cycles over the iterations. The results are shown in Listing 6. As the string approach needs less cycles for the first phase the amount of processor cycles was also steeply increasing. The runtime duration of the XML template object program was increasing linearly.

We did further test on how elements and attributes are represented with XML template objects by extending the program with a static and a dynamic attribute. As a result attributes demand less memory than elements, which can be explained by the pointer using implementation of element and attribute relations.

3.2 XMark Benchmark

To verify our results we use a sensor network application based on the XMark Benchmark data generator. 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 7.

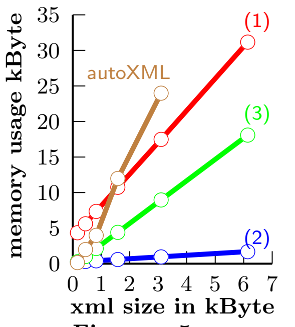


Figure 5: autoXML Memory Usage

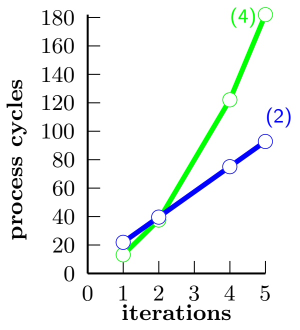


Figure 6: Runtime Efficiency

As a result, the XML template object approach also outperforms the competitors.

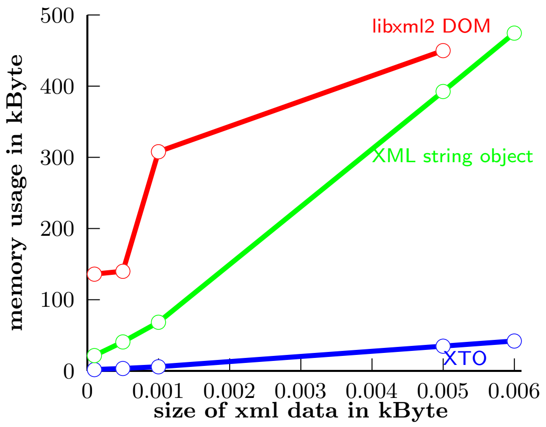


Figure 7: XMark XML Memory Usage

3.3 XML Template Object Program Memory Demand

As a final result, Table 2 shows the program memory demand of the evaluated representations. As described in Section 3, *libXML2 DOM* and *autoXML* are not listed because they excess the memory restrictions of every existing sensor node platform. The remaining two possibilities are the string representation and the XML template objects. Both representation techniques have been also tested for the recommended compiler optimization flag (space optimization: -Os). In result, the string representation framework has a low program memory demand. The more memory efficient XML template object framework has a higher program memory demand, needing 19.8% of the sensor node program memory. This demand becomes even higher, if the evaluation of XPath queries needs to be supported. However, the demand is still under the restrictions and therefore XML template objects are fully applicable on today's sensor node platforms.

3.4 Summarization of Results

Section 3 shows that common DOM parsers and other XML data binding tools like autoXML do not compress XML in a sufficient way. The internal representation size is rather many times higher than the size of the native XML document. However, in this paper proposed XML template objects achieve a sufficient degree of compression and there-

Digital Object Identifier: 10.4108/ICST.WICON2008.4787
http://dx.doi.org/10.4108/ICST.WICON2008.4787

Table 2: Program Memory Demand

XML representation	program
String	12032 (9.2% Full)
String (-Os)	8844 (6.7% Full)
XTO	47866 (36.5% Full)
XTO (-Os) including XPath	32020 (24.4% Full)
XTO (-Os) without XPath	25906 (19.8% Full)

fore are applicable for integrating XML data in sensor network programming. Furthermore, in Section 3.1 we show that representing attributes with template objects has even lower space costs than representing elements. By using XML template objects the processing time for generating linear growing XML documents becomes independent of the actual size of the XML object. This shows that XML template objects are memory efficient and furthermore energy efficient and that the proposed approach is suitable for generating huge XML documents for collecting data on sensor nodes. While Section 3.1 also shows that extensive XML template object usage within application results into a remarkable amount of required program memory, this amount is still conformable with the program memory restrictions of sensor nodes.

4. RELATED WORK

In this section we present approaches in the fields of XML data binding and XML compression. Afterwards, we outline different approaches of the possible application areas: data storage and communication in conjunction with service-oriented architectures in sensor networks.

Representing XML within programming languages is often refered as XML data binding. While there are many existing object-oriented XML data binding frameworks for Java and C++ [22, 9, 1], these frameworks are not applicable for sensor network programming. Beside the choice of language they do not compress XML in a sufficient way to overcome the hardware ressource limitations in sensor networks. For the programming language C there exist one XML data binding framework: autoXML [13]. However as to Section 3, autoXML has also not shown to be applicable for sensor network applications.

The motivation for compression of XML data is its verbosity. Simple compression techniques like zip [10] require the whole XML document before compressing. This is often not possible, especially for dynamicaly growing XML documents. Besides, dynamical processing of XML, e.g. answering queries, may not be possible without an expensive decompression step. Special compressors for XML data can be divided into two groups: those which produce queryable compressed XML data [21, 19, 5] and those which do not [29, 15, 6]. A fundamental problem is that the footprint of these XML compressors and simple compressors is many times higher than the available memory on sensor nodes. So these techniques cannot be applied until the footprints are reduced or the available memory increases. The current status in the development of common sensor nodes is shown in Table 1.

Published approaches for data storage in sensor networks like TinyDB [16], Cougar [31], DSN [7] and SwissQM [20] abstain from the usage of XML. In TinyDB, data is represented as a table with one column for each attribute which is available in the network. Because of independancy between programming and query language in SwissQM no data model is defined.

XML usage is the key feature for using web service techniques in sensor networks and during sensor node programming. Furthermore, the extension of the service oriented paradigm to sensor networks will also provide better conectivity 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 have 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 [25, 23, 8]. In the second category, single sensor nodes provide services which can be composed to more complex web services using gateways for communication only [18, 14, 4]. 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 [18, 14]. However, Delicato et al. [4] propose an XML based communication within the sensor network, but this approach has not been realized on sensor nodes, yet.

5. CONCLUSIONS

In this paper, we presented the XML data binding technique XML template objects to motivate XML usage within sensor networks.

We then proposed the approach $XOBE_{SensorNetwork}$ to integrate XML into sensor network programming.

$XOBE_{SensorNetwork}$ allows to transparently use XML, hiding the XML template object transformation task, and supports static type checking for the detection of failures at compile time, which leads to more stable programs. While the energy and memory limitations in sensor networks still remain, XML template objects allow to compress XML data sufficiently to bridge this technology gap. Furthermore the XML data remains dynamically processable and XML queries can be evaluated directly on XML template objects. By using XML and XML queries natively within sensor networks we propose a solution for supporting heterogeneous networks including the usage of different brands of sensor nodes within a sensor network. Furthermore using highly exchangeable data formats improves and simplifies the connectivity to sensor networks. This supports the integration in heterogeneous networks like the WWW and the usage of service oriented techniques within sensor networks to enhance sensor network engineering.

6. FUTURE WORK

For future work we consider supporting XML data definition during runtime from outside the sensor network by using a DDL. Furthermore we will extend $XOBE_{SensorNetwork}$ by integrating user defined XML template types. These XML template types may optimized the memory efficiency by letting the user determine how XML documents grow under a given schema file.

Beside integrating a dynamic XML query engine, we work on extending existing XML query languages to support continuous queries. Continuous queries are a key feature in sensor network query processing and need therefore to be supported by an XML query engine.

Using XML in sensor networks will not only optimize the way of getting information from the network. Moreover, programming sensor networks remains too complex with existing programming languages and techniques. We therefore are working on applying and extending the service-oriented paradigm to sensor network application engineering. Therefore, XML is a key feature to make programming sensor network applications easier by using web service techniques.

7. REFERENCES

[1] Altova. Xmlspy - xml editor. http://www.altova.com/products/xmlspy/xml_editor.html.

[2] Atmel. Avr studio 4.13. www.atmel.com/avrstudio/.

[3] R. H. Barnett, S. Cox, and L. O'Cull. Embedded c programming and the atmel avr, 2006.

[4] J. Blumenthal, M. Handy, F. Golatowski, M. Haase, and D. Timmermann. Wireless sensor networks - new challenges in software engineering. 2003.

[5] 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.

[6] J. Cheney. Compressing xml with multiplexed hierarchical ppm models. *dcc*, 00:0163, 2001.

[7] 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.

[8] 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.

[9] E. Group. The castor project. http://www.castor.org/.

[10] Gzip. http://www.gzip.org/.

[11] N. Hoeller, C. Reinke, S. Groppe, and V. Linnemann. Xobe Sensor Networks: Integrating XML in sensor network programming. In *Proceedings of the 5th International Conference on Networked Sensing Systems (INSS 2008)*, June 17 - 19 2008.

[12] 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.

[13] J. Kent. Autoxml. http://hgwdev.cse.ucsc.edu/ kent/src/.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] R. Mueller, G. Alonso, and D. Kossmann. Swissqm: Next generation data processing in sensor networks. In *CIDR*, pages 1–9. www.crdrdb.org, 2007.

[21] 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.

[22] E. Ort and B. Mehta. Java architecture for xml binding (jaxb). http://java.sun.com/developer/earlyAccess/xml/jaxb/.

[23] 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.

[24] H. Schuhart, B. C. Hammerschmidt, and V. Linnemann. Integrating Statically Typechecked XML Data Technologies Into Pure Java Architectures. In *Proceedings of the ISCA 15th International Conference on Software Engineering and Data Engineering (SEDE-2006)*, pages 217–222, Los Angeles, California, USA, July 6-8, 2006. ISCA.

[25] 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.

[26] W3C. Xml path language (xpath) 2.0. http://www.w3.org/TR/xpath20/.

[27] W3C. Xquery 1.0: An xml query language. http://www.w3.org/TR/xquery/.

[28] W3C. Xsl transformations (xslt) version 1.0. http://www.w3.org/TR/xslt.

[29] C. Werner, C. Buschmann, Y. Brandt, and S. Fischer. Compressing soap messages by using pushdown automata. *icws*, 0:19–28, 2006.

[30] 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.

[31] Y. Yao and J. Gehrke. The cougar approach to in-network query processing in sensor networks. *SIGMOD Rec.*, 31(3):9–18, 2002.

[32] B. P. . E. Zurich. Btnodes - a distributed environment for prototyping ad hoc networks. http://www.btnode.ethz.ch/.