Shared Information Spaces for Small Devices: The SWARMS Software Concept

Jochen Koberstein, Norbert Luttenberger
Dept. for Computer Science
and Applied Mathematics
Christan-Albrechts-University in Kiel
Christian-Albrechts-Platz 4
D-24098 Kiel
{jko|nl}@informatik.uni-kiel.de

Carsten Buschmann, Stefan Fischer
Institute of Operating Systems
and Computer Networks
Technical University at Braunschweig
Mühlenpfordtstr. 23
D-38106 Braunschweig
{buschmann|fischer}@ibr.cs.tu-bs.de

1 Introduction

In sensor networks [ASSC02]–and also other environments with small devices–the classical client/server co-operation paradigm does no longer seem to be adequate for a number of reasons: (1) Sensor nodes communicate via unreliable wireless media; thus clients cannot rely on the accessibility of "their" servers. (2) Typical request/response protocols for client/server co-operation are built upon point-to-point message exchange. The communication in wireless sensor networks should exploit the broadcast nature of the wireless medium which inherently proposes to let more than one neighbour receive the same message. (3) For mobile sensor networks it cannot be assumed that services offered by a node remain accessible for a certain amount of time. Complex service discovery and routing protocols cause a significant increase in bandwidth and power consumption hardly acceptable for resource-constraint sensor networks.

Therefore we propose a different co-operation concept namely the distributed virtual Shared Information Space (dvSIS). In the following section we elaborate on this concept. In section 3 we introduce related basic technologies. These re-occur in an adapted application development process which is shown in section 4.

2 The dvSIS Co-operation Paradigm

We assume the nodes of a sensor network to behave like a swarm: its members follow a common operation goal requiring their co-operation which is based upon–at least partial–common knowledge about the operational environment and swarm state. This view on the co-operation of the nodes of a sensor network is clearly distinct from e.g. [IGE00, SPMC04] who assume a "star-shaped" co-operation with a number of sensor nodes reporting to a central data sink. Our view is adequate for sensor networks with in-network data pre-processing and for sensor/actor networks where the execution of actions must respect not only local but regional knowledge.

Shared information is usually stored in a common place which is accessible to all participants. For reasons given in the introduction this place should not be a physical, centralized place–we advocate a distributed virtual place. To share information a swarm establishes a distributed virtual Shared Information Space. Every swarm node holds a local instance of the dvSIS, which may be incomplete, partially obsolete, or inconsistent with the local instances of other nodes. By giving up hard requirements on consistency and completeness the dvSIS does not depend on any centralized infrastructure like an information hub. The dvSIS is the union of all its local instances; it exists as an abstraction only.

The dvSIS contains information about the swarm state and configuration as well as on environmental observations. It consists of semi-structured, self-describing information elements which are augmented with syntactic and semantic meta-information describing the context of data acquisition (such as position and reliability) or the data itself (such as the level of aggregation or scope). To make a sensor node's contribution to the dvSIS available to all other sensor nodes the information is flooded in the network by broadcast message transfer. To avoid redundant data transmission the nodes adhere to a content-based flooding control scheme according to the XCast abstract model [KRL04].

Against all concerns with respect to related overhead we decided to model the dvSIS as an XML-coded document. This language technology offers a rich wealth of modelling and processing features among which the capability to define formal grammars for documents is not the least. It enables validation of documents and support for information centric application development. In addition the semi-structured and self-describing nature of XML documents facilitates information processing like merging documents or keeping information up-to-date. Following this approach the dvSIS structure formally is described by an XML Schema [BM01]. A dvSIS XML Schema instance is obviously application specific; it covers three kinds of XML instance documents: (1) the union of all information components being shared among the swarm nodes, (2) the information being held by a single swarm member ("local views on dvSIS") and (3) information elements to be sent as messages ("message instances").

3 XML processing on small devices

A single XML-coded sensor reading may consist of 100 to 200 Bytes depending on the contextual information supplied and tag sizes. So how does this concur with small resource constraint devices–nowadays with about 2 or 4 Kbytes of RAM?

In traditional approaches for handling XML instances two main streams have emerged. The first one is based upon DOM [HHW$^+$02] and maps the documents complete tree hierarchy into a memory representation before the application works on it. This is–as motivated above–not a viable way for resource constraint devices.

The alternative approach is to report parsing events related to lexical units of an XML document to the application by invoking predefined call-back functions. The application thus processes received data without building the document's complete tree. Well known for this processing model is the Simple API for XML (SAX). For message decoding this strategy leads to considerable memory savings, since messages can be transformed on-the-fly from an XML into a local representation, and possibly merged with locally available information; irrelevant information can even be dropped. Unfortunately the SAX provided call-backs are un-typed which implies two drawbacks: (1) the application has to figure out the type of the received content by itself and (2) the schema must be made accessible during runtime for validation purposes.

Enhancing the event based approach towards a typed event based approach seems to be the most promising solution for small devices: (1) the event based approach allows condensing space consuming XML coded data into memory saving internal representations at the earliest possible moment (2) the adaptation of the API to an application specific XML schema makes code shorter and faster while avoiding the need for accessing the schema at runtime, and (3) validation may inherently be provided.

4 Application Software Development Process

Following the above presented conceptual ideas the so-called <<ASTAX framework has been developed (read: CCASTAX). This framework has two main components: STAX/g and STAX/p, where STAX stands for "Simple Typed API for XML". STAX/g is a generator which (1) generates code for a validating parser (STAX/p) and (2) interfaces for the typed call-backs based upon a given XML schema). STAX/p is built upon a new class of automata, called Cardinality Constraint Automata (CCA) which are in some depth explained in [RL02]. At runtime, the parser "links" parsing events to the typed call-back functions generated by STAX/g. These interfaces form an event-based API that maps the defined XML type hierarchy onto programming language constructs (explained by an example below). These handlers need to be "filled in" by the application developer, as well as application specific XCast components. An overview is given in figure 1.

Built upon the availability of the <<ASTAX framework, a straightforward software development process for dvSIS-based application software can be defined. It comprises a sequence of three initial steps: The first and constituting step is the specification of the dvSIS, formalized in an XML Schema that defines type and structure of all information items contained in the dvSIS and all related message formats. In the second step, this schema is fed into STAX/g yielding the validating parser STAX/p plus its controlling CCA automaton, and a collection of typed call-back interfaces. In the third step, the application programmer implements and tests the call-backs. This step is followed–as usual–by the deployment, operation, and maintenance phases, adhering to common software development methodologies.

The <<ASTAX framework supports generation of Java and C Code. Generating C code with STAX/g targets typical sensor nodes. The generated C code is splitted in implementation and header files (.c- and .h-files) which contain function headers and empty function

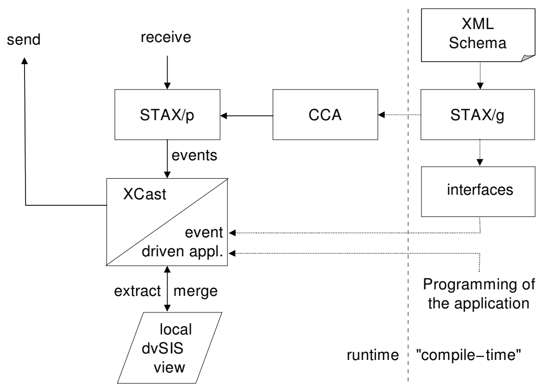

Figure 1: The information centric application development process

bodies. These functions are invoked as call-backs by STAX/p when encountering lexical units (e.g. XML start tags, end tags, attributes and character data) which validate against the given schema. For each XML element and attribute defined in the XML Schema a seperate corresponding .c-file and a .h-file is generated. A .c-file for a complexType comprises so-called `create` and `add` functions for its child elements and its attributes. A .c-file for a simpleType or an attribute comprises only a so-called `setContent` function. For the document's root element a .c-file called Start.c is generated by default which contains the appropriate functions. Additionally so-called `contains` and `merge` call-backs can be generated for complexTypes for information aggregation [LRK04]. Table 1 gives an example: on its left side a fragment of an XML document encoding a sensor reading is depicted. The sequence of call-back invocations when parsing and validating this document is shown on the right side of the table.

<dvSIS>	→	Start.c:	create_dvSIS()
<temp scale="Celsius">	→	dvSIS.c:	create_temp ()
		temp.c:	create_scale()
		scale.c:	setContent("Celsius")
		temp.c:	add_scale(..)
32	→	temp.c:	setContent(32)
</temp>	→	dvSIS.c:	contains_temp(..)
		dvSIS.c:	add_temp(..) *or* merge_temp(..)
</dvSIS>	→	Start.c:	add_dvSIS(..)

Table 1: Sequence of function calls generated when parsing an instance document.

The C code generated by STAX/g may be compiled for a wide variety of platforms including e.g. the Texas Instruments MSP430 processor which is for example used on the Embedded Sensor Boards [ESB] developed at the FU Berlin, Germany. The MSP430 provides UARTs which may be used for wireless communication tasks. Receiving a byte triggers an interrupt which leads to the execution of the corresponding interrupt service routine (ISR). To continuously receive data this routine must return in time. Since STAX/p directly calls application functions STAX/p is decoupled from the ISR by an event queue. The ISR recognizes lexical units only and schedules related events for further processing, i.e. validation and application execution.

For a first proof of concept we have written an XML Schema corresponding to the set of sensors provided by the above mentioned Embedded Sensor Boards. Compiling the generated validating parser in conjunction with the ISR leads to a program code size of about 8 Kbytes. Obviously this heavily depends on the schema's complexity.

5 Outlook

Next steps are the development of a number of demo applications. Our plan is also to study more space saving message encodings delivering similar language support as XML.

References

[ASSC02] Akyildiz, I., Su, S., Sankarasubramanian, Y., and Cayirci, E.: Wireless Sensor Networks; A Survey. *Computer Networks*. 38(4):393–422. March 2002.

[BM01] Biron, P. V. and Malhotra, A. XML Schema Part 2: Datatypes. `http://www.w3.org/TR/xmlschema-2/`. May 2001.

[ESB] Website of the Embedded Sensor Board (ESB). `http://www.scatterweb.com`.

[HHW+02] Hors, A. L., H'egaret, P. L., Wood, L., Nicol, G., Robie, J., Champion, M., and Byrne, S. Document Object Model (DOM) Level 3 Core Specification. October 2002.

[IGE00] Intanagonwiwat, C., Govindan, R., and Estrin, D.: Directed diffusion: a scalable and robust communication paradigm for sensor networks. In: *Proceedings of the sixth annual international conference on Mobile computing and networking*. ACM. 2000.

[KRL04] Koberstein, J., Reuter, F., and Luttenberger, N.: The XCast Approach for Content-based Flooding Control in Distributed Virtual Shared Information Spaces—Design and Evaluation. In: *Springer Lecture Notes in Computer Science 2920*. First European Workshop on Wireless Sensor Networks. S. 188–203. January 2004.

[LRK04] Luttenberger, N., Reuter, F., and Koberstein, J.: XML Language Binding Support for Pervasive Communication in Distributed Virtual Shared Information Spaces. In: *Second IEEE International Conference on Pervasive Computing and Communication*. S. 181–186. March 2004. Workshop for Middleware Support for Pervasive Computing.

[RL02] Reuter, F. and Luttenberger, N. Cardinality Constraint Automata: A core technology for Efficient XML Schema-aware Parsers. `http://www.swarms.de/publications/cca.pdf`. 2002.

[SPMC04] Szewczyk, R., Polastre, J., Mainwaring, A., and Culler, D.: Lessons from a Sensor Network Expedition. In: *Springer Lecture Notes in Computer Science 2920*. First European Workshop on Wireless Sensor Networks. S. 307–322. January 2004.