# Ein Ansatz zur effizienten Softwareentwicklung für drahtlose Sensornetze auf Basis von Shared Information Spaces

Carsten Buschmann, Stefan Fischer

Institut für Betriebssysteme und Rechnerverbund
Technische Universität Braunschweig
Mühlenpfordtstr. 23, D-38106 Braunschweig
{buschmann|fischer}@ibr.cs.tu-bs.de

Jochen Koberstein, Norbert Luttenberger

Institut f. Informatik und praktische Mathematik
Christan-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4, D-24098 Kiel
{jko|nl}@informatik.uni-kiel.de

## 1 Einleitung

In drahtlosen Sensornetzen [ASSC02] ebenso wie in anderen Umgebungen scheint das Client/Server Kooperationsparadigma aus einer Reihe von Gründen nicht mehr länger angemessen: (1) Da solche Sensorknoten über ein unzuverlässiges drahtloses Medium kommunizieren, können sich die Clients nicht auf die Verfügbarkeit von Server verlassen. (2) Die typischen Request/Response-Protokolle basieren auf einem Punkt-zu-Punkt-Nachrichtenaustausch. Die Kommunikation in Sensornetzen sollte hingegen die Broadcastcharacteristik des drahtlosen Mediums ausnutzen und davon Gebrauch machen, dass mehrere Nachbarn eine einmal gesendete Nachricht empfangen können. (3) In mobilen Sensornetzen kann man nicht davon ausgehen, dass von einem Knoten angebotene Dienste über eine bestimmte Zeit verfügbar bleiben. Komplexe Dienstfindungs- und Routingprotokolle verursachen eine signifikante Steigerung des Bandbreiten- und somit Energiebedarfes, der für ressourcenbeschränkte Sensornetze unangemessen erscheint.

Daher schlagen wir ein anderes Kommunikationsparadigma vor, den so genannten *distributed virtual Shared Information Space* (dvSIS). Im folgenden Abschnitt werden wir dieses Konzept näher erläutern. Abschnitt 3 stellt die damit zusammenhängenden Basistechnologien vor. Diese finden sich dann in dem in Abschnitt 4 vorgestellten informationszentrierten Software-Entwicklungsprozess wieder.

## 2 Das dvSIS Kooperationsparadigma

Wir gehen davon aus, dass sich die Knoten eines Sensornetzes wie ein Schwarm verhalten: Seine Mitglieder verfolgen ein gemeinsames Ziel, das eine Kooperation erfordert, die auf einem zumindest teilweise gemeinsamen Wissen über den Zustand von Schwarm und Umwelt basiert. Diese Sichtweise auf die Kooperationsform in Sensornetzen ist deutlich verschieden von derjenigen in [IGE00, SPMC04], wo von einer sternförmigen Zusammenarbeit mit einer Anzahl von Knoten, die einer zentralen Quelle Daten liefern, ausgeht. Unsere Sicht scheint insbesondere für Anwendungen mit Datenverarbeitung und –aggregation im Sensornetz angemessen, bei denen diese Operationen nicht nur von lokalem Wissen abhängen.

Gemeinhin werden gemeinsam genutzte Informationen in einer allen verfügbaren zentralen Instanz gespeichert. Aus den in der Einleitung genannten Gründen sollte diese Instanz jedoch kein physisch zentraler, sondern ein virtueller Ort sein. Um also Informationen gemeinsam zu

nutzen, etabliert der Schwarm einen so genannten *distributed virtual Shared Information Space*. Jedes Schwarmmitglied hält eine lokale Instanz dieses dvSIS, die unvollständig, teilweise obsolet oder inkonsistent mit den lokalen Instanzen anderer Knoten sein kann. Indem wir die Forderungen nach Konsistenz und Vollständigkeit aufweichen, wird eine zentrale Instanz wie ein *information hub* überflüssig. Der dvSIS ergibt sich als die Vereinigung aller lokalen Instanzen; er existiert nur als Abstraktion.

Der dvSIS enthält Informationen über Zustand und Konfiguration des Schwarms genauso wie über Umweltbeobachtungen. Er besteht aus semi-strukturierten, selbstbeschreibenden Informationselementen, die mit syntaktischen und semantischen Metainformationen hinsichtlich des Kontextes der Datenaquisition (wie z.B. Position oder Zuverlässigkeit) oder der Daten selbst (wie z.B. Aggregationsgrad oder Gültigkeitsbereich) angereichert sind. Um die Beiträge eines Sensorknotens zum dvSIS anderen Knoten zugänglich zu machen, werden die Informationen mittels Broadcast im Netzwerk verbreitet. Um dabei den Rahmen der Informationsausbreitung zu steuern und redundante Übertragungen zu verhindern, kommt *content based forwarding* gemäß der XCast-Abstraktion [KRL04] zum Einsatz. Dabei entscheiden applikationsspezifische Filter auf Basis des Inhaltes von Datenpaketen über deren Empfang, Versand und Verarbeitung.

Trotz des Overheads haben wir uns entschieden, den dvSIS als ein XML-kodiertes Dokument zu modellieren. Diese Sprachtechnologie bietet einen reichen Fundus an Modellierungs- und Verarbeitungsfunktionalitäten, vor allem auch die Möglichkeit, formale Grammatiken für Dokumente zu definieren. Sie ermöglicht das Validieren von Dokumenten und unterstützt die informationszentrierte Applikationsentwicklung. Darüber hinaus vereinfacht die semi-strukturierte und selbstbeschreibende Natur von XML Dokumenten Informationsverarbeitungsschritte wie das Verschmelzen oder Aktualisieren von Dokumenten. Diese Modellierung ist dabei auf andere Technologien wie z.B. ASN.1 übertragbar. Dem auf XML Ansatz folgend ist die Struktur des dvSIS formal in einem XML Schema [BM01] definiert. Ein dvSIS XML Schema ist selbstverständlich applikationsspezifisch. Sie beschreibt dabei drei Arten von XML Dokumenten: (1) die von einzelnen Knoten vorgehaltene Information (lokale Sicht auf den dvSIS), (2) die Vereinigung aller lokalen Sichten der Schwarmmitglieder (dvSIS) und (3) die als Nachrichten versandten Informationselemente (Nachrichteninstanzen).

## 3 XML auf kleinen Geräten

Ein einzelner XML-kodierter Sensorwert kann abhängig von der Kontextinformation und Taglängen 100 bis 200 Bytes umfassen. Wie verträgt sich dies mit kleinen ressourcenbeschränkten Geräten, die heute vielleicht über 2 oder 4KBytes RAM verfügen?

Unter den traditionellen Ansätzen zur Handhabung von XML haben sich zwei wesentliche Strömungen heraus kristallisiert. Der erste basiert auf DOM [HHW+02] und bildet den gesamten Dokumentenbaum auf eine Speicherrepräsentation ab, auf der die Anwendung dann arbeitet. Dies ist – wie oben motiviert – für ressourcenbeschränkte Geräte kein geeigneter Weg.

Der alternative Ansatz basiert auf der Generierung von Parserereignissen, die mit den lexikalischen Einheiten des XML Dokuments korrespondieren und den Aufruf von vordefinierten Callback-Funktionen auslösen. Somit können die empfangenen Daten verarbeitet werden, ohne dabei den kompletten Dokumentenbaum aufzubauen. Die Simple API for XML (SAX) ist das bekannteste Beispiel für dieses Modell. Hinsichtlich des Dekodierens von Nachrichten führt diese

Strategie zu einer nicht unbedeutenden Ersparnis an Speicher, da die Nachrichten on-the-fly von XML in eine applikationsinterne Repräsentation umgewandelt und gegebenenfalls mit lokal vorhandenen Daten verschmolzen werden können; irrelevante Informationen können darüber hinaus sofort verworfen werden. Unglücklicherweise sind die SAX Callbacks nicht typsicher, was zwei Nachteile hat: (1) die Applikation muss den Typ der empfangenen Daten selbst bestimmen, und (2) das Schema muss zwecks Validierung zur Laufzeit verfügbar sein.

Für kleine Geräte scheint die Erweiterung dieses ereignisbasierten zu einem getypten ereignisbasierten Ansatz die viel versprechendste Lösung zu sein: (1) der ereignisbasierte Ansatz erlaubt das Eindampfen Speicher verzehrender XML-kodierter Daten in Speicher sparende interne Repräsentationen zum frühest möglichen Zeitpunkt, (2) eine Adaption der API auf ein applikationsspezifisches XML Schema ermöglicht schlanken und schnellen Code, ohne auf das Schema zur Laufzeit zugreifen zu müssen, und (3) das Validieren wird inhärent ermöglicht.

## 4 Prozess einer informationszentrierten Applikationsentwicklung

Den oben präsentierten konzeptionellen Ideen folgend haben wir das so genannte <<ASTAX Framework (sprich: CCASTAX) entwickelt. Es umfasst zwei wesentliche Komponenten: STAX/g und STAX/p, wobei STAX für *Simple Typed API for XML* steht. STAX/g ist ein Generator, der (1) Code für einen validierenden Parser (STAX/p) und (2) Methodenrümpfe für die getypten Callback-Funktionen aus einem gegebenen XML Schema generiert. STAX/p fußt auf einer neuen Klasse von endlichen Automaten, den so genannten Cardinally Constraint Automata (CCA), die in [RL02] detailliert vorgestellt werden. Zur Laufzeit reicht STAX/p Parserereignisse an die getypten Callback-Funktionen, die STAX/g generiert hat, weiter. Diese Funktionen bilden ein ereignisbasiertes API, das die definierte XML Typhierarchie auf programmiersprachliche Konstrukte (siehe auch das Beispiel in Tabelle 1) abbildet. Sie müssen vom Applikationsentwickler mit Leben gefüllt werden, genau wie das Erstellen der XCast-Filter in seine Verantwortung fällt. Einen Überblick des vollständigen Prozesses gibt Abbildung 1.

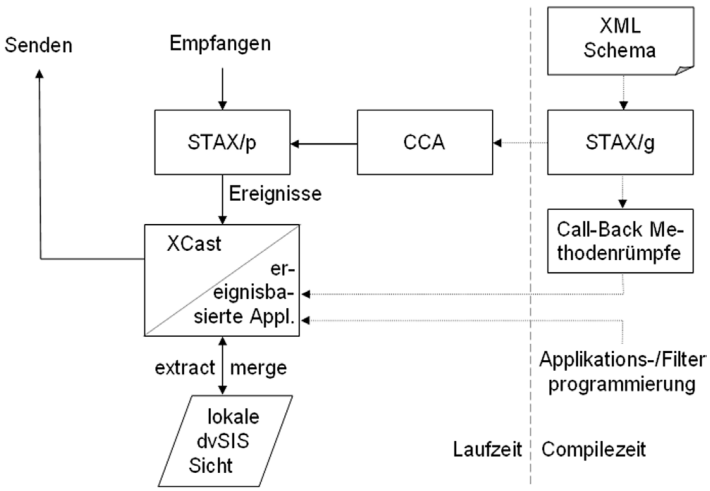


**Abbildung 1.** Prozess einer informationszentrierten Applikationsentwicklung

Ausgehend vom <<ASTAX Framework kann ein gradliniger Prozess zur Entwicklung von dvSIS-basierter Applikationssoftware definiert werden. Er umfasst drei wesentliche Schritte: Der

erste und grundlegendste Schritt ist die Spezifikation des dvSIS, formalisiert in einem XML Schema, das Typ und Struktur aller Informationseinheiten im dvSIS und allen damit zusammenhängenden Nachrichten beschreibt. Im zweiten Schritt wird aus diesem Schema mit Hilfe von STAX/g der validierende Parser STAX/p mit der zugehörigen Automatendefinition sowie eine Menge von typisierten Callback-Funktionsrümpfen generiert. Als dritten Schritt implementiert der Applikationsentwickler die Callback-Funktionen und die XCast-Filter. Danach folgt der übliche Softwareentwicklungsprozess aus Deployment, Betrieb und Wartung, der sich an den allgemeinen Methoden der Softwareentwicklung orientiert.

Das <<ASTAX Framework unterstützt das Generieren von Java oder C Code, wobei letzteres auf typische Sensorknoten zielt. Der C Code ist in Implementierungs- und Headerdateien (.c and .h-Dateien) unterteilt, die die leeren Funktionsrümpfe bzw. die Funktionsheader enthalten. Diese Funktionen werden von STAX/p als Callbacks aufgerufen, wenn die entsprechenden XML Elemente (z.B. XML Start- oder Endtags, Attribute oder Character Data) gelesen und erfolgreich validiert wurden. Für jedes XML Element und Attribut werden entsprechende .c und .h-Dateien generiert. Eine .c-Datei für einen `complexType` umfasst so genannte `create` und `add` Funktionen für seine Kindelemente und Attribute, während für `simpleTypes` lediglich eine `setContent` Funktion enthalten ist. Für die Dokumentenwurzel wird per Default die Datei Start.c erzeugt, die die entsprechenden Funktionen enthält. Zusätzlich können für `complexTypes` so genannte `contains` und `merge` Callbacks erzeugt werden, um Datenaggregation während des Parsens zu ermöglichen [LRK04]. Tabelle 1 zeigt ein Beispiel: Die linke Seite zeigt ein Fragment eines XML Dokuments, das einen Sensorwert repräsentiert. Die Sequenz der Callback-Aufrufe, die beim Parsen und Validieren auftreten, ist auf der rechten Seite dargestellt.

| XML Token | | File | Inserted method |
|---|---|---|---|
| `<dvSIS>` | → | Start.c | create_dvSIS() |
| `  <temp` | → | dvSIS.c | create_temp () |
| `  time="11:23:00"` | → | temp.c | create_time() |
| | → | time.c | setContent("11:23:00") |
| | → | temp.c | add_time() |
| `  x="21"` | → | temp.c | create_x() |
| | → | x.c | setContent(21) |
| | → | temp.c | add_x() |
| `  y="5">` | → | temp.c | create_y() |
| | → | y.c | setContent(5) |
| | → | temp.c | add_y() |
| `  32` | → | temp.c | setContent(32) |
| `  </temp>` | → | dvSIS.c | contains temp() |
| | → | dvSIS.c | add_temp() *or* merge_temp() |
| `</dvSIS>` | → | Start.c | add_dvSIS() |

**Tabelle 1.** Sequenz der Funktionsaufrufe beim Parsen eines Instanzdokumentes

Der von STAX/g erzeugte C Code kann für eine Vielzahl von Plattformen kompiliert werden, darunter auch der Texas Instruments MSP430 Mikrocontroller, der auf den von der FU Berlin entwickelten Embedded Sensor Boards [ESB] eingesetzt wird und auf extrem geringen Energieverbrauch ausgelegt ist. Für diesen Controller haben wir haben wir eine prototypische Implementierung entwickelt und untersucht. Er verfügt unter anderen über einen UART, der für Kommunikation über das drahtlose Netzwerk genutzt wird. Das Empfangen eines Bytes löst

einen Interrupt aus, der das Ausführen der zugehörigen Interrupt Service Routine (ISR) bewirkt. Um das nächste Byte empfangen zu können, muss diese Routine rechtzeitig zurückkehren. Da STAX/p jedoch direkt Applikationsfunktionen aufruft, ist der Parser durch eine Ereigniswarteschlange von der ISR entkoppelt. Dabei erkennt die ISR lediglich die lexikalischen Einheiten und fügt der Schlange entsprechende Ereignisse hinzu.

| Code Module | Größe |
|---|---|
| dvSIS Callback-Funktionsrümpfe | 2.0 kB |
| dvSIS CCA Beschreibung | 0.3 kB |
| STAX/p statischer Parser | 4.1 kB |

**Tabelle 2.** Codegröße der von STAX/g für das Beispiel erzeugten Module

Wir haben mit dem Beispiel aus Tabelle 1 eine Reihe von Untersuchungen durchgeführt. Zunächst haben wir die Größe des Codes untersucht, der aus dem zugehörigen Schema (es lässt zwischen 1 und 100 Temperaturmesswerte zu) erzeugt wird. Tabelle 2 zeigt die Codegröße aufgeteilt in die drei Komponenten Funktionsrümpfe, CCA Beschreibung und statischer Parser. Obwohl die Größe der Funktionsrümpfe und des CCA selbstverständlich von der Komplexität des Schemas abhängt, wird dennoch deutlich, dass der Code für den ESB Knoten mit seinen 60kB Programmspeicher problemlos handhabbar ist. Nichtsdestotrotz können sich jedoch Probleme bei der Handhabung großer Instanzdokumente ergeben, da die Sensorknoten über nicht mehr als 2kB RAM verfügen.
Im Rahmen von Laufzeitmessungen haben wir auch gemessen, wie lange es dauert, eingehende Nachrichten zu verarbeiten. Das Beispiel aus Tabelle 1 wird in weniger als einer Millisekunde geparst und validiert. Steigert man die Anzahl der in der Nachricht enthaltenen Messwerte, wächst die Verarbeitungszeit im Parser quadratisch und im Validierer linear. Wir sind jedoch davon überzeugt, dass das quadratische Verhalten des Parsers auf ein Implementierungsproblem (vermutlich quadratisches Verhalten der Speicherverwaltung) zurück zu führen ist, und das Parsen prinzipiell in linearer Zeit möglich ist.

## 5 Zusammenfassung aus Ausblick

In dieser Arbeit haben wir aufgezeigt, wie ein informationszentrierter Prozess zur Entwicklung von Applikationssoftware für drahtlose Sensornetze aussehen kann. Auf Basis des *distributed virtual Shared Information Space*, der die Mechanismen des CCA und von XCast verwendet, konnten wir zeigen, dass die Verarbeitung von XML auf ressourcenbeschränkten Geräten wie typischen Sensorknoten möglich ist.
Neben der Erweiterung und Optimierung der <<ASTAX Implementierung umfassen die nächsten Schritte die Entwicklung kompletter Applikationen auf Basis der vorgestellten Konzepte. Auf diese Weise wollen wir mehr Erfahrungen mit dem dvSIS und der Art und Weise, wie man damit effizient Applikationen entwickeln kann, gewinnen. Darüber hinaus werden wir Mechanismen für Kontextbewusstsein integrieren und effizientere Kodierungen des XML Information Set für den Transport über die Luftschnittstelle entwickeln, die dennoch direktes Parsen und Validieren erlauben.

## Literatur

[ASSC02] Akyildiz, I., Su, S., Sankarasubramanian, Y., Cayirci, E.: Wireless Sensor Networks; A Survey. *Computer Networks*. 38(4):393ñ422. 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., 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., 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., 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., 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.