Electronic Communications of the EASST
Volume 37 (2011)

Proceedings of the
Workshops der wissenschaftlichen Konferenz
Kommunikation in verteilten Systemen 2011
(WowKiVS 2011)

AFFIX – Ein ressourcenbewusstes Framework für Kontextinformationen in eingebetteten verteilten Systemen

Michael Schulze, Marcus Förster

12 pages

Guest Editors: Horst Hellbrück, Norbert Luttenberger, Volker Turau
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

AFFIX – Ein ressourcenbewusstes Framework für Kontextinformationen in eingebetteten verteilten Systemen

Michael Schulze, Marcus Förster

Institut für Verteilte Systeme
Arbeitsgruppe Eingebettete Systeme und Betriebssysteme
Universität Magdeburg
{mschulze@ivs.cs, Marcus.Foerster@st}.ovgu.de

Abstract: Kontextinformationen im Allgemeinen sind von enormer Bedeutung, um den Sinngehalt von Aussagen oder abstrakter ausgedrückt von Daten überhaupt zu erfassen. Für die Interaktion in verteilten Systemen sind sie von enormer Wichtigkeit. Jedoch gerade im eingebetteten Bereich besitzen die Geräte oft nicht die notwendigen Ressourcen, um bestehende Lösungen basierend auf XML oder OWL zu verarbeiten. Um diese Lücke zu schließen, wird AFFIX vorgestellt, welches ressourceneffiziente Mechanismen für das Applizieren und Verarbeiten von Kontextinformationen auf Basis von Attributen bereitstellt. Im Vergleich schneidet AFFIX mit bis zu 87,5% geringerem Platzbedarf für die Binärrepräsentation ab, was sich insbesondere auf ressourcenlimitierten Systemen sowie bei der Kommunikationen über eingebettete Netzwerke bemerkbar macht.

Keywords: Kontextinformationen, eingebettete Systeme, Publish/Subscribe

1 Einleitung

Kontextinformationen im Allgemeinen sind von enormer Bedeutung, um den Sinngehalt von Aussagen oder abstrakter ausgedrückt von Daten überhaupt zu erfassen. In unserem täglichen Umfeld könnten wir ohne diese zusätzlichen Informationen keine koordinierten Interaktionen mit anderen Menschen durchführen. Zum Beispiel lässt der Satz „Gibst Du es mir bitte.“, für sich allein betrachtet, keine Aussage über das Was zu. Plausible Objekte wären: Buch, Kugelschreiber, Hammer, Messer, etc. Menschen haben eine Fülle von Möglichkeiten den Kontext zu setzen und zu erfassen, sei es durch zusätzliche weitere gesprochene Sätze, durch nonverbale Kommunikation mit Hilfe von bsw. Gesten oder durch das Arbeiten an einer gemeinsamen Aufgabe, um nur einige zu nennen.

In Softwaresystemen insbesondere in verteilten eingebetteten Systemen spielen Kontextinformationen ebenfalls eine entscheidende Rolle, da solche Systeme oftmals Aufgaben realisieren, die zeitliche sowie qualitative Randbedingungen aufweisen. Bei Nichteinhaltung offenbart sich zumindest eine Beeinträchtigung der Qualität des erbrachten Services (QoS). Der Kontext wird demzufolge durch die Systemumgebung, die gemeinsame Aufgabe und die ausgetauschten Daten definiert. In statischen Umgebungen ist der Kontext implizit gegeben und braucht nicht zusätzlich übermittelt zu werden, weil dort durch das Setup bzw. Design des Systems grundsätzlich klar ist, wer, wem, was, wie mitteilt. Wenn jedoch Systeme dynamische Interaktionsbeziehungen vor-

aussetzen, ist ein Apriori-Wissen nicht vorhanden und Kontextinformationen sind explizit beizusteuern. Vielfach wird auf XML oder OWL für die Beschreibung des Kontextes zurückgegriffen [SKSM06, GPZ04]. Einerseits existiert hier eine Vielfalt an vorhandenen Softwarelösungen zu deren Verarbeitung. Andererseits wird das Entwickeln und Debuggen durch die Tatsache der leichten Interpretierbarkeit auch durch den Menschen erleichtert. Nachteilig sind jedoch die zur Übertragung und Verarbeitung benötigten Ressourcen, die eingebettete Systeme oftmals nicht bereitzustellen vermögen.

In diesem Zusammenhang stellt sich die Frage: Welche Informationen beschreiben den Kontext einer Interaktion zwischen Kommunikationsteilnehmern? Zur Verdeutlichung, welche Kontextinformationen und -quellen relevant sind, wird als Beispiel die Teleoperation eines Roboters betrachtet.

- Die Teleoperationsanwendung definiert Dienstgüteanforderungen an die Kommunikation wie die maximale Latenz für Datenpakete oder den minimal benötigten Durchsatz.
- Das Kommunikationssystem spezifiziert die erbringbaren Eigenschaften z. B. ebenfalls in Form von Latenz und Durchsatz.
- Die kommunizierten Daten enthalten zusätzliche Informationen wie Sequenznummern, Zeitstempel, Gültigkeit oder Position im Falle eines mobilen Roboters.

Mit Hilfe solcher Informationen oder auch Spezifikationen können erstens Plausibilitätstests im Voraus der Kommunikation durchgeführt werden. Zweitens ermöglichen sie das Einrichten und Überwachen der Dienstgüte (Kontext der Kommunikation selbst) und drittens können die den Daten beigefügten Kontextinformationen bei etwaigen Kommunikationsproblemen beispielsweise bei Verzögerungen helfen, geeignet darauf zu reagieren.

In dem vorliegenden Papier wird AFFIX (Attribut Framework For provIding conteXt) beschrieben. AFFIX ermöglicht Spezifikationsbeschreibungen, -prüfungen und -überwachungen sowie das Anheften, Verarbeiten und Filtern von Kontextinformationen. Weiterhin realisiert AFFIX eine effiziente und ressourcenschonende Binärkodierung der Kontextinformationen, so dass selbst sehr limitierte eingebettete Systeme wie Mikrocontroller mit wenigen kByte RAM und FLASH Speicher diese nutzen können.

Im folgenden 2. Abschnitt werden verwandte Arbeiten auf dem Gebiet der Beschreibungssprachen und Binärkodierungen untersucht und deren Limitierungen aufgezeigt. Im 3. Abschnitt wird kurz die publish/subscribe Middleware FAMOUSO vorgestellt, in deren Umfeld AFFIX entwickelt, eingebettet und eingesetzt wird. Daran anschließend wird das Konzept von AFFIX im 4. Abschnitt detailiert vorgestellt und im 5. evaluiert. Das Papier schließt mit einer Zusammenfassung.

2 Verwandte Arbeiten

Frølund und Koistinen stellen in [FK98] die QoS Modeling Language (QML) vor, mit deren Hilfe Anwendungen Kommunikationsanforderungen formulieren. Die so spezifizierten QoS-Parameter werden für die Aushandlung der Dienstgüte zwischen Diensterbringer und Dienstnehmer verwendet. Ein Schwachpunkt in QML ist die Möglichkeit, Anforderungsausdrücke wie

$latency > 30s$ zu definieren. Ein Anwendung könnte also eine „Mindestlatenz" fordern und die Frage ist, was ein System mit dieser Aussage anfangen kann. Mindestlatenzen sind immer erfüllbar, indem die Antwort des Diensterbringers mindestens um den Wert der Latenz verzögert wird. Jenes kann lokal im Dienstnehmer erfolgen und muss nicht Gegenstand der Dienstgüteaushandlung sein. Eine Anwendung spezifiziert mit einer Mindestlatenz also eher den frühest möglichen Antwortzeitpunkt und nicht die Latenz der Datenübertragung selbst. Die von AFFIX bereit gestellte Attributesprache ist entsprechend strikt und erlaubt keine mehrdeutigen Definitionen.

Ähnlich zu QML erscheinen die Ansätze von Li zur QoS-Erweiterung der ANSA Architektur ([Li94]). Hier werden Konzepte analog zu denen in [FK98] vorgestellt. Leider bleibt vollständig unklar, wie genau sich ein sogenanntes *QoS-item* definiert. Sowohl ein Paar (Attributtyp, Wert) als auch ein Tripel (unter Einbeziehung einer Relation) werden als mögliche Darstellungsform angeboten, jedoch ohne einen Hinweis unter welchen Bedingungen und mit welcher Bedeutung eine der beiden Formen zur Anwendung kommt. Weiterhin fraglich ist, wie die Bereichskomparatoren einzuordnen sind. Semantisch sind diese ähnlich der Tripel-Form der *QoS-item*-Darstellung mächtig. Ein bestimmter QoS-Parameter wird hinsichtlich der möglichen Werte eingeschränkt. Da nur für wenige kleine Teile von ANSA überhaupt prototypische Implementierungen existieren, wird entsprechend keine praktische Umsetzung der QoS-Sprache bzw. deren Anwendung vorgestellt.

Es gibt in der Literatur noch weitere Arbeiten, die sich mit QoS Spezifikationen in Form von formalen Beschreibungen beschäftigen. Oft werden Lösungen erarbeitet, die entweder nur zur Designphase der Anwendungsentwicklung oder nur zur Laufzeit wirken ([HSG07], [KZSP08], [MRA+08], [LBS+98]). Andere Ansätze schließen aufgrund der Anforderungen an die Hardware den Einsatz auf eingebetteten Systemen aus ([DXGE07], [VDM+01]). Hier sind ebenfalls die Ansätze basierend auf XML und OWL z. B. in [SKSM06, GPZ04] einzuordnen, die aufgrund des enormen Betriebsmittelbedarfes für die Verarbeitung dieser Sprachen ungeeignet sind.

Eine ressourcenschonendere binäre Repräsentation der Kontextinformationen könnte man zum Beispiel mit Hilfe von ASN.1 [LKM94] oder XDR [Eis06] realisieren. Leider ist die Kodierung von Informationen mit diesen Notationen ebenfalls nicht optimal. Beispielsweise kodiert ASN.1 die definierten Typen explizit, wodurch pro Datum zusätzlich ein Programmiersprachentypidentifikator wie Integer benötigt wird. Bei den im Allgemeinen relativ kleinen Datenmengen, die pro Attribut anfallen, würde ASN.1 einen erheblichen Overhead generieren.

XDR auf der anderen Seite verzichtet auf eine explizite Typisierung der kodierten Daten, allerdings gibt es eine vorgeschriebene Blockgröße von 4 Byte pro Datenelement. Unter der Annahme, dass Attribute aus mehreren Datenelementen bestehen (mindestens: Attributtyp und Wert), würde jedes Attribut mindestens 8 Byte in seiner Binärdarstellung verbrauchen. Attribute sollen jedoch lediglich als Meta-Informationen an die kommunizierten Nutzdaten angehängt werden. Dabei ist das Ziel, eine minimale Anzahl an Bytes für alle angehängten Attribute zu reservieren. Eine Mindestbedarf von 8 Byte pro Attribut stehen diesem Ziel entgegen.

3 FAMOUSO

FAMOUSO (**F**amily of **A**daptive **M**iddleware for autonom**OU**s **S**entient **O**bjects [Sch08, Sch09]) ist eine ereignisbasierte Kommunikationsmiddleware, der ein Publisher/Subscriber Modell zu

Grunde liegt. FAMOUSO realisiert eine einfach anzuwendende Kommunikationsinfrastruktur, die unabhängig von dem unterliegenden Kommunikationsmedium sei es ein CAN-Bus [Rob91] oder ein IP-basiertes Netzwerk der Anwendung eine uniforme Programmierschnittstelle bietet. Die Middleware läuft auf Systemen unterschiedlicher Leistungsklassen angefangen vom ressourcenstarken Server bis hinunter zum limitierten Mikrocontroller. Aufgrund dieser Eigenschaften besitzt FAMOUSO geeignete Mittel für die Adaption auf eine spezifische Plattform beziehungsweise ein Kommunikationsmedium und bietet überdies Mechanismen für die dynamische Anpassung an Umgebungsparameter [Sch09].

Publisher und Subscriber sind Rollen, die Anwendungen bei der Kommunikation einnehmen und entsprechend spezifizieren sie die Art der Ereignisse, welche sie produzieren bzw. konsumieren. Auf dieser Grundlage bietet FAMOUSO ein spontanes, dynamisches n-zu-m Kommunikationverhalten ohne Annahmen bzgl. der Ereignissynchronität. Die asynchrone Kommunikation verhindert Kontrollflussabhängigkeiten und bewirkt die Autonomie der Teilnehmer.

Der Kommunikationsgegenstand – das Ereignis – kann durch verschiedene Stimuli erzeugt werden. Einerseits können externe Ereignisse von Sensoren detektiert zur Generierung eines internen Ereignisses führen oder anderseits periodisch durch eine Uhr initiiert werden, um beispielsweise den Status einer Variablen innerhalb eines Knotens kontinuierlich im System zu verbreiten. Unabhängig von der Art des Ursprunges sind FAMOUSO-Ereignisse folgendermaßen aufgebaut:

Subjekt repräsentiert durch einen eindeutigen Bezeichner, charakterisiert den Inhalt.

Inhalt sind die Daten selbst, z. B. der Wert einer Distanzmessung.

Kontextattribute z. B. Position, Zeit oder Gültigkeit, welche optional sind.

Ein Subjekt hat eine globale Bedeutung, weshalb Subjekte einen eindeutigen Adressraum definieren. Dieser Umstand wird genutzt, um zwischen verschiedenen Netzen eine Kommunikation zu ermöglichen und diese zu steuern. Die Eindeutigkeit der Subjekte erlaubt erstens den Informationsfluss an Netzwerkgrenzen zu filtern, falls ein Subjekt nur innerhalb eines spezifischen Teilnetzes angefordert wurde und zweitens werden Ereignisse entsprechend weiter propagiert, wenn deren Subjekte außerhalb abonniert wurden.

Aus Anwendungssicht und für viele Einsatzzwecke wäre die Definition der Ereignisse und deren direkte Benutzung ausreichend. Kommen jedoch QoS-Anforderungen wie Echtzeitfähigkeit oder Zuverlässigkeit hinzu, braucht es eine weitere Abstraktion. FAMOUSO verwendet neben den beschriebenen Ereignissen daher Ereigniskanäle als Abstraktion für die Ereignisübertragung. Ein Ereigniskanal ist grundsätzlich unidirektional, wodurch die Rolle als Publisher beziehungsweise als Subscriber klar erkennbar ist. Die Definition eines Ereigniskanals ähnelt dem eines Ereignisses und gliedert sich wie folgt:

Subjekt entspricht den korrespondierenden Ereignissen.

QoS-Attribute beschreiben Disseminationseigenschaften wie Periode, Frist oder Latenz.

Callbacks definieren einerseits einen Ausnahmehandler zur Signalisierung von Verletzungen der spezifizierten QoS-Attribute und andererseits einen Benachrichtigungshandler für die Signalisierung von eingetroffenen Ereignissen.

Ereigniskanäle dienen der Anwendung folglich zur Spezifikation von Anforderungen und zweitens ermöglichen sie der Middleware notwendige lokale sowie netzwerkseitige Ressourcen zu reservieren. Neben der Ressourcenreservierung wird bei der Erzeugung eines Ereigniskanals auch das Binden des Subjektes an eine spezifische Netzwerkadresse durchgeführt. Dies geschieht jedoch vollständig transparent für die Anwendung und FAMOUSO verbirgt somit die Heterogenität der unterliegenden Netzwerke.

4 AFFIX – Attribute Framework For provIding conteXt

Der konzeptionelle Ausgangspunkt für die Entwicklung von AFFIX ist die Multi-Level Composability Check Architecture (MLCCA). In [SL09] stellen Schulze und Lukas MLCCA vor, welche Komponierbarkeitstests und das Überprüfen von Spezifikationen auf Gültigkeit zur Design-, Übersetzungs- und Ausführungszeit ermöglicht. AFFIX stellt eine effiziente Realisierung der MLCCA dar und fügt überdies die Möglichkeit von optionalen Kontextattributen an Ereignissen, Filterspezifikationen und eine effiziente Binärrepräsentation hinzu.

Zur Verdeutlichung des Konzeptes zeigt die Abbildung 1 an welchen Positionen Spezifikationen, Filter sowie Attributannotationen ihre Wirkung entfalten. Innerhalb der Netzwerkebene, bei FAMOUSO durch die Netzwerkschicht repräsentiert, beschreiben beispielsweise netzwerkspezifische QoS-Provisions die Eigenschaften des Netzwerkes, wodurch sie explizit gemacht werden. Anwendungen steuern ihre Anforderungen in Form von QoS-Requirements bei und die Überprüfung auf eine Spezifikationsverträglichkeit ist ein integraler Bestandteil von AFFIX.

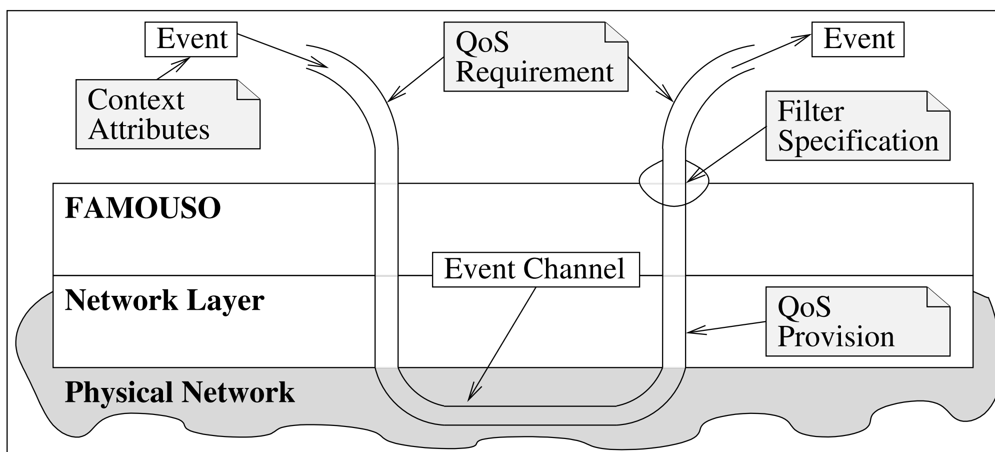

Abbildung 1: QoS-Spezifikationen, Filter und Kontextannotationen und deren Auftreten mit Bezug zur FAMOUSO Middleware und der Anwendung

Wenn die Spezifikationen das Etablieren der Kommunikation gestatten, sind Subscriber dennoch nicht grundsätzlich an allen Ereignissen eines Kanals interessiert. Um Subscriber vor nicht benötigten Ereignissen abzuschirmen, realisiert AFFIX eine Filtersprache für Kontextattribute, damit Subscriber die Selektion nicht selbst durchführen müssen. Nur wenn ein Ereignis einen

gesetzten Filter passieren kann, wird der Subscriber notifiziert, wodurch die Last auf Anwendungsebene insgesamt reduziert wird und nicht benötigte Ereignisse frühzeitig aussortiert werden. Dieser Mechanismus hilft, Ressourcen insbesondere zur Laufzeit zu sparen, was sich sehr vorteilhaft auf eingebettete limitierte Systeme auswirkt.

Wie bereits MLCCA ist auch AFFIX vollständig in C++ realisiert und bedient sich extensiv der Template-Metaprogrammierung [Vel95, SSK10], welche lediglich einen standardkonformen Compiler voraussetzt. Sämtliche Attribute werden dafür auf C++-Typen abgebildet, mit denen man in der Folge während des Übersetzungsvorganges Berechnungen, Prüfungen, Selektionen, etc. durchführen kann. Weiterhin lässt sich in Abhängigkeit von den verwendeten Typen angepasster Quellcode automatisch durch die Metaprogramme erzeugen. Dieser Ansatz erlaubt es, nur die tatsächlich verwendete und notwendige Funktionalität mit in das Endsystem aufzunehmen, wodurch sich AFFIX sehr ressourceneffizient darstellt.

4.1 Attributsprache

Die Attributsprache besteht aus den Grundelementen: Attribut, Operator und Ausdruck. Aus Platzgründen wird auf eine vollständige formale Definition der Sprache im Rahmen dieses Papiers verzichtet und statt dessen ein Überblick gegeben.

Attribute die Basisentitäten des Frameworks sind Tupel der Form (Attributtyp, Wert). Zusätzlich hat jedes Attribut eine Stärkerrelationseigenschaft und eine unbestimmte Anzahl an Kennzeichnern. Die Stärkerrelationseigenschaft beschriebt, ob Attribute gleichen Typs mit einem anderen Wert stärker sind. Zum Beispiel ist $(Durchsatz, 300\,kB/s)$ schwächer als $(Durchsatz, 500\,kB/s)$, aber $(Latenz, 10\,ms)$ ist stärker als $(Latenz, 20\,ms)$. Die Stärkerelation definiert folglich eine Ordnung innerhalb einer Attributklasse. Kennzeichner hingegen beschreiben weitere Eigenschaften eines Attributes. Dies können systemrelevante wie *requirable* oder beliebige benutzerdefinierte sein.

Operatoren dienen dem relationalen Vergleich ($<$, $\leq$, $==$, $>$, $\geq$), dem logischen Verknüpfen (not, and, or) oder der Existensprüfung ($\exists$), wodurch im Ergebnis ein Wahrheitswert geliefert wird. Der Mengenoperator (set) erzeugt eine Sequenz von Attributen und die Zuweisung ($=$) weist einem Attribut einen angegebenen Wert zu.

Ausdrücke beschreiben Verknüpfungen von Attributen mit Hilfe der vorgestellten Operatoren und Ausdrücke können selbst Teil eines umfassenden Ausdrucks sein, so dass sich eine rekursive Struktur ergibt.

In der obigen Aufstellung wurden alle Operatoren zusammengefasst dargestellt, was jedoch nicht den Regeln der formalen Sprachdefinition entspricht. Deshalb wird in den folgenden Abschnitten konkretisiert, welche Operatoren für die unterschiedlichen Spezifikationsarten maßgebend sind.

4.2 QoS-Spezifikationen

AFFIX unterscheidet zwei Arten von QoS-Spezifikationen. Auf der Systemseite beschreibt die QoS-Provision die Fähigkeiten bzw. den Kontext eines Netzwerkes und die aufgelisteten Attribute definieren Maximalwerte. Anwendungsseitig werden QoS-Requirements spezifiziert, die

die Anforderungen an die Kommunikation definieren und für jedes geforderte Attribute wird ein Mindestwert verlangt. Generell dürfen in den QoS-Spezifikationen nur Attribute auftauchen, die mit *requirable* gekennzeichnet sind, was letztendlich nur auf Attribute zutrifft, die Qualitätseigenschaften beschreiben. Demzufolge darf ein TTL-Attribut (Time-To-Live) oder benutzerdefinierte Attribute dieses Kennzeichen nicht besitzen.

Da eine Provision Systemfähigkeiten beschreibt, kann ein Attribut sinnvollerweise nur genau einen konkreten Wert besitzen. Eine Provision z. B. $(Latency < 20ms)$ hätte keinen Mehrwert. Es würde die Frage aufkommen, welche maximale Latenz die Anwendung annehmen darf. Eine Obergrenze von $20ms$ bedeutet letztendlich, dass nur dies garantiert wird. Analog gilt dies für Aussagen wie $(Throughput > 300 kbit/s)$. Um Mehrdeutigkeiten auszuschließen, wird für Provisions nur der $=$ und der *set* Operator zugelassen und ein Attribut darf maximal ein Mal in einer Spezifikation erscheinen. Somit kann ein Systementwickler jedem spezifizierten Attribut lediglich einen definierten Wert zuordnen. Falls das System unvorhersehbar bessere Werte liefert, stellt dies keine Verletzung der Spezifikation dar, da die Garantie nach wie vor hält.

Für QoS-Requirements verhält es sich bzgl. der verwendbaren Operatoren genauso wie bei den Provisions. Der Unterschied besteht in der Semantik, denn eine Anwendung fordert durch ein Requirement QoS-Minimalwerte für bestimmte Attribute, wo die Provision QoS-Maximalwerte für Attribute definiert.

Auf Programmiersprachenebene werden sowohl Provisions als auch Requirements auf C++-Klassentemplates abgebildet, wodurch sie für Template-Metaprogramme zugänglich und verarbeitbar werden. Der folgende kurze Quelltext zeigt jeweils eine Provision, ein Requirement und den Aufruf eines Metaprogramms, dass die Konformität der beiden Spezifikationen überprüft.

```
typedef SetProvider<Latency<10>, Throughput<500> >::attrSet prov;
typedef SetProvider<Latency<20>, Throughput<300> >::attrSet req;

typedef RequirementChecker<prov, req>::type checker;
```

Im gezeigten Beispiel kann die Anforderungen erfüllt werden. Sollte der Test jedoch eine Unverträglichkeit ergeben, gibt der Compiler eine „sprechende“ Fehlermeldung aus und bricht die Übersetzung ab. Die Fehlermeldung zeigt die Attribute, welche nicht durch das System garantiert werden können und demzufolge auch nicht zur Laufzeit zu realisieren sind. Dadurch können semantische Fehler frühzeitig entdeckt werden [SL09]. Forderungen nach Attributen, welche nicht in einer Provision erscheinen, gelten grundsätzlich als unerfüllbar, da dass System über diese keine Aussage trifft.

4.3 Kontextannotationen

QoS-Spezifikationen werden an der Netzwerkschicht bzw. an Ereigniskanälen annotiert. Diese Spezifikationen beschreiben den Kontext für die Übertragung von Ereignissen. Um Ereignisse ebenfalls in einen Kontext zu setzen, ermöglicht AFFIX das Anheften von Kontextattributen auch hier. Prinzipiell gelten die gleichen Bedingungen und Operatoren wie bei den Spezifikationen mit der Ausnahme, dass auch Attribute mit anderen Kennzeichnungen als *requirable* erlaubt sind. Zum Beispiel definiert

```
typedef ExtendedEvent<16, Id<10>, TTL<0>, TimeStamp<>, Location<> > e;
```

ein Ereignis mit verschiedenen Attributen und einer verfügbaren Nutzlast von 16 Byte. Weiterhin zeigt es, dass *Id* mit dem Wert 10 und *TTL* mit 0 initialisiert wird. *TimeStamp* sowie *Location* erhalten Standardwerte, weil kein Parameter angegeben wurde. Dies bewirkt, dass sie mit der momentan Zeit und dem Ort besetzt werden, sobald ein Objekt vom Typ *e* erzeugt wird, wodurch ebenfalls die Attribute in eine Binärrepräsentation (siehe 4.5) überführt werden.

4.4 Filterspezifikationen

Im Gegensatz zu den bisher betrachteten Spezifikationen und Annotationen arbeiten Filterspezifikationen grundsätzlich in der Laufzeitwelt auf den Binärrepräsentationen der Attribute. Ein gegebener Filter untersucht und klassifiziert die einem Ereignis angehefteten Attribute und liefert einen Wahrheitswert. Folglich sind nur der Existenzoperator und die logischen sowie relationalen Operatoren zulässig, um eine Filterspezifikation zu formen, da weder der Zuweisungs- noch der Mengenoperator einen booleschen Wert generiert.

Für das problemorientierte Formulieren von Filtern stellt AFFIX eine Domain Specific Language (DSL) zur Verfügung, welche direkt in die Sprache C++ eingebettet ist. Mit Hilfe der DSL lassen sich Filterausdrücke sehr leicht erstellen und auf Ereignisse in Form von Lambdafunktionen anwenden. Ein Filterausdruck wie

$$(TTL > 0)(e) \Longrightarrow falsch$$

angewendet auf ein Ereignis vom Typ *e* aus dem vorangegangenen Abschnitt liefert den dargestellten Wahrheitswert. Welche Semantik im Einzelfall dahinter steckt, ist abhängig von der jeweiligen Anwendung. Die TTL gibt z. B. an, wie viele Netzwerkübergänge ein Ereignis maximal passieren darf. Wird der oben dargestellte Filter nun im Gatewayknoten vor der Weiterleitung auf Ereignisse angewendet, werden diese entsprechend in die Klassen *wahr* (Ereignis darf weitergeleitet werden) und *falsch* (Ereignis muss verworfen werden) eingeteilt. Mit Hilfe der Filter lässt sich somit nicht nur sehr feingranular der Internetzwerkverkehr steuern, sondern auch auf Anwendungsebene genau beschreiben, an welchen Ereignissen ein Subscriber interessiert ist.

4.5 Binärrepräsentation

Die Diskussion über bestehende Kodierungsverfahren in Abschnitt 2 offenbart, dass diese ungeeignet für ressourcenlimitierte Systeme sind. Daher wird eine eigene Binärdarstellung realisiert, die speziell auf Attribute zugeschnitten ist. Die resultierende Inkompatibilität zu bestehenden Standards ist zwar nachteilig, jedoch wiegt die effiziente Ressourcenausnutzung schwerer. Im Hinblick auf ressourcenbeschränkte Systeme, für die FAMOUSO maßgeblich konzipiert ist, lassen sich die folgenden Anforderungen herausstellen:

1. **Kompaktheit**: Die kodierten Daten sollen eine minimale Anzahl an Bytes benötigen und möglichst viele Informationen sollten hierfür implizit kodiert werden.

2. **(De)Kodierungsaufwand**: Die Überführung von Daten in die Binärdarstellung und zurück muss effizient sein.

Zwischen diesen beiden Punkten besteht ein Zielkonflikt, denn je kompakter die Darstellung um so mehr Aufwand für die Umwandlung ist notwendig. Ein weiterer wichtiger Aspekt bei der

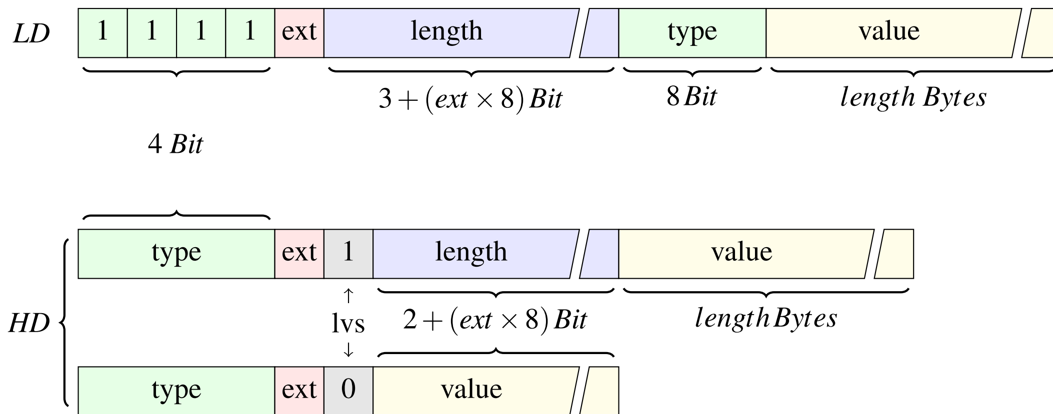

Abbildung 2: Attributbinärrepräsentation in *Low Density (LD)* und *High Density (HD)*

Entwicklung des Binärformats liegt auf dem effizienten Iterieren über Attributsequenzen. Das heißt, die Anzahl der benötigten Bytes, die ein Attribut beansprucht, muss bei der Dekodierung schnell verfügbar sein. Bei der Iteration erlaubt es ein rasches Überspringen von Attributen, die für bestimmte Prüfungen irrelevant sind. Gerade für die Anwendung von Filterausdrücken (siehe 4.4) spielt dies eine entscheidende Rolle und wurde deswegen bei der Konzipierung des Binärformats bedacht.

Das Binärformat ermöglicht *Low Density (LD)* und *High Density (HD)* Kodierung und grundsätzlich wird *Big Endian*-Byteordnung verwendet. Die *LD*/*HD* Unterteilung wurde eingeführt um sehr häufig zu übertragende Kontextattribute noch kompakter zu gestalten, wodurch erstens der Speicherverbrauch und zweitens die Ereignisgröße beim Versand minimiert wird. Welcher Klasse ein Attribut angehört, wird über ein zusätzliches Kennzeichen ähnlich dem *requirable* am Attribut spezifiziert. Maximal lassen sich 15 Attribute als *HD* und 256 als *LD* packen.

Allgemein wird zwischen Attributheader und -wert unterschieden. Der Header enthält den Attributtyp und die Kodierungsinformationen. Seine Größe variiert zwischen 6 Bit - 2 Byte bei *HD* und 2 Byte - 3 Byte bei *LD*. Die Abbildung 2 zeigt beide Kodierungsformen schematisch. Ein gesetztes *ext*-Bit bewirkt generell die Erweiterung des Headers um ein Byte. Ob eine *LD*- oder *HD*-Kodierung vorliegt, wird durch den Inhalt der ersten vier Bit bestimmt. Wenn dort nur Einsen stehen, ist es ein *LD*-Typ, bei dem der Attributtyp im letzten Headerbyte steht. Jedes andere Bitmuster in diesen vier Bit charakterisiert einen Attributtyp mit *HD*-Kodierung.

Ab dem sechsten Bit unterscheiden sich die Kodierungen und zunächst wird die *LD*-Variante betrachtet. Die dem *ext*-Bit folgenden Bits charakterisieren die Byteanzahl für die Repräsentation des Attributwertes. In Abhängigkeit vom *ext*-Bit sind es drei oder elf Bit, wodurch der Attributwert maximal 2^{11} Byte beanspruchen kann. Wie erwähnt, schließt sich der Längendarstellung der Attributtyp im letzten Headerbyte an, gefolgt vom Attributwert.

Die *HD*-Kodierung stellt sich komplexer dar. Hier enthält das sechste Bit einen zusätzlichen Schalter für den Wechsel zwischen impliziter und expliziter Längendarstellung. Das *lvs*-Bit (Length/Value Switch) zeigt an, ob ihm direkt der Attributwert $lvs = 0$ oder die benötigte Byteanzahl $lvs = 1$ folgt. In Verbindung mit dem *ext*-Bit ergibt sich ein zwei bzw. zehn Bit breites

Feld für die Wert- oder Längendarstellung. Im Fall der Längenkodierung schließt sich ihr die Attributwertdarstellung an.

Binärkodierte Attribute werden in einer Sequenz zusammengefasst. Dies gilt sowohl für QoS-Spezifikationen als auch für Kontextattribute, die an FAMOUSO-Ereignisse angeheftet werden. Dem bisherigen Schema folgend, wird auch hier ein effizientes Rahmenformat geschaffen.

Eine Attributsequenz beginnt ebenfalls mit einem Header, welcher die Gesamtbyteanzahl aller enthaltenen Attribute aufnimmt und ihm folgen die binärkodierten Attribute direkt. Wie bei den Attributen wurde auch hier ein *ext*-Bit verwendet, um für kleine Attributlisten ein Headerbyte zu sparen. Wenn $ext = 0$ stehen maximal $2^7 - 1$ Byte für Attributkodierungen zur Verfügung. In der erweiterten Form ($ext = 1$) wird analog zur Erweiterung des Attributheaders ein weiteres Byte angehängt. Dadurch stehen insgesamt 15 Bits zur Verfügung, womit Sequenzgrößen von bis zu 32535 Byte realisierbar sind. Vorteilhaft an der Kodierung der akkumulierten Byteanzahl ist, dass Anwendungen die Attribute schnell überspringen können, wenn an jenen kein Bedarf besteht. Würde die Attributanzahl stehen, müsste jedes Attribut dekodiert werden, um ein Überlesen zu zulassen und um an die Nutzdaten zu gelangen.

5 Ergebnisse

Zum Vergleich der Effizienz der *LD* und *HD* AFFIX-Kodierungsvarianten gegenüber ASN.1 unter Verwendung der Basic Encoding Rules (BER) und XDR wird die jeweils resultierende Datengröße in der folgenden Tabelle gegenübergestellt. Einerseits ist die Kodierung eines Attributs *Id* mit dem 16 Bit-Wert 0 dargestellt. Zum Anderen dient die Attributsequenz des in Abschnitt 4.3 definierten Ereignistyps *e* als Vergleichsbasis. Für *TTL* wird ein 8 Bit Wert angenommen. Ein *TimeStamp* repräsentiert die Anzahl der *ms* seit 1970, welches in einem 64 Bit-Wert gespeichert wird. Die *Location*, dargestellt als Zusammenfassung eines 16 Bit-Längen- und Breitengrades, benötigt zur Wertdarstellung daher 32 Bit. Die Tabelle stellt die Anzahl der jeweils benötigten Bytes und den daraus resultierenden Overhead relativ zur *LD*-Kodierung gegenüber.

Kodierung	`Id<0>`	Overhead	`Attributsequenz`	Overhead
AFFIX HD	1	33,33%	19	79,17%
AFFIX LD	3	100,00%	24	100,00%
ASN.1 (BER)	6	200,00%	33	137,50%
XDR	8	266,67%	36	150,00%

Sowohl bei der Kodierung einzelner Attribute als auch von Sequenzen liegt der allgemeine *LD*-AFFIX-Ansatz im Verbrauch unter den genannten Alternativen. Die *HD*-Variante erhöht die Effizienz der Kodierung nochmals, so dass z. B. für ein einzelnes *Id*-Attribut im Gegensatz zur XDR-Kodierung 87,5% weniger Speicherplatz benötigt wird.

6 Zusammenfassung

Die vorliegende Arbeit präsentiert AFFIX ein Framework für die ressourceneffiziente Bereitstellung von Kontextinformationen in eingebetteten verteilten Systemen. Das Framework integriert sich nahtlos in die ereignisbasierte publish/subscribe Middleware FAMOUSO, wo es

unterschiedliche Aufgaben wahr nimmt. AFFIX ermöglicht QoS-Spezifikationsbeschreibungen, -prüfungen und -überwachungen sowie das Anheften, Verarbeiten und Filtern von Kontextinformationen. Mit Hilfe einer in die C++-Sprache eingebetteten DSL lassen sich Filter überdies sehr einfach und problemorientiert formulieren, was einerseits die Lesbarkeit des Quelltextes erhöht und andererseits durch den verwendeten Mechanismus der Template-Metaprogrammierung ein hohes Optimierungspotential für den Compiler bietet.

Weiterhin realisiert AFFIX eine effiziente und ressourcenschonende Binärkodierung der Kontextinformationen, so dass selbst sehr limitierte eingebettete Systeme wie Mikrocontroller mit wenigen kByte RAM und FLASH Speicher diese nutzen können. Die Evaluierung zeigt, dass alternative Kodierungsansätze nicht die Effizienz erreichen und deshalb ungeeignet für diese Geräteklasse sind.

Literatur

[DXGE07] G. Deng, M. Xiong, A. Gokhale, G. Edwards. Evaluating Real-Time Publish/Subscribe Service Integration Approaches in QoS-Enabled Component Middleware. In *ISORC '07: Proceedings of the 10th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing*. Pp. 222–227. IEEE Computer Society, Washington, DC, USA, 2007.

[Eis06] M. Eisler. XDR: External Data Representation Standard. RFC 4506, May 2006. http://www.ietf.org/rfc/rfc4506.txt

[FK98] S. Frølund, J. Koisten. QML: A Language for Quality of Service Specification. Technical report, Hewlett Peckard Software Technology Laboratory, 1998.

[GPZ04] T. Gu, H. K. Pung, D. Q. Zhang. A middleware for building context-aware mobile services. In *Proceedings of IEEE 59th Vehicular Technology Conference. (VTC '04-Spring) (IEEE Cat. No.04CH37514)*. Volume 5, pp. 2656–2660. IEEE Computer Society Press, Milan, Italy, May 2004.

[HSG07] J. Hoffert, D. Schmidt, A. Gokhale. A QoS Policy Configuration Modeling Language for Publish/Subscribe Middleware Platforms. In *Proceedings of International Conference on Distributed Event-Based Systems (DEBS)*. 2007.

[KZSP08] J. Kaiser, S. Zug, M. Schulze, H. Piontek. Exploiting self-descriptions for checking interoperations between embedded components. *International Workshop on Dependable Network Computing and Mobile Systems (DNCMS 08)*, pp. 41–45, Oct. 2008.

[LBS+98] J. P. Loyall, D. E. Bakken, R. E. Schantz, J. A. Zinky, D. A. Karr, R. Vanegas, K. R. Anderson. QoS Aspect Languages and Their Runtime Integration. In *In Proceedings of the Fourth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR98)*. Pp. 28–30. Springer-Verlag, 1998.

[Li94] G. Li. A Model of Real-Time QoS. Technical report, Architecture Projects Management Limited - ANSA, 1994.

[LKM94] R. G. Lavender, D. G. Kafura, R. W. Mullins. Programming with ASN.1 Using Polymorphic Types and Type Specialization. In *Proceedings of the IFIP TC6/WG6.5 International Conference on Upper Layer Protocols, Architectures and Applications*. Pp. 151–166. Elsevier Science Inc., New York, NY, USA, 1994.

[MRA+08] E. Meshkova, J. Riihijärvi, J. Ansari, K. Rerkrai, P. Mähönen. An Extendible Meta-Data Specification for Component-Oriented Networks with Applications to WSN Configuration and Optimization. In *IEEE PIRMC 2008*. Cannes, France, Sept. 2008.

[Rob91] Robert Bosch GmbH. *CAN Specification Version 2.0*. September 1991.

[Sch08] M. Schulze. FAMOUSO Project Website. 2008-2011. [(online), as at: 25.02.2010]. http://famouso.sourceforge.net

[Sch09] M. Schulze. FAMOUSO – Eine adaptierbare Publish/ Subscribe Middleware für ressourcenbeschränkte Systeme. *Electronic Communications of the EASST (ISSN: 1863-2122)* 17:12, 2009. Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009). http://eceasst.cs.tu-berlin.de/index.php/eceasst/article/view/195/213

[SKSM06] T. Springer, K. Kadner, F. Steuer, Y. Ming. Middleware Support for Context-Awareness in 4G Environments. In *Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks (WoWMoM'06)*. Pp. 203–211. IEEE Computer Society Washington, DC, USA, Buffalo-Niagara Falls, NY, USA, June 2006.

[SL09] M. Schulze, G. Lukas. MLCCA – Multi-Level Composability Check Architecture for Dependable Communication over Heterogeneous Networks. In *Procedings of 14th International Conference on Emerging Technologies and Factory Automation*. ETFA'09, pp. 859–866. IEEE Press, Piscataway, NJ, USA, Palma de Mallorca, Spain, September 22-26 2009.

[SSK10] C. Steup, M. Schulze, J. Kaiser. Exploiting Template-Metaprogramming for Highly Adaptable Device Drivers a Case Study on CANARY an AVR CAN-Driver. In *Proceedings of 12th Brazilian Workshop on Real-Time and Embedded Systems (WTR)*. Gramado/RS, Brazil, 24 May 2010.

[VDM+01] N. Venkatasubramanian, M. Deshp, S. Mohapatra, S. Gutierrez, J. Wickramasuriya. Design and Implementation of a Composable Reflective Middleware Framework. In *Proceedings of the 21th International Conference on Distributed Computing Systems (ICDCS-21)*. Pp. 644–653. IEEE Computer Society Press, 2001.

[Vel95] T. L. Veldhuizen. Using C++ template metaprograms. *C++ Report* 7(4):36–43, May 1995. Reprinted in C++ Gems, ed. Stanley Lippman.