Electronic Communications of the EASST
Volume 17 (2009)

Workshops der Wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2009 (WowKiVS 2009)

FAMOUSO – Eine adaptierbare Publish/Subscribe Middleware für ressourcenbeschränkte Systeme

Michael Schulze

12 pages

Guest Editors: M. Wagner, D. Hogrefe, K. Geihs, K. David
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

FAMOUSO – Eine adaptierbare Publish/Subscribe Middleware für ressourcenbeschränkte Systeme

Michael Schulze

Institut für Verteilte Systeme
Arbeitsgruppe Eingebettete Systeme und Betriebssysteme
Universität Magdeburg
mschulze@ivs.cs.uni-magdeburg.de

Abstract: Eingebettete Systeme erbringen in unserem täglichen Leben eine Vielzahl von Aufgaben und der Anwendungsbereich umfasst ein weites Feld von der Haushaltselektronik über die Telekommunikation bis hin zu automotiven Anwendungen und Industrieautomation. Allgemein geht der Trend zu vernetzten Systemen, die zusammen eine Aufgabe erbringen. Software für solche Systeme grundsätzlich neu zu entwickeln, ist einerseits kostenintensiv und andererseits ist die Interoperabilität erschwert. Konfiguration und Adaption der Software an die jeweilige Situation ist wünschenswert. Als Antwort auf diese Herausforderungen wird in dieser Arbeit die adaptive publish/subscribe-Kommunikationsmiddleware FAMOUSO präsentiert. Weiterhin wird der Einsprachenansatz für die Adaption vorgestellt und gezeigt, welche Formen der Adaption unterstützt werden.

Keywords: verteilte eingebettete Systeme, Publish/Subscribe, Middleware, Adaption, Selbstorganisation

1 Einleitung

Eingebettete Systeme erbringen in unserem täglichen Leben eine Vielzahl von Aufgaben und ihre Existenz und Allgegenwärtigkeit wird von vielen Menschen oftmals nicht (mehr) wahr genommen. Der Anwendungsbereich umfasst ein weites Feld von der Haushaltselektronik über die Telekommunikation bis hin zu automotiven Anwendungen und Industrieautomation. Die Entwicklung geht insgesamt von Einzelsystemen zu verteilten und vernetzen Systemen, die eine Gesamtaufgabe in Kooperation erbringen. Mit der zunehmenden Komplexität und Funktionsvielfalt steigt auch der Aufwand der Integration und im Allgemeinen ist eine Adaption an die jeweilige Umgebung wünschenswert.

Für entsprechend leistungsstarke Systeme mit ausreichend Ressourcen gibt es bereits Lösungen, die im Sinne des Plug-and-Play eine dynamische Laufzeitadaption ermöglichen. Hierbei handelt es sich jedoch meist um Standard-IT Systeme, die viele MB Speicher aufweisen und über eine rechenkräftige CPU verfügen. Wird jedoch das weite Feld der eingebetteten Systeme betrachtet, in dem mehr als 98% der im Jahre 2000 produzierten Prozessoren zum Einsatz kamen [Ten00], findet man dort meist sehr limitierte Ressourcen im Bezug auf Speicher und Rechenleistung vor. Dies begründet, dass bekannte Techniken und Vorgehensweisen von standardisierten IT-Systemen nicht anwendbar beziehungsweise nur in sehr begrenztem Ausmaß adaptiert werden können.

Eine Systemsoftware (z.B. Betriebssystem, Middleware) darf in diesem Bereich der Anwendung nur die Funktionalität bereit stellen, die auch tatsächlich benötigt wird. Hierdurch wird den limitierten Ressourcen Rechnung getragen und es verbleibt mehr Raum für die Erfüllung der eigentlichen anwendungsspezifischen Aufgaben. In diesem Umfeld bietet sich der familienbasierte Ansatz nach Parnas [Par76] an, der von einer minimalen Basis ausgehend eine immer weitere Verfeinerung vorschlägt. Durch entsprechende Auswahl bestimmter für die Anwendung notwendiger Erweiterungen und Funktionalitäten wird das Familienmitglied bestimmt und ausgewählt, welches den Anforderungen minimal gerecht wird. Neben diesem Ansatz der Familienorientierung wird in dieser Arbeit ein neuer Ansatz zur effizienten Adaptierung für ressourcenbeschränkte Systeme vorgestellt.

Die weitere Arbeit ist wie folgt gegliedert. Der Abschnitt 2 stellt eine Einordnung von Adaptionformen vor. In Abschnitt 3 wird die publish/subscribe Middleware FAMOUSO vorgestellt sowie gezeigt, welche Formen der Adaption unterstützt werden und wie diese umgesetzt sind. Das Papier schließt mit verwandten Arbeiten auf diesem Gebiet und einem Fazit mit Ausblick.

2 Formen der Adaption

Adaption von Software hat unabhängig vom Einsatzgebiet in z.B. eingebetteten Systemen viele Freiheitsgrade. Unter der Voraussetzung, dass das Warum (Umgebungsanpassung, Performanceverbesserung, Fehlerbeseitigung, etc.) der Adaption geklärt ist, muss zunächst bestimmt werden, welche Formen der Adaption existieren und wie diese einzuordnen sind. Hierfür werden drei Fragestellungen formuliert, an denen entlang eine Klassifizierung vorgenommen wird:

- Was wird adaptiert?
- Wann wird adaptiert?
- Wie wird adaptiert?

Die Fragen beziehungsweise deren Beantwortung greifen naturgemäß ineinander. Das – Was wird adaptiert? – kann sich bei Software auf das Verhalten, die Struktur und die Architektur beziehen und dementsprechend ist das Wann und das Wie abhängig von dieser Entscheidung. Eine Verhaltensadaption ist beispielsweise Gegenstand von adaptiven Algorithmen, welche in Abhängigkeit von den Eingabewerten verschiedene Mechanismen benutzen, um die Aufgabe zu erfüllen. Aus dieser Definition heraus ist ersichtlich, dass es eine inhärente Adaption zur Laufzeit (das Wann) ist und es dafür keiner zusätzlichen Infrastruktur bedarf.

Strukturelle Adaptionen passen Software innerhalb von abgeschlossenen Einheiten an. Das Verändern der internen Struktur einer Komponente unter Wahrung des gleichen Verhaltens z.B. zum Zwecke der Beseitigung von Fehlern oder aus Performancegründen fällt in diese Kategorie. Um auch solche Adaptionen durchführen zu können, ist eine Infrastruktur notwendig, die die entsprechenden Aufgaben, das Ersetzen von Komponenten, erfüllen kann. Ebenso sind für die dritte Form, der architekturellen Adaption, zusätzliche Hilfsmittel notwendig, da diese Form eine Kombination der beiden erstgenannten und somit die anspruchsvollste ist, weil hier zusätzlich Abhängigkeiten zwischen Komponenten zu beachten sind.

Strukturelle sowie architektonische Anpassungen sind in zwei generelle Formen, die statische sowie die dynamische Adaption, zu unterteilen. Bei der statischen Adaption handelt es sich um die Konfigurierung des Softwaresystems, welche im Zuge der Übersetzung die Adaption auf die Zielstellung oder das Zielsystem durchführt. Die dynamische Adaption findet zur Laufzeit statt. Zu unterscheiden sind dabei die Startphase, in der die Adaption an die jeweilige Umgebung erfolgt, welche dann für die restliche Laufzeit konstant ist, oder die allgemeine Laufzeitadaption, welche zu jedem beliebigen Zeitpunkt angestoßen werden kann.

Die Laufzeitadaptionen bei den beiden letztgenannten erfolgt durch den Austausch von Softwarekomponenten auf dem System, welches durch die Tatsache erschwert wird, dass einerseits die Komponenten in einem ruhigen Zustand (nicht in Benutzung) sein müssen und andererseits eine Statusübertragung durchgeführt werden muss, um die Funktion hernach weitererfüllen zu können.

Auf Systemen mit limitierten Ressourcen ist das Überführen beziehungsweise das Austauschen von Komponenten eine Herausforderung. Im Folgenden wird zunächst die adaptive publish/subscribe Middleware FAMOUSO vorgestellt und gezeigt, welche Formen der Adaption angewendet und unterstützt werden.

3 FAMOUSO

FAMOUSO (**F**amily of **A**daptive **M**iddleware for autonom**OU**s **S**entient **O**bjects [SZ08, Sch08]) ist eine Kommunikationsmiddleware, welche die Ideen und Konzepte des Vorgängers COSMIC (**CO**operating **SM**art dev**IC**es [CKV04, SZ07]) aufgreift, konkretisiert und verfeinert. Der Fokus von FAMOUSO ist dabei auch darauf gerichtet, geeignete Mittel für die Adaption zur Verfügung zu stellen. Zum Beispiel wird die Adaption auf eine spezifische Plattform beziehungsweise ein Kommunikationsmedium ermöglicht und es werden Mechanismen bereit gestellt, die eine dynamische Anpassung an Umgebungsparameter erlauben.

Als Kommunikationsparadigma liegt FAMOUSO ein ereignis-basiertes Publisher/Subscriber Modell zu Grunde. Im Unterschied zur sonst üblichen adressbasierten Kommunikation findet die inhaltsbasierte Kommunikation Verwendung, bei der Ereignisse die Hauptabstraktion des Interaktionsmodells darstellen. Publisher und Subscriber sind Rollen, die Anwendungen bei der Kommunikation einnehmen. Je nach Ausprägung als Publisher oder Subscriber spezifizieren sie die Art des Ereignisses, welches sie produzieren beziehungsweise konsumieren. Auf dieser Grundlage bietet FAMOUSO ein spontanes und dynamisches n-zu-m Kommunikationverhalten, und es werden keine impliziten Annahmen bezüglich Synchronität der Ereignisse gemacht. Die Kommunikationform ist asynchron und verhindert Kontrollflussabhängigkeiten, welches die Autonomie der Kommunikationsteilnehmer bewirkt.

Die Hauptabstraktion der Kommunikation – das Ereignis – kann durch zwei Stimuli erzeugt werden. Einerseits kann der Trigger für die Erzeugung ein spontanes Hardwareereignis beziehungsweise andererseits ein periodisches sein. Im ersten Fall könnte ein Sensor ein externes Ereignis detektieren und eine Unterbrechung auslösen und im anderen Fall kann ein Ereignis ebenso 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 Ereignisses sind FAMOUSO-Ereignisse aus drei Teilen aufgebaut:

1. einem *Subjekt*, repräsentiert durch einen eindeutigen 64-bittigen-Bezeichner (Unique Identifier), welcher den Inhalt charakterisiert,

2. dem *Inhalt* beziehungsweise den Daten selbst, z.B. der Wert einer Distanzmessung oder eine Temperatur und

3. zusätzlichen *Attributen* z.B. Position, Zeit, Gültigkeit, welche optional sind

Ein Subjekt hat eine globale Bedeutung über alle Teilnetze hinweg, weshalb Subjekte einen global eindeutigen Adressraum bilden. Dieser Umstand wird genutzt, um zwischen verschiedenen Teilnetzen eine Kommunikation zu ermöglichen und diese zu steuern. Die Eindeutigkeit der Subjekte wird ausgenutzt, um erstens den Informationsfluss an Netzwerkgrenzen zu filtern, falls ein Subjekt nur innerhalb eines spezifischen Teilnetzes angefordert wurde und zweitens es entsprechend weiter zu propagieren, wenn es außerhalb abonniert wurde.

Aus der Sicht des Anwendung und für viele Einsatzzwecke wäre die Definition der Ereignisse und deren direkte Benutzung ausreichend. Im Bereich der eingebetteten Systeme findet man jedoch häufig Anforderungen bezüglich Echtzeitfähigkeit und Zuverlässigkeit. Um hier die Anwendbarkeit von FAMOUSO ebenfalls zu ermöglichen, verwendet FAMOUSO neben den beschriebenen Ereignissen auch Ereigniskanäle als Abstraktion für die Ereignisübertragung. Ein Ereigniskanal ist grundsätzlich unidirektional, wodurch die Rolle als Publisher beziehungsweise als Subscriber auch klar erkennbar ist. Die Definition eines Ereigniskanals ist dem eines Ereignisses ähnlich und gliedert sich ebenfalls in drei Teile:

1. das *Subjekt* ist das selbe und entspricht den korrespondierenden Ereignissen,

2. *Attribute* beschreiben Disseminationseigenschaften wie Periode, Frist, maximale Latenz, Zuverlässigkeit

3. und *Callbacks* definieren z.B. für den Subscriber den Ausnahmehandler, welcher aufgerufen wird, falls ein Ereignis nicht rechtzeitig entsprechend der spezifizierten Attribute eintrifft bzw. den Benachrichtigungshandler, wenn ein Ereignis eingetroffen ist. Für den Publisher existiert nur der Ausnahmehandler, der aktiviert wird, wenn das Ereignis nicht entsprechend der Attribute veröffentlicht werden konnte.

Mit Hilfe von Ereigniskanälen kann die Anwendung folglich die Übertragungseigenschaften spezifizieren und zweitens bieten sie der Middleware die Möglichkeit der Reservierung der notwendigen lokalen sowie netzwerkseitigen Ressourcen, sobald ein Ereigniskanal erzeugt wird. Außer den Ressourcen wird bei der Erzeugung eines Ereigniskanals auch das Binden des Subjektes an eine spezifische Netzwerkadresse sowie im Falle von Echtzeitanforderungen eine Bandbreitenreservierung durchgeführt. Dies geschieht jedoch vollständig transparent für die Anwendung und FAMOUSO verbirgt somit die Heterogenität der grundlegenden Netzwerke.

Beim Einsatz der Middleware läuft typischerweise auf jedem Knoten eine Instanz von FAMOUSO, die sich beim Start automatisch in das bestehende Netzwerk integriert und die von der Anwendung benutzten Ereigniskanäle einrichtet. FAMOUSO wurde speziell, jedoch nicht ausschließlich, für den Einsatz und die Kooperation zwischen intelligenten Sensoren und Aktoren

entwickelt. Die Middleware ist auf einer breiten Hardwarebasis angefangen von 8-Bit Mikrocontrollern (z.B. Atmel AVR), über 16-Bit Mikrocontroller (z.B. Motorola HC12) jeweils nativ bis hin zu 32-Bit Systemen unter Linux und Windows einsetzbar und kann über verschiedene Kommunikationsnetze z.B. CAN (Controller Area Network [Rob91], UDP-Multicast, ZigBee [Zig03], AWDS [AWD08] interagieren.

3.1 Architektur

Die Architektur von FAMOUSO ist in Abbildung 1 schematisch dargestellt. Zu erkennen ist, dass es sich um eine Schichtenarchitektur handelt, bei der ausgehend von der unteren Schicht eine zunehmende Verfeinerung und Erhöhung des Abstraktionsniveaus in Richtung Anwendung auftritt. Die Anwendung sieht, unabhängig von der Konfiguration, nur das einheitliche Interface bestehend aus Ereigniskanälen für Publisher und Subscriber. Entsprechend der Rolle der Anwendungen verwenden sie eine der Varianten oder beide. Die Darstellung zeigt ebenfalls, dass die Anwendungen in verschiedenen Sprachen (C/C++, C#, Python, Java) programmiert werden können beziehungsweise der Einsatz und Zugriff aus Ingenieurssystemen (Matlab, Simulink, Labview) möglich ist.

In den einzelnen Schichten werden jeweils spezielle Funktionalitäten angeboten. Auf der Ebene des `Event Layer` werden die Ereigniskanäle durch den *Event Channel Handler (ECH)* verwaltet, der neben der lokalen Kommunikation auch die Verwaltung der notwendigen Ressourcen und das Sicherstellen der Qualitätsanforderungen erbringt. Anwendungen können im Falle von Echtzeitqualitätseigenschaften zum Beispiel einer Frist von 50ms für Ereigniskanäle spezifizieren und wenn jene überschritten wird, ruft die Middleware einen Callback in der Anwendung für diesen Ereigniskanal auf. Dieser Infrastrukturmechanismus der Middleware erlaubt Verletzungen von Quality-of-Service-Anforderungen anzuzeigen, wodurch Anwendungen die Möglichkeit erhalten, sich zu adaptieren. Der Kommunikationsgegenstand im `Event Layer` ist das bereits vorgestellte Ereignis (siehe Abschnitt 3).

Der folgende Layer, der `Abstract Network Layer`, ist für die Funktionalität verantwortlich, die für mehrere Netzwerke gleichermaßen bereitgestellt werden kann. Hier ist zum einen das Fragmentierungsprotokoll zu nennen, welches sich den entsprechenden zu kommunizierenden Ereignissen sowie den Gegebenheiten der Netzwerkschicht anpasst. Andererseits erbringt die Schicht die Echtzeitkommunikationseigenschaft für zu sendende Ereignisse, die hier allgemein als Nachrichten bezeichnet werden.

In der untersten Schicht, dem konkreten `Network Layer`, wird die Funktionalität gekapselt, die absolut spezifisch für das jeweilige Netzwerk steht. Das hier angesiedelte Bindeprotokoll, welches die Abbildung des Subjektes (64-bittiger eindeutiger Bezeichner) auf Netzwerkadressen durchführt, muss im Gegensatz zum CAN-Netzwerk bei UDP-Multicast eine unterschiedliche Arbeitsweise zeigen. Demzufolge ist es ein spezialisiertes Protokoll für das Netzwerk. Mit dem Konfigurationsprotokoll verhält es sich genauso, da einerseits im IP-Netz eine entsprechende Adresse gefragt ist und andererseits bei CAN immer eindeutige Kommunikationsidentifier bestehen. Folglich existiert für jedes unterstützte Netzwerk eine Netzwerkanbindung in Form eines spezialisierten `Netzwerk Layer`, der die oberen Schichten mit den Funktionalität für die Kommunikation anbindet.

Die Darstellung 1 zeigt den prinzipiellen Aufbau der Middleware FAMOUSO für einen Kno-

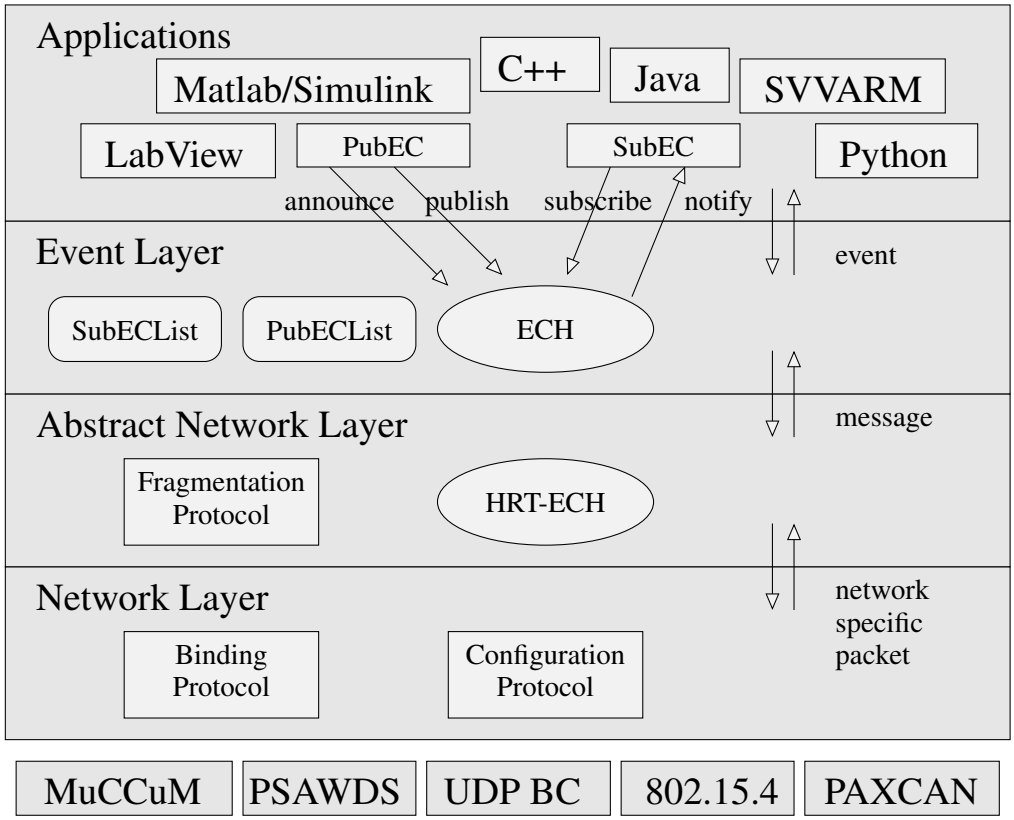

Abbildung 1: FAMOUSO – Schichtenarchitektur mit Anbindungsmöglichkeiten aus Anwendungssicht sowie Netzwerkfähigkeiten

ten, der genau eine Kommunikationsschnittstelle besitzt. Für den Fall eines Gateways, welches als Mittler zwischen Netzen fungiert, ist die Struktur aus Sicht der Anwendung identisch. Aus systemischer Sicht der Middleware spaltet sie sich unterhalb des `Event Layer` auf und kommuniziert mit Hilfe der `Netzwerk Adapter` mit zwei Kommunikationsschnittstellen. Der `Netzwerk Adapter` kann aber rekursiv angewendet werden, weshalb beliebig komplexe Strukturen erzeugt werden können (siehe Konfigurationsbeschreibung 2).

3.2 Statische Adaption

Der Begriff der statischen Adaption kann mit dem Begriff der Konfiguration des Systems zusammengefasst werden, für die es mehrere Arten der Umsetzung gibt. Unser Konfigurationsklassenansatz bedient sich der durch die Programmiersprache zur Verfügung gestellten Mittel. Der Ansatz kann auch als Einsprachenansatz bezeichnet werden, da keine externen zusätzlichen Beschreibungssprachen, Interpreter oder Übersetzer für die Umsetzung notwendig sind. Insgesamt bedient sich der Konfigurationsansatz extensiv der Mechanismen der Template-(Meta-)-Programmierung, um ein optimal angepasstes und nur die notwendige Funktionalität enthaltenes System zu erzeugen. Das dieser Ansatz mit Templates nicht notwendigerweise mit dem Aufblä-

Listing 1: Konfigurationbeschreibung für den AVR (at90can128) mit Anbindung an ein CAN-Netzwerk

```
class config {
    typedef NLCAN < PAXCAN < canary > >     NL_CAN;
    typedef AbstractNetworkLayer < NL_CAN > ANL_CAN;
    typedef EventLayer < ANL_CAN >          EL;
public:
    typedef PublisherEventChannel  < EL >   PEC;
    typedef SubscriberEventChannel < EL >   SEC;
};
```

hen der Systeme einhergeht, zeigen Arbeiten von Schulze [Sch02] oder Goldthwaite [Gol04].

Der Quellcode 1 zeigt z.B. die Konfiguration einer einfachen Variante der Middleware, wie sie für die AVR-Plattform innerhalb eines CAN-Netzwerkes aussieht. Es ist zu erkennen, dass die elementaren Module in einer Art Baukastensystem ineinander gesteckt werden. Beginnend mit der spezifischen Netzwerkschicht wird der für den AVR entwickelte CAN-Treiber *canery* in das generische *PAXCAN*-Interface eingehängt, welches wiederum mit dem *CAN Network Layer* verbunden wird. Die Konfiguration des *Config*- und *Binding Protocol* sowie anderer Protokolle in den übrigen Schichten wurde hier aus Gründen der Übersichtlichkeit weggelassen. Diese werden aber ähnlich beschrieben und mit den Schichten verbunden.

Weiterführend werden die Schichten stackartig verbunden und schlussendlich die Ereigniskanäle für die Publikation sowie für das Abonnieren von Ereignissen definiert. Eine Anwendung kann in der Folge Instanzen von diesen Typen erzeugen und in die Kommunikation eintreten.

Dem generellen Aufbau der Konfiguration nach diesem Schema folgend werden auch komplexere Systeme wie Gateways spezifiziert. In der Darstellung 2 ist die Konfigurationsbeschreibung für ein Gateway zwischen drei Netzwerken dargestellt, wobei zwei CAN-Busse sowie ein UDP-Netzwerk zusammengeschaltet werden. Hier soll erwähnt werden, dass Gateways nur selektiv Ereignisse weiterleiten, wenn diese in dem entsprechenden Unternetzwerk auch angefordert wurden. Die Gatewayfunktionalität wird durch entsprechende Ereigniskanäle erbracht, die dynamisch angelegt werden, sobald eine Anforderungen/Abonnement erfolgt.

Die Verknüpfung der verschiedenen Netzwerke nehmen die *Network Adapter* wahr. Ein *Netwerk Adapter* kann zwei abstrakte Netzwerke aufnehmen und diese unter einem Bild zusammenführen und beispielsweise dem *Event Layer* einheitlich präsentieren. Da die *Network Adapter* selbst stapelbar sind, sind komplexe Konstruktionen wie die beschriebene Gatewaykonfiguration einfach herstellbar.

Aus der Sicht der Anwendung ist es aber wiederum vollkommen transparent, ob sie sich auf einem Knoten mit einer Netzanbindung oder auf einem Gateway befindet. Sie verwendet immer einfach die Abstraktionen der Ereigniskanäle.

3.3 Dynamische Adaption

Die Middleware FAMOUSO unterstützt neben der beschriebenen statischen Adaption die Adaption zur Laufzeit. Im Allgemeinen wird auf den ressourcenbeschränkten Systemen selten die

Listing 2: Konfigurationbeschreibung eines komplexen Gateways mit zwei CAN-Netzwerken und einer UDP-Multicast Anbindung

```
class config {
    typedef MuCCuM < deviceUDP >                          NL_UDP;
    typedef NLCAN < PAXCAN < SocketCAN, deviceCAN0 > > NL_CAN0;
    typedef NLCAN < PAXCAN < SocketCAN, deviceCAN1 > > NL_CAN1;
    typedef AbstractNetworkLayer < NL_UDP >             ANL_UDP;
    typedef AbstractNetworkLayer < NL_CAN0 >            ANL_CAN0;
    typedef AbstractNetworkLayer < NL_CAN1 >            ANL_CAN1;
    typedef NetworkAdapter < ANL_CAN0, ANL_CAN1 >       GW_CAN;
    typedef NetworkAdapter < ANL_UDP, GW_CAN >          GW_UDP_CANs;
    typedef EventLayer     < GW_UDP_CANs >              EL;
public:
    typedef PublisherEventChannel<EL>                   PEC;
    typedef SubscriberEventChannel<EL>                  SEC;
};
```

Notwendigkeit bestehen, eine Umkonfiguration der Struktur beziehungsweise der Architektur durchzuführen. Nichtsdestotrotz wurden bereits Schritte in unserer Arbeitsgruppe in diese Richtung unternommen und beispielsweise die Möglichkeit erkundet, auch auf tiefst eingebetteten Systemen wie dem AVR [Atm] (4kB RAM, 4kB EEPROM, 128kB FLASH, 16MHz) Systemkomponenten auszutauschen [Die07]. FAMOUSO bietet hier die Möglichkeit, Anwendungen sowie die Middleware auszutauschen. Konzeptionell sind auch kleinere Einheiten möglich, aber aus Ressourcengründen wird momentan diese grobe Einteilung gewählt, da die Verwaltung vieler kleiner Einheiten auf diesen limitierten Systemen zu schwergewichtig ist. Im Allgemeinen muss auf den ressourcenbeschränkten Systemen im Anschluss an den Austausch ein Neustart systembedingt durchgeführt werden, um die Adaptierungen beziehungsweise ausgetauschten Komponenten wirken zu lassen.

Generell wird aber die dynamische Verhaltensadaption in verschiedenen Schichten bis hin zur Anwendung verwendet. Auf der Anwendungsebene ist beispielsweise der Callback-Mechanismus ausschlaggebend, um auf Abweichungen im spezifizierten Kommunikationsverhalten reagieren zu können. Folglich bietet die Middleware der Anwendung kontextgewahre Möglichkeiten. Eine einfache Adaptionsform auf das Verletzen einer Deadline könnte in Echtzeitumgebungen das Einnehmen eines sicheren Zustandes (fail-safe) sein.

Abgesehen von den Anwendungen arbeitet auf der Ebene der Middleware z.B. das Konfigurationsprotokoll (Startphase) und das Bindeprotokoll (jeweils beim Einrichten eines Ereigniskanals) adaptiv, da diese es erlauben, einerseits Knoten dynamisch in bestehende Kommunikationsnetze zu integrieren und andererseits die Bindung von Ereigniskanälen auf netzwerkspezifische Bezeichner zu vollbringen.

Ein weiteres Protokoll welches sich adaptiv darstellt, ist das *Fragementation Protocol*. Dieses ist im besonderen Maße interessant, da es sich sowohl statisch als auch dynamisch adaptiert. Die statische Anpassung erfolgt währende der Konfigurations- beziehungsweise Übersetzungszeit, da hier Informationen aus den spezifischen Netzwerken (z.B. Paketgröße) herangezogen werden, um die Parameter des Protokolls für dieses Netzwerk festzulegen. Die dynamische Komponente

tritt hinzu, wenn sich das Protokoll den zu übermittelnden Nachrichten anpasst, da es versucht immer möglichst effizient mit der verfügbaren Netzwerknutzlast umzugehen. Folglich ist kein statisches Format vorgegeben, sondern am Anfang einer fragmentierten Übertragung wird das Format definiert, mit der die Nachricht ausgeliefert wird.

4 Verwandte Arbeiten

Für die Darstellung der verwandte Arbeiten muss zwischen dem Bereich Publish/Subscribe Systeme und Adaption unterschieden werden.

Die Literatur kennt eine Vielzahl unterschiedlicher Publish/Subscribe-Systeme. Vielen gemein ist, dass sie meist auf recht homogener Hardware laufen und oft recht hohe Ressourcenanforderungen stellen. Vertreter aus diesem Gebiet sind z.B. SIENA [CRW01], READY [GKP99] oder HERMES [PB02], welche zwar ein skalierbares Publish/Subscriber-System bieten, aber ein statisches verteiltes Ereignis-Broker-Servernetzwerk mit zuverlässige TCP-Kommunikation als Rückgrat voraus setzen. Die damit verbundene Schwergewichtigkeit schließt freilich den Einsatz auf Mikrocontrollern aus. Darüberhinaus sind diese Systeme nicht selbstorganisierend, sondern erfordern für den Aufbau der Backbone-Server administrativen Einsatz.

Der von der Object Management Group (OMG) spezifizierte Data Distribution Service for Real-time Systems [OMG07] ermöglicht ebenfalls eine Publish/Subscribe-Kommunikation und in einigen Umsetzungen bzw. Konfigurationen sind keine Backbone-Server erforderlich. Dennoch ist diese Spezifikation nicht anwendbar, da auf der Kommunikationsebene allein Ethernet bzw. IP gesprochen werden muss. Neben der Schwergewichtigkeit und dem Ausschluss anderer Kommunikationsmedien ist dieser Standard daher nicht anwendbar.

Das hier vorgestellte System FAMOUSO erlaubt hingegen den Einsatz auf einem breiten Hardwarespektrum angefangen von Mikrocontrollern bis hin zu Standard-IT sowie verschiedenen Kommunikationsmedien unter Beibehaltung der gleichen Kommunikationsschnittstelle für die Anwendung. Der Übergang zwischen den Systemen und Netzwerken ist auf Anwendungsebene zudem vollständig transparent.

Im Bereich der Adaption gibt es ebenfalls eine reiche Auswahl an Literatur. Eine oft genutzte Möglichkeit für die Konfigurierung ist die bedingte Kompilierung unter Zuhilfenahme von Preprozessoren. Dies führt im Allgemeinen aber zu einer sehr unübersichtlichen, schwer zu wartenden und erweiterbaren Quelltextbasis [SC92]. Andere Ansätze auf diesem Gebiet verwenden beispielsweise feature-orientierte Techniken [KCH+90, FAM08], welche auf externe Beschreibungen für die Komposition von Systemen setzen. Im Gegensatz dazu verwendet der hier vorgestellte Ansatz einzig die dem C++-Standard [Str97] innewohnenden Mechanismen und Sprachkonstrukte, weshalb sie durch jeden standardkonformen Übersetzer handhabbar sind.

5 Fazit und Ausblick

Diese Arbeit präsentiert die publish/subscribe Middleware FAMOUSO, welche auf sehr heterogener Hardware läuft, eine breite Palette an Kommunikationsmedien unterstützt und Bindungen zu verschiedenen Programmiersprachen und Ingenieurssystemen bietet, wodurch der Anwendungsentwickler einen großen Spielraum der einsetzbaren Mittel hat. Für die Bereitstellung die-

ser Vielfältigkeit unter Wahrung der immer gleichen Kommunikationschnittstelle bedarf es der Adaption innerhalb der Middleware. Um die Kommunikationsmechanismen von FAMOUSO auch auf ressourcenbeschränkten Systemen einsetzbar zu machen, wird sich vielfach der statischen Adaption bedient. Hier bietet FAMOUSO alle Formen der statischen Adaption angefangen vom Verhalten über die Struktur bis hin zur Architektur an.

FAMOUSO bietet ebenfalls Mechanismen für die dynamische Adaption an. Einerseits werden dynamische Verhaltensadaptionen in der Middleware selbst angewendet und andererseits werden Mittel zur Verfügung gestellt, die eine Überwachung von spezifizierten Qualitätsanforderungen ermöglichen. Die Middleware ist entsprechend den Anforderungen gewahr und sie informiert Anwendungen, wenn diese verletzt werden. Anwendungen können daraufhin Selbstadaptionen durchführen oder Adaptionen der Middleware anstoßen.

FAMOUSO befindet sich in ständiger Weiterentwicklung. Geplante Erweiterungen sind beispielsweise ein Werkzeug zur Verfügung zu stellen, welches es erlaubt, die Middleware gemäß einem Baukasten auch grafisch zu konfigurieren und zusammen zu setzen. Konfigurationsbeschreibungen sollen dann anhand der grafischen Repräsentation automatisch erzeugt werden. Diese Form wäre für den Anwender sehr intuitiv und außerdem weniger fehleranfällig. Wir werden auch weitere Schritte bei dem effizienten Austausch von Komponenten auf kleinsten limitierten Systemen gehen. Das Ziel ist hier, einen ähnlichen Komfort zu erreichen, wie er auf Standard-IT-Systemen anzutreffen ist, auf denen einfach eine neue Version einer Software installiert wird, welche dann umgehend Verwendung findet.

Literatur

[Atm] Atmel. 8-bit Microcontroller with 32K/64K/128K Bytes of ISP Flash and CAN Controller.
http://atmel.com/dyn/resources/prod_documents/doc7682.pdf

[AWD08] AWDS Project Homepage. 2008.
http://awds.berlios.de

[CKV04] A. Casimiro, J. Kaiser, P. Verissimo. An Architectural Framework and a Middleware for Cooperating Smart Components. In *ACM Computing Frontiers conference (CF'04)*. Pp. Pages: 28 – 39. Ischia, Italy, April 2004.
http://portal.acm.org/citation.cfm?id=977098

[CRW01] A. Carzaniga, D. S. Rosenblum, A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Trans. Comput. Syst.* 19(3):332–383, 2001.
http://portal.acm.org/citation.cfm?id=380767

[Die07] J. Diederich. Unterstützung von in situ (Re)Konfiguration eingebetteter Systeme. Master's thesis, Otto-von-Guericke-Universität Magdeburg, 2007.

[FAM08] FeatureC++ Project Website. online, http://wwwiti.cs.uni-magdeburg.de/iti_db/forschung/fop/featurec/, 2008.

[GKP99] R. Gruber, B. Krishnamurthy, E. Panagos. The architecture of the READY event notification service. 1999.
citeseer.ist.psu.edu/gruber99architecture.html

[Gol04] L. Goldthwaite. C++ Performance. Technical report, International Organization for Standardization, Geneva, Schwitzerland, Juli 2004. (TR 18015:2004(E)).

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University, Software Engineering Institute, Pittsburg, PA, USA, November 1990. (CMU/SEI-90-TR-21).

[OMG07] OMG. *Data Distribution Service for Real-time Systems Version 1.2*. Object Managment Group, 1. January 2007.
http://www.omg.org/cgi-bin/apps/doc?formal/07-01-01.pdf

[Par76] D. L. Parnas. On the Design and Development of Program Families. *IEEE Transactions on Software Engineering* SE-2(1):1–9, March 1976.

[PB02] P. R. Pietzuch, J. Bacon. Hermes: A Distributed Event-Based Middleware Architecture. In *ICDCSW '02: Proceedings of the 22nd International Conference on Distributed Computing Systems*. Pp. 611–618. IEEE Computer Society, Washington, DC, USA, 2002.

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

[SC92] H. Spencer, G. Collyer. #Ifdef Considered Harmful, or Portability Experience With C News. In *the USENIX Summer 1992 Technical Conference*. Pp. 185–197. USENIX Association Berkley, 1992.

[Sch02] M. Schulze. Maßgeschneidertes Planungswesen in Betriebssystemfamilien. Master's thesis, Otto-von-Guericke-Universität Magdeburg, Magdeburg, Fakultät für Informatik, 2002.

[Sch08] M. Schulze. FAMOUSO Project Website. online, http://famouso.sourceforge.net, 2008.

[Str97] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1997. third edition.

[SZ07] M. Schulze, S. Zug. Using COSMIC – A real world case study combining virtual and real sensors. In *Proceedings of the 5th Minema Workshop on Middleware for Network Eccentric and Mobile Applications*. Magdeburg, Germany, September 11–12 2007.

[SZ08] M. Schulze, S. Zug. Exploiting the FAMOUSO Middleware in Multi-Robot Application Development with Matlab/Simulink. In *In Proceedings of the ACM/IFIP/USENIX 9th Internatinal Middleware Conference (Middleware2008)*. Leuven, Belgium, 1-5 December 2008.

[Ten00] D. Tennenhouse. Proactive Computing. *Communications of the ACM 43* 5:43–50, 2000.

[Zig03] ZigBee Alliance. *ZigBee Specification - IEEE 802.15.4*. 2003.