Management-Aufgaben bei komponentenbasierten verteilten Systemen im Fahrzeug-Telematikbereich

Volker Feil[1] und Matthias Stümpfle[2]

[1] Universität Stuttgart, Institut für Nachrichtenvermittlung und Datenverarbeitung, Pfaffenwaldring 47, 70569 Stuttgart, Germany
feil@ind.uni-stuttgart.de

[2] DaimlerChrysler AG, Research and Technology 3, Vehicle IT-Architecture, HPC T728, 70546 Stuttgart, Germany
matthias.stuempfle@daimlerchrysler.com

Kurzfassung Der Beitrag beschreibt die Software-Architektur DANA (*Distributed Systems Architecture for Networks in Automotive Environments*), mit der komponentenbasierte verteilte Systeme für die Fahrzeugtelematik dynamisch zusammengebaut werden können. Im Gegensatz zu den verbreiteten Technologien, die ein Zusammenbau von „Off-The-Shelf"-Komponenten mit Hilfe von Tools erlauben, werden Komponenten bei DANA durch eine im System verteilte Management-Software zur Laufzeit ins System aufgenommen bzw. wieder entfernt. Hierzu muss dieses Management folgende Grundaufgaben bearbeiten: Die Anordnung von Komponenten im verteilten System, die Bereitstellung einer Funktionalität zur Komponenten-Entdeckung und die Verwaltung der Kommunikationsbeziehungen zwischen Komponenten. In diesem Beitrag wird auf Mechanismen zur Bearbeitung dieser Aufgaben eingegangen und Implementierungsaspekte, die insbesondere Ergebnisse bei einer prototypischen Realisierung einschliessen, vorgestellt.

1 Einleitung

Immer mehr Software-Dienste und -Anwendungen halten Einzug ins Fahrzeug. So gehören beispielsweise im Bereich der Fahrzeugtelematik der digitale Rundfunk, Navigation, Notruf, Verkehrsinformationsdienste, Fahrzeug-Diagnose aber auch Internet-Dienste wie E-Mail und WWW [4] entweder schon zur Standard-Ausstattung oder stehen kurz vor der Einführung.

Im nächsten Abschnitt wird die Entwicklung der für solche Anwendungen und Dienste notwendigen Software-Systeme im Telematikbereich für Fahrzeuge aufgezeigt. Die Realisierung solcher Systeme durch Software-Komponenten ist ein vielversprechender Ansatz. Durch Management-Software können Komponenten dynamisch und ohne manuelle Eingriffe zu einem System zusammengesetzt werden. In den darauf folgenden Abschnitten wird auf die wesentlichen Aufgaben dieses Komponenten-Managements eingegangen.

Komponenten müssen angeordnet, d. h. einem Gerät zugeordnet werden (Abschnitt drei). Ihnen muss es möglich gemacht werden, andere Komponenten, die gesuchte Dienste anbieten, zu entdecken (Abschnitt vier). Möchte eine Komponente den Dienst einer anderen Komponente nutzen, so muss das Einrichten einer Kommunikationsbeziehung zwischen diesen Komponenten ermöglicht werden (Abschnitt fünf).

Im sechsten Abschnitt werden schließlich relevante Arbeiten zur dynamischen Komponentenanordnung vorgestellt und mit dieser Arbeit verglichen.

2 Software-Systeme im Bereich der Fahrzeugtelematik

Schon heute ist die Software im Fahrzeug auf Rechner verteilt, die im Motor- und Komfortbereich z. B. über den CAN-Bus und im Telematikbereich über den optischen D2B-Ring [2] zur Übertragung von Audio-Daten miteinander gekoppelt sind. In Zukunft werden noch breitbandigere Übertragungsmedien eingesetzt werden. Um auch Video-Bilder darstellen zu können, wird eine Vernetzung des Fahrzeug-Telematikbereichs durch einen optischen MOST-Ring [7] oder einen IEEE 1394-Bus [14] angestrebt. Beide Übertragungsmedien erlauben es, Daten sowohl asynchron (z. B. Internet-Daten) als auch synchron/isochron (z. B. Video-Ströme) auszutauschen. Im Fahrzeug der Zukunft werden eine Vielzahl von Anwendungen, die unterschiedliche Dienstgüten (QoS, Quality of Service) erfordern, innerhalb eines heterogenen verteilten Systems koexistieren.

2.1 Evolution der Software-Systeme im Fahrzeug-Telematikbereich

Die Innovationszyklen im IT-Bereich sind so kurz, dass Anwendungen und Dienste in der Regel während der Lebensdauer eines Fahrzeuges (ca. 10 Jahre) veralten und ersetzbar werden bzw. ergänzt werden müssen. Aus diesem Grund muss die Austauschbarkeit von Diensten und Anwendungen während der Fahrzeug-Lebensdauer gewährleistet sein. Wegen dieser Hauptanforderung wird eine Entwicklung im Fahrzeug-Telematikbereich hin zu komponentenbasierten verteilten Systemen erwartet:

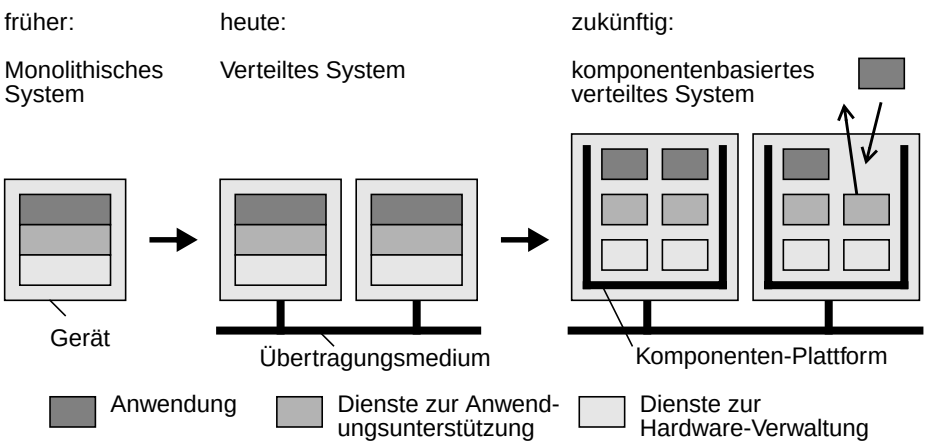

Bild 1: Entwicklung von monolithischen zu komponentenbasierten verteilten Systemen

Bild 1 zeigt, dass früher Anwendungen (und Dienste) durch ins Fahrzeug eingebaute „Stand-Alone“-Geräte realisiert wurden. Heutzutage sind Anwendungen auf mehrere Geräte verteilt, die über ein Bussystem gekoppelt sind. Die Software ist hierbei noch

statisch an die Hardware gebunden. Erst durch einen nächsten Entwicklungsschritt, bei dem die Software in einzelne Komponenten aufgeteilt wird, wird die starre Beziehung zwischen Hard- und Software aufgehoben. Somit können Software-Komponenten an die Fahrzeug-Hardware zur Systemlaufzeit gebunden werden. Das Aufnehmen (bzw. auch Entfernen) von Software-Komponenten geschieht transparent für Nutzer (Fahrer und Passagiere) durch die Existenz einer Management-Software. Im Gegensatz zu manuellen Software-Updates (z. B in der Werkstatt) ist hierbei eine Möglichkeit gegeben, das System während der Fahrzeug-Lebensdauer mit neuer Funktionalität an neue Gegebenheiten anzupassen oder zu erweitern.

2.2 Komponentenbasierte Systeme

In diesem Beitrag wird unter dem Begriff Software-Komponente binärer Code [12] verstanden, der in ein System während dessen Laufzeit aufgenommen werden kann.
Die rasche Verbreitung neuer Technologien wie z. B. dem *Enterprise Java Beans*-Konzept [17] oder dem *CORBA Component Model* [8] zeigen den wachsenden Erfolg der Komponenten-Software. Solche Technologien erlauben es, komplexe Systeme aus „Off-The-Shelf"-Komponenten zusammenzubauen. Sie ermöglichen diesen Zusammenbau z. B. durch visuell unterstützende Werkzeuge und durch die genaue Spezifikation von Schnittstellen. Darüber hinausgehend soll nun durch DANA der Zusammenbau des Systems dynamisch während der Systemlaufzeit durch eine Management-Software ermöglicht werden.

Bild 2 zeigt, mit welchen Funktionalitäten Komponenten im Telematikbereich des Fahrzeugs versehen sein können.

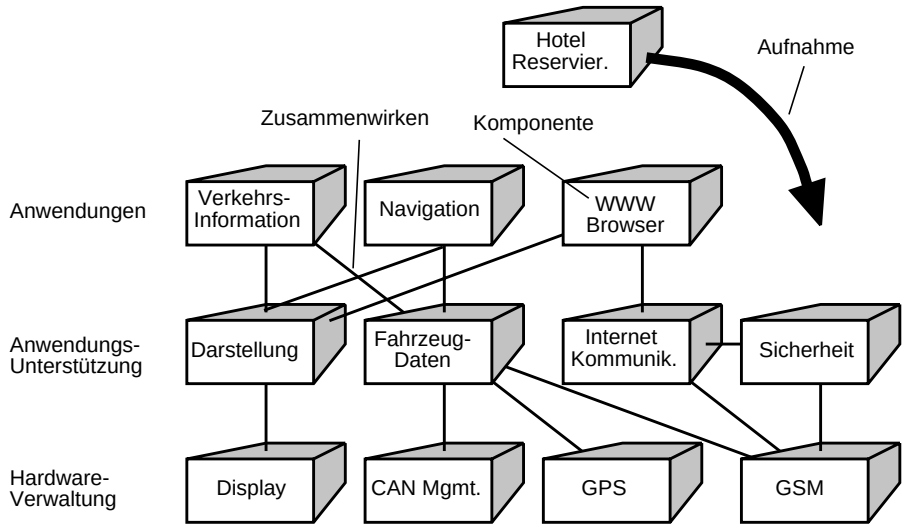

Bild 2: Software-Komponenten im Telematikbereich des Fahrzeugs

Die Komponenten, die im System verteilt sind, wirken hierbei zusammen. Beispiels-

weise bereitet die Navigations-Komponente in Bild 2 Fahrzeug-Daten auf, die wiederum vom Motorbereich (CAN-Bus) oder einem GPS-Empfänger stammen können. Die aufbereiteten Daten, werden dann an eine Komponente weitergereicht, die für die Darstellung der Daten verantwortlich ist. Diese Komponente sorgt dann z. B. dafür, dass die Daten auf dem richtigen Bildschirm dargestellt werden, falls mehrere Bildschirme im Fahrzeug installiert sind.

2.3 DANA – Eine Architektur für komponentenbasierte verteilte Systeme

Bild 3 zeigt das bei DANA [3] (*Distributed Systems Architecture for Networks in Automotive Environments*) entwickelte Komponenten-Modell.

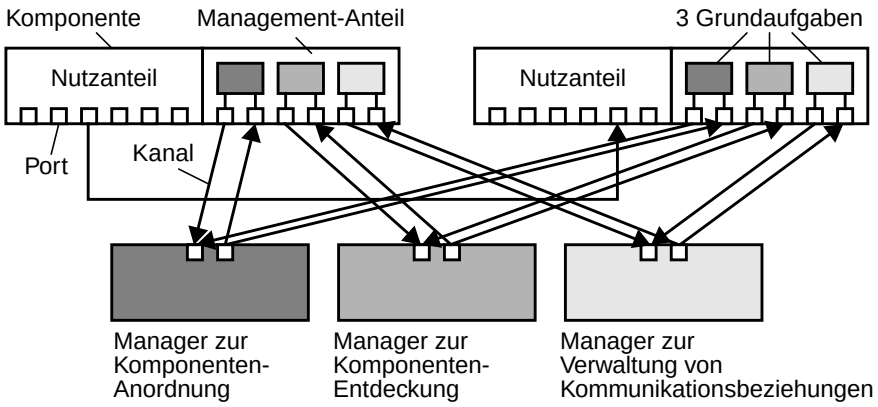

Bild 3: DANA-Komponenten-Modell

Eine Komponente besteht aus einem Nutzanteil und einem Management-Anteil. Im Management-Anteil werden für die Komponente die drei Grundaufgaben – Komponenten-Anordnung, Bereitstellung eines Mechanismus zur Komponenten-Entdeckung und Verwaltung von Kommunikationsbeziehungen – bearbeitet. Hierzu kommuniziert der Management-Anteil mit mehreren Managern, die selbst Komponenten mit speziellem Nutzanteil entsprechen. Die Kommunikation zwischen Komponenten, d. h. auch mit Managern, erfolgt bei DANA mit Hilfe von gerichteten Kanälen, die an Ports enden. Ports dienen zur Bezeichnung des Zugangs zu den angebotenen Diensten.

3 Anordnung von Komponenten

Eine Hauptaufgabe des Managements ist es, die dynamische Aufnahme neuer Komponenten ins System zu ermöglichen. In diesem Abschnitt wird diese Anordnungs-Aufgabe feiner untergliedert.

3.1 Mechanismen für die Anordnung von Komponenten

Dabei sind mehrere Aspekte zu beachten:

Speicherung. Es muss entschieden werden, ob Komponenten durch einen nicht-flüchtigen Speicher persistent auch während einer Systeminaktivität im System gehalten werden (*Caching*).

(Re-)Konfiguration. Weiter ist zu klären, welchem Gerät eine Komponente bei einem aktiven System zugeordnet werden kann, damit sie ihren Teil zur Systemfunktionalität beiträgt. Wichtige Parameter bei der Zuordnung einer Komponente zu einem Gerät sind die von der Komponente zur Laufzeit benötigten Hardware-Ressourcen, die durch andere Software-Komponenten erbrachten Dienste sowie Dienstgüteanforderungen, die bei der Ressourcenzuteilung berücksichtigt werden müssen.

Die Konfigurationsaufgabe wird immer zuerst bei der System-Initialisierung gelöst. Sie wird jedoch, während das System aktiv ist, fortlaufend neu gestellt. So erfordert ein i. a. unvorhersehbares Laufzeitverhalten der Kontrollflüsse von Komponenten ein dynamisches Balancieren der Last. Zudem müssen zur Laufzeit durch vom System (z. B. durch Komponentenabhängigkeiten oder durch Benutzer-Initiative) oder von außerhalb (z. B. durch Einspielen von Software-Aktualisierungen, *Updates*) kommende Initiativen neue Komponenten aufgenommen werden. Unter dem Begriff *Komponentenabhängigkeit* (siehe auch [6]) wird hierbei verstanden, dass aktive Komponenten im System angebotene Dienste benötigen, und deshalb die Existenz weiterer, diesen Dienst erbringende Komponenten gewährleistet sein muss.

Weiter kann sich durch die Mobilität des Fahrzeugs die drahtlose Zugangstechnologie so ändern kann, dass temporär für das Fahrzeug ein kostengünstiger Zugang zur externen Infrastruktur zur Verfügung steht. Es kann deshalb durchaus sinnvoll sein, Komponenten auf bereitstehende Rechner in der Infrastruktur auszulagern, um so die Erweiterung der Systemfunktionalität während des kostengünstigen Zugangs zu ermöglichen [9].

Sicherheit. Natürlich muss bei der Anreicherung des Telematiksystems um neue Funktionalität die Erfüllung der gegebenen Sicherheitsanforderungen gewährleistet sein. So ist bei der Aufnahme von Komponenten in bestimmte Systembereiche deren Echtheit (Authentizität) nachzuweisen. Die Verifikation der Authentizität kann prinzipiell durch einen Signierungsmechanismus für Komponenten ermöglicht werden. Die zur Überprüfung der Signatur benötigten öffentlichen Schlüssel können dann durch eine Zertifizierungs-Infrastruktur ins Fahrzeug gelangen.

3.2 Implementierungsaspekte

Java besitzt mit seinen Eigenschaften der Objekt-Serialisierung und des Klassen-Ladens das Potenzial, als einheitliche Grundlage für Komponenten-Plattformen einge-

setzt zu werden. Eine solche Vereinheitlichung ist erstrebenswert, wenn Komponenten zu einem erst zur Laufzeit bestimmten Gerät zugeordnet werden sollen.

Neue Tendenzen im Umfeld der eingebetteten Systeme zeigen, dass Java durchaus skalierbar im Verbrauch von Speicher- und Rechenleistung sein kann. So ist der minimale Speicherbedarf einer JBED-Maschine 10 kByte [13]. Deshalb ist es denkbar, dass Java im bezüglich des Verbrauchs von Speicher und Rechenleistung konservativen Fahrzeugumfeld durchaus eine wichtige Rolle übernehmen kann.

Beim realisierten Prototypen wird momentan eine auf Linux laufende JVM als Komponentenplattform verwendet. Linux ist hierbei vor allem wegen seines frei verfügbaren Quell-Codes als Entwicklungsplattform sehr attraktiv. So konnte z. B. eine bestehende Treiber-Entwicklung für den IEEE 1394-Bus [1] für den prototypischen Einsatz u. a. um einen isochronen Übertragungsdienst erweitert und modifiziert werden. Das prototypische System soll nun in einem nächsten Schritt um Plattformen erweitert werden, die voraussichtlich im Fahrzeug-Telematikbereich in Zukunft eine Rolle spielen können. So sind hier eingebettete Systeme zu nennen, die je Rechner nur eine JVM als Plattform zur Ausführung von Java Byte Code (z. B. JBED) beinhalten. Solche JVMs setzen direkt auf die Hardware ohne Unterstützung eines zusätzlichen Betriebssystems auf. Komponenten-Plattform und Betriebssystem sind zu einer Einheit verschmolzen.

4 Entdeckung von Komponenten

Um einen Dienst nutzen zu können, muss eine Komponente eine andere, diesen Dienst bereitstellende Komponente innerhalb des Systems finden.

4.1 Mechanismen für die Entdeckung von Komponenten

Eine bekannte Möglichkeit, Diensterbringer zu entdecken, besteht darin, dass diese Diensterbringer sich bei einer zentralen Stelle im verteilten System registrieren, damit eine suchende Instanz bei dieser Registratur nach einem Dienst nachfragen kann. Ein bekannter Vertreter für diesen Mechanismus ist z. B. der *Naming Service* von CORBA [15]. Bei lokal begrenzten Netzen, insbesondere wenn solche Netze auf Broadcasting-Medien (z. B. Ethernet oder IEEE 1394) beruhen, kann zudem vorteilhaft die Kommunikationsart des Broadcastings eingesetzt werden. So ermöglicht Jini [18], dass durch Broadcasting-Nachrichten nach Registraturen (*Lookup Services*) gesucht wird oder im umgekehrten Fall, dass sich die Registratur per Broadcasting bekannt macht. Auch bei UPnP [16] werden durch Broadcasting *Announce*-Pakete mit Dienst-Informationen verbreitet bzw. mit *Discovery*-Paketen nach dem Ort bestimmter Dienste gefragt.
Bei DANA tauschen spezielle Manager, sogenannte LSM (*Local Service Manager*) Dienst-Informationen per Broadcasting aus (siehe Bild 4). Diese LSM sind für die Komponenten ihres Gerätes verantwortlich. Komponenten müssen deshalb beim lokalen LSM registriert werden bzw. können bei ihm diensterbringende Komponenten erfragen.

Es gilt immer, dass der Ort der diensterbringenden Komponente im heterogenen Verteilten System durch ein einheitliches Adressierungsformat beschrieben werden muss. Folglich wird auch im DANA-Management eine von den Hardware-Adressen losgelöste, logische Adressierung verwendet.

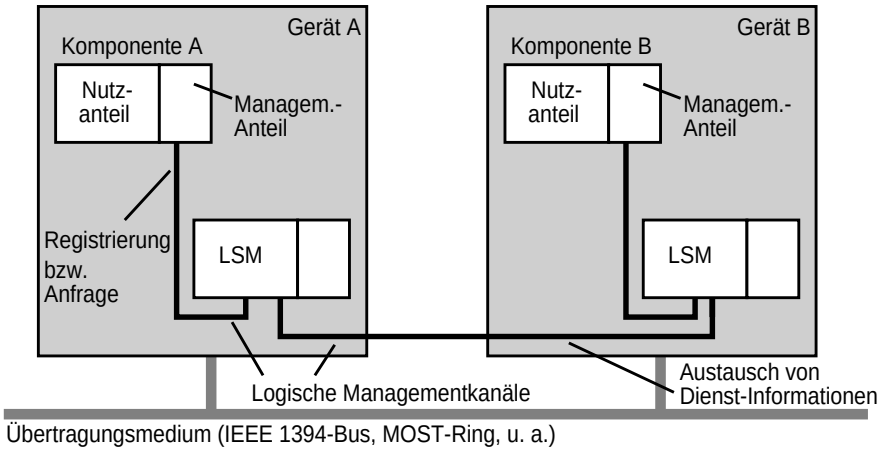

Bild 4: Austausch von Dienst-Informationen

4.2 Implementierungsaspekte

Ein Problem beim Einsatz universeller Technologien wie Jini und UPnP im Fahrzeugbereich ist, dass durch ihre Universalität nicht die im Fahrzeug häufig vorkommenden leistungsschwachen Geräte miteinbezogen werden können. Das einheitliche Adressierungsformat wird z. B. bei diesen universellen Technologien durch die Existenz der IP-Schicht erreicht, damit jedes Gerät, das dann zumindest IP-Funktionalität bereitstellen muss, durch eine IP-Adresse adressierbar ist. Bei der Realisierung des Broadcasting-Mechanismus muss, wie auch die prototypische Implementierung zeigt, jedoch lediglich auf ein einheitliches Format der Broadcasting-Nachrichten geachtet werden, so dass auf sehr einfache Weise bei der Implementierung der Management-Software auch leistungsschwache Geräte berücksichtigt werden können. Zwischen Geräten mit verschiedenen Plattformen kann durchaus ein Austausch dieser Broadcasting-Nachrichten stattfinden. So können auch Dienste im System berücksichtigt werden, die nicht von Komponenten auf den in Abschnitt 3 geforderten einheitlichen Komponenten-Plattformen angeboten, sondern durch Software bzw. Hardware auf sehr einfachen Geräten bereitgestellt werden.

5 Verwaltung von Kommunikationsbeziehungen

Damit ins System geladene Komponenten mit anderen Komponenten unter Gewährleistung einer gewünschten Dienstgüte kommunizieren können, muss eine Verwaltung von Kommunikationsbeziehungen vorgesehen werden.

5.1 Mechanismen zum Verwalten von Kommunikationsbeziehungen

Bei DANA werden Kommunikationsbeziehungen zwischen Komponenten durch logische Kanäle dargestellt. Kanäle können eingerichtet, zu Punkt-zu-Mehrpunkt-Kanäle erweitert, reduziert und wieder abgebaut werden. Das hierfür benötigte Kanal-Mana-

gement wird durch mehrere Manager realisiert. Wenn ein Übertragungsmedium vorhanden ist, müssen die betroffenen *Local Channel Manager* (LCM), die für die Ressourcen-Verwaltung in einem Gerät verantwortlich sind, und ein *Range Channel Manager* (RCM) für die Ressourcen-Verwaltung auf dem Übertragungsmedium miteinbezogen werden. Durch Protokolle handeln zwei LCM die Kanal-Modifikationen aus.

In Bild 5 initiiert der LCM in Gerät A einen Vorgang zum Einrichten eines Kanals, bei dem *Establish*-Nachrichten ausgetauscht werden. Da das Management nur logische Adressen kennt, muss an dieser Stelle im Treiber für das Übertragungsmedium eine Umsetzung auf die entsprechenden Hardware-Adressen vorgenommen werden, um die Management-Nachrichten übertragen zu können. Bei geräteübergreifenden Kanälen wird der für das Übertragungsmedium zuständige RCM durch den Austausch von *MediaAlloc*-Nachrichten miteingebunden. Er sorgt dafür, dass die für den Kanal geforderte Dienstgüte auch auf dem Übertragungsmedium gewährleistet wird, falls dies möglich ist.

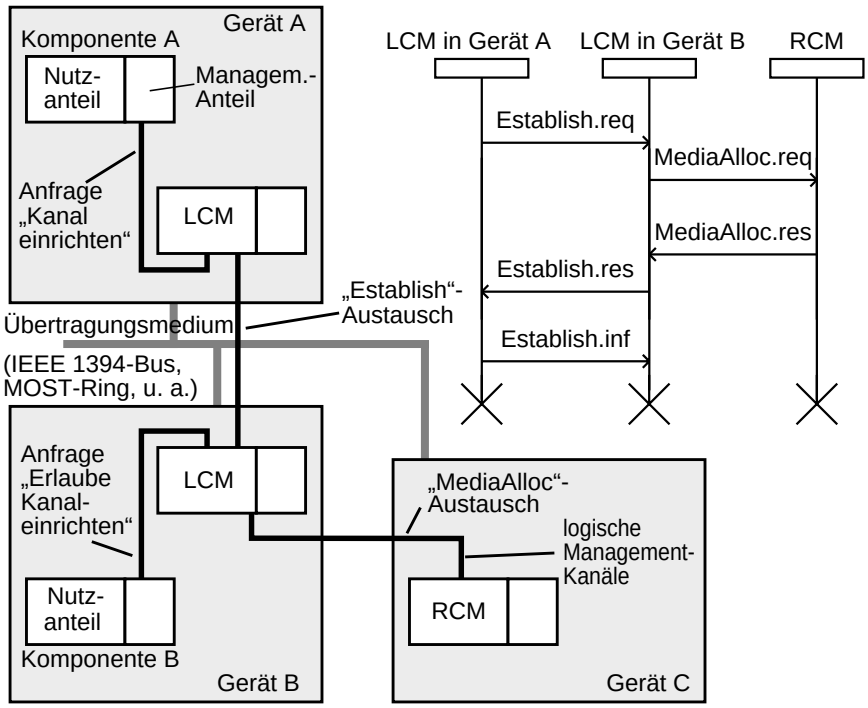

Bild 5: Kanal Einrichten

5.2 Implementierungsaspekte

Der bei der prototypischen Implementierung verwendete, getaktete IEEE 1394-Bus erlaubt es, Daten asynchron bzw. isochron in Form von Rahmen zu übertragen. Bei DANA sorgt ein RCM für die Verwaltung isochroner Ressourcen. Über ihn kann ein Knoten Bandbreite reservieren. Wenn keine Überreservierung vorliegt, wird eine erfolgreiche Bus-Arbitrierung zur Übertragung von Daten innerhalb eines Zyklusses (ca. 125µs) gewährleistet. Ein IEEE 1394-Knoten entspricht beim Prototypen einer handelsüblichen PCI-Host-Adapter-Karte.

Für die Ressourcen-Verwaltung bei der prototypischen Implementierung werden folgende Schritte beachtet:

Ressourcen-Reservierung im Initiator-Gerät. Nach der Initiierung einer Kanal-Modifikation durch eine Komponente (Initiierung „Kanal einrichten“ in Bild 5), versucht der zuständige LCM, die lokal im Gerät für diesen Kanal benötigten Ressourcen zu reservieren (bzw. freizugeben). Falls beispielsweise ein Kanal zum Senden von Daten mit einer garantierten Datenrate von mindestens 8 MBit/s eingerichtet werden soll, muss der LCM ein spezielles Bandbreitenregister im IEEE 1394-Knoten auf einen entsprechenden Wert setzen. So muss eingetragen werden, dass ein 128 Byte großer isochroner Rahmen je Zyklus versendet wird (128Byte/125µs>8MBit/s). Nach der Reservierung dieser Ressource gibt der LCM durch das Versenden einer *EstablishReq*-Nachricht die Inititative an den LCM im empfangenden Gerät weiter.

Ressourcen-Reservierung für den Bus. Der beauftragte LCM erkennt, dass die *EstablishReq*-Nachricht aus einem anderen Gerät stammt, und dass deshalb ein geräteübergreifender Kanal eingerichtet werden muss. Er leitet deshalb die Initiative unmittelbar an den für das Übertragungsmedium zuständigen RCM durch eine *MediaAllocReq*-Nachricht weiter. Bei der prototypischen Implementierung stehen RCM für Ethernet und IEEE 1394 zur Verfügung. Während der RCM für Ethernet jeden Versuch zur Bandbreitenreservierung durch eine Fehlernachricht zurückweist, sorgt der RCM für IEEE 1394 durch Verwaltung aller Bandbreite-Reservierungen dafür, dass keine Überreservierung an Bandbreite stattfindet. Danach wird die Initiative wieder an den LCM im beauftragten Gerät zurückgegeben.

Ressourcen-Reservierung im beauftragten Gerät. Da der LCM im beauftragten Gerät für das Einrichten eines Endpunktes eines empfangenden Kanals zuständig ist, muss er seinem IEEE 1394-Knoten mitteilen, welcher isochrone Kanal abgehört werden soll.

Nach dem Einrichten eines Kanals kann innerhalb jeder Komponente über einen eindeutigen Bezeichner für das Kanalende (CEI, *Channel Endpoint Identifier*) auf den an einem Port endenden Kanal zugegriffen werden. Dies bedeutet, dass eine ortstransparente Kommunikation ermöglicht wird, bei der keiner Komponente der Ort des Kommunikationspartners in Form einer Adresse bekannt sein muss. Durch die den Kanal realisierende Software, zu der auch der Treiber für das Übertragungsmedium gehört, können dann vom sendenden Kanalende aus alle empfangenden Kanalenden adressiert werden. Prinzipiell ist bei Punkt-zu-Mehrpunkt-Kanälen die Verwendung von Mulicast-Hardware-Adressen möglich.

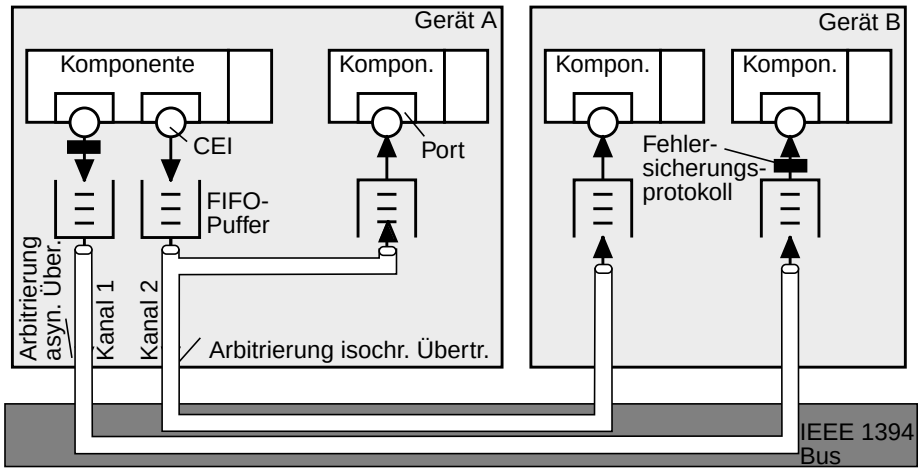

Bild 6: Realisierung von Kanälen

Bild 6 zeigt exemplarisch die Realisierung von zwei Kanälen. Über Kanal 1 können Daten ohne Bandbreitegarantie (asynchrone Übertragung) zuverlässig mit Hilfe eines Fehlersicherungsprotokolls zwischen zwei Komponenten ausgetauscht werden. Über Kanal 2 empfangen zwei Komponenten Daten von einer sendenden Komponente. Für die Übertragung der Daten auf dem IEEE 1394-Bus steht eine garantierte Bandbreite zur Verfügung.

6 Verwandte Arbeiten

In diesem Abschnitt werden relevante Arbeiten, die sich mit der dynamischen Aufnahme von Komponenten beschäftigen, vorgestellt. Zudem findet eine Abgrenzung zwischen DANA und diesen Arbeiten statt.

In [9] wird eine Internet basierte Komponentenarchitektur für Fahrzeug Client- und Serversysteme vorgestellt, die zu COSIMA [10] weiterentwickelt wurde. DANA setzt prinzipiell ebenfalls auf die Arbeiten zu dieser Architektur auf, wobei der Schwerpunkt auf die Dienst-Entdeckung und Kommunikations-Verwaltung in verteilten Systemen mit leistungsschwachen Endgeräten gelegt wird. Während die Architektur von [9] und COSIMA als eine auf das Internet aufsetzende Middleware betrachtet werden kann, greift DANA durch spezielle Treiber direkt auf das jeweilige Übertragungssystem (z. B. IEEE 1394) zu. Internet-Pakete werden folglich über einen eingerichteten DANA-Kanal ausgetauscht.

Weiter gibt es Arbeiten, die das Laden von Komponenten zu einem Client untersuchen. So bestehen bei [11] die Komponenten aus Java Beans [17], die durch ein XML-Format beschrieben werden, damit sie auf einen Ziel-Rechner geladen werden können. Im Unterschied hierzu werden Komponenten bei DANA nicht auf einen einzigen Ziel-Rechner sondern in ein verteiltes System eines Fahrzeuges geladen. Während einige Mechanismen z. B. zur Erfüllung der Sicherheitsanforderungen (Zertifizierung

von Komponenten bzw. Einführung von *Trusted Third Parties*) sich ähnlich sind, existieren bei DANA noch weitere Mechanismen hinsichtlich der Komponenten-Anordnung, Dienst-Entdeckung und Kommunikationsverwaltung im Verteilten System.

In [5] wird eine Architektur zur automatischen Konfiguration von komponentenbasierten verteilten Systemen erläutert. Das Ziel, gemäß dem sogenannten *WYNIWYG*-Paradigma (*What You Need Is What You Get*) nur den minimal notwendigen Satz von Komponenten ins System zu laden, ist auch bei DANA vorhanden. Zudem werden Grundvoraussetzungen für das Laden von Komponenten benannt, die auch für DANA gelten: Die von der Komponente benötigten Hardware-Ressourcen (Beschaffenheit und Kapazität) und die zu nutzenden Dienste, die durch Software erbracht werden, müssen vorhanden sein. Hierbei wird die Reservierung lokaler Hardware-Ressourcen (CPU, Speicher) bei der Komponenten-Anordnung berücksichtigt. Im Unterschied dazu billigt man bei DANA jedoch auch dem Kommunikationsmedium reservierbare Ressourcen zu. Man geht also keineswegs von einem „Best Effort"-Netz (z. B. Internet) aus. Die Folge davon ist, dass bei DANA eine Kommunikationsbeziehung zwischen dynamisch angeordneten Komponenten zuerst erfolgreich eingerichtet werden muss. Weiter bildet in [5] CORBA mit seinen Diensten (z. B. *Naming Service*) die Grundlage sowohl für die Architektur als auch für die Implementierung im verteilten System. Bei DANA werden hingegen nur grundlegende Mechanismen und Austauschformate von Management-Nachrichten spezifiziert. Die Implementierung selbst kann für jedes Gerät optimiert werden.

7 Zusammenfassung und Ausblick

In diesem Beitrag wurde die komponentenbasierte Architektur für verteilte Systeme im Fahrzeug DANA vorgestellt. Es wurde ein Komponentenmodell eingeführt, das auf der Bearbeitung von Grundaufgaben zur Komponenten-Anordnung und -Entdeckung und zur Verwaltung von Kommunikationsbeziehungen zwischen Komponenten durch ein Management beruht.

In einem nächsten Schritt soll die Architektur anhand der Einbeziehung von eingebetteten Systemen bei der prototypischen Implementierung weiter validiert und hinsichtlich der Berücksichtigung der dort vorherrschenden Randbedingungen überprüft werden. Während die Mechanismen zur Komponenten-Entdeckung und Kommunikationsverwaltung durch DANA genau spezifiziert sind, ist die Spezifikation der Mechanismen zur Komponenten-Anordnung Gegenstand der aktuellen Forschung bei DANA. Ziel ist es, eine umfassende Architektur für komponentenbasierte verteilte Systeme für den Fahrzeug-Telematikbereich zu erhalten.

8 Literatur

1. A. Bombe, S. Rougeaux, E. Pirker: The Linux 1394 project, http://linux1394.sourceforge.net/
2. C. Ciocan: The Domestic Digital Bus System (D2B). IEEE Transactions on Consumer Electronics, vol. 36, no. 3, (1990) 619-622
3. V. Feil, U. Gemkow, M. Stümpfle: A Vehicular Software Architecture Enabling Dynamic Alterability of Services Sets, In: C. Linnhoff-Popien, H.-G. Hegering (ed.): USM 2000, LNCS 1890, München (2000) 68-80
4. A. Jameel, M. Stümpfle, D. Jiang, A. Fuchs: Web on Wheels: Toward Internet-Enabled Cars. IEEE Computer, vol. 31 (1998) 69-71
5. F. Kon: Automatic Configuration of Component-based Distributed Systems. PHD Thesis, University of Illinois at Urbana-Champaign (2000)
6. F. Kon, R. H. Campbell: Dependence Management in Component-Based Distributed Systems. IEEE Concurrency, 8(1), (2000) 26-36
7. P. König, C. Thiel: Media Oriented Systems Transport. it + ti – Informationstechnik und Technische Informatik, vol. 5 (1999) 36-42
8. J. Mischkinsky (ed.): CORBA 3.0 New Components Chapters. OMG, CORBA Component Model FTF Draft (1999)
9. M. Stümpfle, A. Jameel: Eine Inernet basierte Komponentenarchitektur für Fahrzeug Client- und Serversystemes. In: R. Steinmetz (ed.): Kommunikation in Verteilten Systemen (KiVS), 11. ITG/GI-Fachtagung (1999) 6 - 19
10. M. Stümpfle, et. al.: COSIMA – A Component System Information and Management Architecture. IEEE Intelligent Vehicles Symposium 2000, Dearborn, USA (2000)
11. S. Rudkin, A. Smith: A Scheme for Component Based Service Deployment. In: C. Linnhoff-Popien, H.-G. Hegering (ed.): USM 2000, LNCS 1890, (2000) 68-80
12. C. Szyperski: Component-Software. Addison-Wesley, New York (1997)
13. J. Tryggvesson, T. Mattson, H. Heeb: JBED: Java for Real-Time Systems. Dr. Dobb's Journal, Miller Freemann (1999)
14. IEEE: Standard for a High Performance Serial Bus. IEEE Std. 1394-1995 (1996)
15. OMG: Naming Service Specification. CORBA Services V. 1.0 (2000)
16. UPnP Forum: www.upnp.org
17. Sun Microsystems, Inc.: Java Beans Specification, V. 1.01 (1997)
18. Sun Microsystems. Inc.: Jini Architecture Specification, V 1. 01 (1999)