Beschreibung von Kommunikationsszenarien in heterogenen automotiven Systemen

Matthias Stümpfle

Institut für Nachrichtenvermittlung und Datenverarbeitung
Prof. Dr.-Ing. P. Kühn
Universität Stuttgart
Seidentraße 36, D-70174 Stuttgart
Telefon: +49-711-121-2485 Fax: +49-711-121-2477
email: stuempfle@ind.e-technik.uni-stuttgart.de

Zusammenfassung

In automotiven Systemen wie Personenwagen, Lastkraftwagen usw. werden heute serielle Bussysteme eingesetzt, um den Verkabelungsaufwand und das Gewicht des Fahrzeugs zu reduzieren. In den daraus entstehenden Netzen sind elektronische Komponenten unterschiedlicher Hersteller miteinander verbunden. Deshalb müssen für die Kommunikation dieser Komponenten offene Schnittstellen bereitgestellt werden, die auf entsprechenden Softwareschichten aufbauen.
In diesem Beitrag wird eine Methode vorgestellt, mit der es möglich ist, standardisierte Kommunikationsszenarienbeschreibungen anzugeben und somit den Designprozeß für ein derartiges System, unter der Berücksichtigung der Randbedingungen im automotiven Sektor, zu steuern. Ziel ist es dabei, außer der Standardisierung, die verwendete Hardware so transparent wie möglich und dadurch die Benutzung einfach zu machen.

1 Einleitung

In automotiven Systemen wie Personenwagen, Lastkraftwagen usw. vollzieht sich eine Entwicklung zu immer komplexeren elektronischen Systemen. Insbesondere der Wunsch nach mehr und intelligenterer Kommunikation zwischen einzelnen elektronischen Komponenten wächst stetig, sowohl im Komfortelektronikbereich (z.B. Sitzsteuerung oder Klimaanalage) als auch in der Steuer-und Regelelektronik (z.B. Motorsteuerung). Kabellängen über 2 km sind deshalb bei heutigen Standardlösungen, bei denen Kommunikationsteilnehmer jeweils direkt miteinander verbunden sind, keine Seltenheit. Diese Verkabelungsart verursacht Kosten, erhöht unnötig das Fahrzeuggewicht und erschwert die Wartung. Ergänzungen und Erweiterungen sind aufgrund hoher Packungsdichten in Kabelsträngen nur schwer oder überhaupt nicht realisierbar.
Ein erster Schritt um diese Nachteile aufzulösen, war die Einführung serieller Bussysteme, die im Zeitmultiplex genutzt werden können. Es entstehen dadurch lokale Netze, die Komponenten unterschiedlicher Zulieferer und Hersteller miteinander verbinden. Daraus folgt, daß für das Design und die Implementierung eines Kommunikationssystems auf dieser Basis offene standardisierte Schnittstellen vorhanden sein müssen. Diese Schnittstellen werden, da sie nicht nur für die Applikationsanbindung sondern auch für die Hardwareanbindung über

entsprechende Treiber verantwortlich sind, auf einer wohldefinierten Software-Schichtung aufbauen. Dadurch wird es möglich, neue Komponenten in ein bestehendes Netz aufzunehmen. Die klare Gliederung der Software ist insbesondere deshalb wichtig, da durch sie und den damit realisierten Diensten, in Zukunft ein entscheidender Anteil der Wertschöpfung eines Fahrzeugs erzielt werden.
Auch an anderen Stellen wird nach geeigneten Beschreibungsformen gesucht. Die Vereinigung „CAN in Automation", die sich auf die Anwendung des CAN-Busses spezialisiert hat und in der hauptsächlich die Automatisierungsindustrie vertreten ist, spezifizieren momentan, ausgehend von einer Art Applicationlayer, Dienste für den automotiven Bereich [1]. Allgemein kann festgehalten werden, daß im automotiven Bereich der OSI-Protokollstack auf die Schichten 1,2 und 7 reduziert ist [2].
Schwarz [3] geht einen ähnlichen Weg. Er erweitert Hochsprachen, wie sie für die Mikrocontrollerprogrammierung angeboten werden, um Befehle für die Kommunikation. Eine weitere Aktivität zeichnet sich im Moment unter den Kraftfahrzeugherstellern und der Zulieferindustrie ab. Unter dem Namen OSEK sollen hier Echtzeitbetriebssystemdienste standardisiert werden. Basierend auf diesen Diensten lassen sich entsprechende Kommunikationsfunktionen realisieren [4].
Bestrebungen im MSR-Projekt [5], in dem ebenfalls mehrere Hersteller und Zulieferer zusammenarbeiten, sehen die Standardisierung von Funktionsbeschreibung und Werkzeugverwendung an der Schnittstelle zwischen Hersteller und Zulieferer vor.

Im einzelnen unterteilt sich dieser Beitrag in die folgenden Abschnitte: in Kapitel zwei werden allgemeine Anforderungen an ein Kommunikationssystem unter der Berücksichtigung des automotiven Umfeldes aufgestellt. Diese Anforderungen haben insbesondere auch Einfluß auf den Design-und Entwicklungsprozeß eines Netzes. Eine dieser Anforderungen, die Darstellung aller notwendigen Kommunikationsbeziehungen, ist in Kapitel drei genauer beleuchtet. Ausgehend von diesen Anforderungen wird in Kapitel vier die Beschreibungssprache ODL (Open Description Language) vorgestellt. Diese Sprache stellt eine Methode zur standardisierten Beschreibung von Kommunikationsszenarien dar. In Kapitel fünf wird die Entwicklungsumgebung erläutert, in die diese Sprache eingebettet ist und Kapitel sechs enthält einen Ausblick auf diesbezügliche aktuelle Forschungsbereiche an unserem Institut. Im Anhang befindet sich schließlich ein Beispiel für eine ODL-Beschreibung.

2 Allgemeine Anforderungen und Randbedingungen

Bestimmte Funktionen des elektronischen Systems eines Fahrzeugs werden in sogenannten Steuergeräten (ECU, electronic control unit) zusammengefaßt. Diese Funktionen, die als Prozesse in diesen Steuergeräten ablaufen, gehören somit logisch bzw. topologisch zusammen. Als Prozeß bezeichnen wir den dynamischen Ablauf von Befehlen zur Bereitstellung einer bestimmten Funktionalität. Einzelne Steuergeräte werden durch das serielle Bussystem miteinander verbunden. Der typische Aufbau eines Steuergeräts beinhaltet dabei einen Mikrocontroller mit dem Applikationscode, einen Buscontroller für die Abwicklung des Busprotokolls und weitere Hardware zur Sensor- und/oder Aktoransteuerung.
Eine wichtige Randbedingung bei der Entwicklung von elektronischen Komponenten in automotiven Systemen ist der Herstellungspreis dieser Komponenten. Deshalb soll z.B. für die Applikation/Kommunikation sowenig wie möglich RAM gebraucht werden (in der Größen-

ordnung von wenigen hundert Byte!), da RAM einen hohen Anteil der Bauteilkosten eines Steuergeräts bzw. des Mikrocontrollers ausmacht. Eine Möglichkeit dieses Problem zu lösen ist, soviel wie möglich „dynamische" Datenstrukturen zum Applikationscode hinzu in das billigere ROM eines Steuergeräts zu legen. Werden entsprechende Datenstrukturen, die die unveränderliche Architektur des Systems beschreiben, verwendet, kann dem Compiler z.B. durch die Angabe eines Schlüsselworts die Verlagerung in das ROM abverlangt werden. Diese Verlagerung kann außerdem automatisch erfolgen.
Diese aufgeführte Bedingung legt eine zweistufige Beschreibung nahe: in einer ersten Stufe werden die statische Architektur und die festen Zusammenhänge zwischen den Kommunikationsteilnehmern festgelegt und in der zweiten Stufe, im entsprechenden Applikationscode, werden dem Programmierer Kommunikationsdienste zur Verfügung gestellt.
In diesem Beitrag soll im wesentlichen auf die erste Stufe, die Architekturbeschreibung, eingegangen werden. Der Nachteil dieser Methode besteht natürlich darin, daß Kommunikationsbeziehungen zur Laufzeit des Systems feststehen und nicht einfach gewechselt werden können, da die beschreibenden Datenstrukturen fest im ROM des Steuergeräts stehen. Ein entsprechendes Netzmanagement muß dieses Problem für Ausnahmesituationen, in denen z.B. eine Station ausfällt, lösen können. Darauf soll hier aber nicht näher eingegangen werden. Allgemein stellt dieser Nachteil aber keine Einschränkung für die Zielsysteme dar, da im Regelbetrieb keine Änderungen der Kommunikationsbeziehungen notwendig sind.

Wenn man nun Kommunikationsszenarien beschreiben will, muß man zunächst die speziellen Anforderungen an diese Beschreibung kennen. Aus der Diskussion mit Fahrzeugherstellern und eigenen Untersuchungen haben wir einige zentrale Punkte abgeleitet, die bei einer Beschreibung berücksichtigt werden müssen. Unter ihnen sind:

- die Notwendigkeit einer standardisierten Beschreibung, die allen Beteiligten einer Entwicklung eine gemeinsame Sprache bietet,
- die Notwendigkeit zur Einhaltung eines festgelegten Entwicklungsablaufes. Hierzu muß die Funktion eines Netzkoordinators geschaffen werden,
- die Notwendigkeit, die für das System benötigte Information so abstrakt wie möglich darstellen zu können,
- die Notwendigkeit, mit der abstrakten Beschreibung alle für ein automotives System notwendigen Kommunikationsbeziehungen darstellen zu können.

Die drei ersten Aspekte sollen im folgenden kurz ausgeführt werden; der letzte Punkt wird im nachfolgenden Kapitel ausführlicher dargestellt.

Die Notwendigkeit einer standardisierten Beschreibung

Sollen verschiedene Hersteller in den Designprozeß einbezogen werden, so muß eine gemeinsame Beschreibungsbasis bereitgestellt werden. Auf dieser Beschreibungsbasis wird der Hersteller seine Spezifikation erstellen und an den jeweiligen Zulieferer weiterleiten [6]. Die Erstellung der Spezifikation erfolgt mit einem Werkzeug, das auch dem Zulieferer zur Verfügung stehen muß und das außer der Funktionsspezifikation auch die Dokumentation für das Projekt festlegt. Auf der Basis der standardisierten Beschreibung kann außerdem eine automatische Architekturüberprüfung stattfinden.

Die Notwendigkeit eines Entwicklungsablaufs

Entwicklungsabläufe charakterisieren jeden Produktionsprozeß. Sollen Kommunikationsnetze für Fahrzeuge erstellt werden, so sind daran in der Regel mehrere Personengruppen von verschiedenen Firmen beteiligt. Der Hersteller des Fahrzeugs legt im Normalfall die Funktionalität des Gesamtsystems fest und unterteilt es dann in Einzelkomponenten (Steuergeräte), die dann von verschiedenen Zulieferern hergestellt werden. Diese Zulieferer müssen nicht die vollständige Information über die Netzarchitektur erhalten. Ihnen genügt es zu wissen, welche Information sie wann über das Netz erhalten und welche Information sie wann bereitstellen müssen.
Derartige Anforderungen lassen sich mit einer hierarchischen Beschreibung lösen. Ein Netzkoordinator beim Hersteller spezifiziert das Gesamtsystem. Dann erhält ein Zulieferer genau den Teil dieser Gesamtbeschreibung, der den Kommunikationsanteil seines Systems festlegt.
Der Applikationsprogrammierer beim Zulieferer erhält diesen Teil in Form einer „include"-Datei und fügt sie seiner eigenen Beschreibung des Steuergeräts hinzu. Damit kann er sein Gerät erstellen, ohne daß andere über dessen internen Aufbau Bescheid wissen. Die Schnittstellen werden über die Beschreibung festgelegt.

Die Notwendigkeit zur Abstraktion

Elektronische Systeme und insbesondere die darauf realisierte Software werden immer komplexer. Wenn deshalb für die Erstellung der Software geeignete Programmierschnittstellen angeboten werden (API, Application Programming Interfaces), dann muß sich der Applikationsprogrammierer nicht um das darunterliegende System kümmern. Er kann vielmehr auf einen wohldefinierten Satz von Funktionen zugreifen, die ihm die Kommunikationsfähigkeit bereitstellen. Werden zusätzlich objektorientierte Methoden verwendet, stehen dem Programmierer immer dieselben Funktionen zur Verfügung, egal wie die darunterliegende Konfiguration aufgebaut ist. Es wird dann zur Laufzeit automatisch die jeweilige Implementierung aufgerufen.
Diese Abstraktion bezieht sich auf die Erstellung der Applikation. Eine ähnliche Abstraktion muß bei der Beschreibung des Kommunikationsszenarios erreicht werden, d.h. auch hier soll die zugrundeliegende Hardware soweit wie möglich verborgen bleiben. Allerdings muß es auch Möglichkeiten geben, dem System zusätzliche Hardware mit Hilfe der Beschreibung bekannt zu machen. Dies ist insbesondere wichtig, da der Markt an entsprechenden Bausteinen sehr dynamisch ist.

3 Kommunikationstypen in automotiven Systemen

Für die Beschreibung der Kommunikationsszenarien ist es wichtig zu klären, welche Kommunikationsformen unterstützt werden müssen. Das folgende Beispiel verdeutlicht diese Anforderungen (Abb. 1).
Das dargestellte Steuergerät wird mit einem Echtzeit-multiprocessing-Betriebssystem betrieben und erlaubt, in diesem Fall, das „gleichzeitige" Bearbeiten von fünf verschiedenen Prozessen.

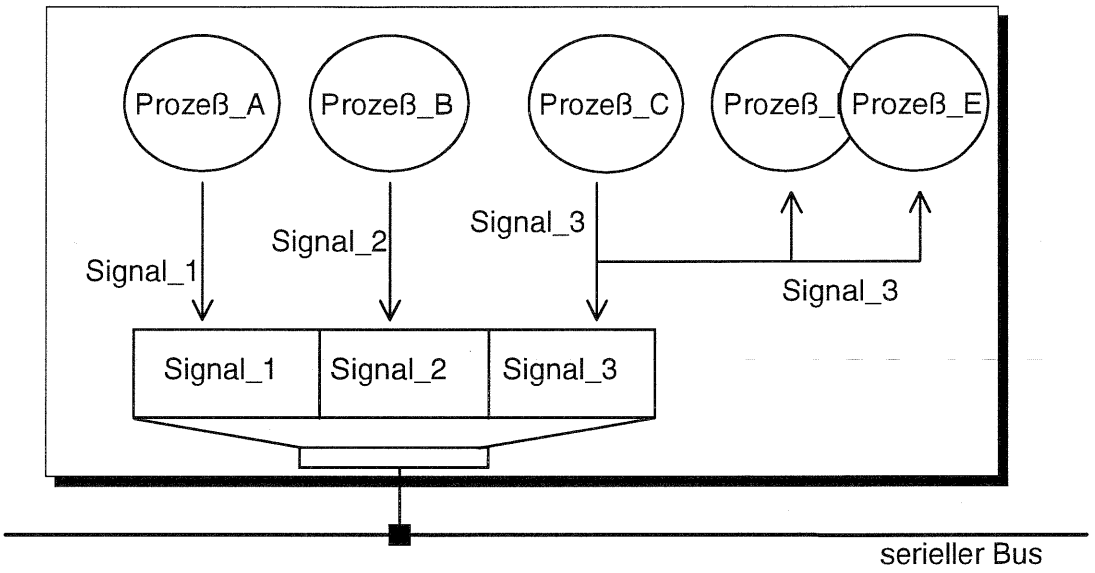

Abb. 1: Paketierung und 1:n-Kommunikation in einer ECU

Diese Prozesse kommunizieren miteinander, indem sie Signale austauschen. Prozeß C schickt Signal 3 (z.B. Information über die Kühlmitteltemperatur) an die Prozesse D und E. Außerdem soll dieses Signal auch über den seriellen Bus zu einem anderen Steuergerät übertragen werden. In diesem Szenario zeigt sich der typische Fall einer 1:n Kommunikation mit einem Sender und mehreren Empfängern.
Bedingt durch die verwendete Botschaftsadressierung des darunterliegenden Bussystems (CAN [7], VAN [8] oder J1850/SCP [9] sind möglich) muß der Sender nicht wissen, welche Empfänger in anderen Steuergeräten sich für seine Information interessieren. Damit erfüllt man, bei entsprechender Ausnutzung, bereits prinzipbedingt die Anforderung nach dem standardisierten Entwicklungsprozeß: Der Programmierer des Steuergeräts muß keine Empfangsadressen für seine bereitgestellte Information kennen und kann sein Gerät unabhängig vom restlichen Netz entwerfen.
Die zweite Kommunikationsart, die in diesem Beispiel dargestellt ist, ist die n:m-Kommunikation. Es wird dabei die Eigenschaft der Bussysteme ausgenutzt in ihrer Nutzlast mehrere Informationsteile transportieren zu können. N Sendeprozesse (A,B, und C) übertragen ihre Signale mittels eines Buspakets an m Empfänger, die wiederum auf anderen Steuergeräten im Netz lokalisiert sind. Diese müssen natürlich wissen, an welcher Stelle eines ankommenden Buspakets die für sie relevante Information steht. Dies läßt sich mit der Information erzielen, die der Netzkoordinator für die einzelnen Systeme zusammenstellt.

Abgeleitet aus dem Gesagten und zur Gewährleistung des Informationsaustauschs innerhalb und zwischen ECUs unterscheiden wir drei abstrakte Kommunikationsformen

- Intra-Modul Interprozeß-Kommunikation, die den Datenaustausch zwischen Prozessen ermöglicht, die sich innerhalb eines Steuergeräts befinden. Notwendige Synchronisationsmechanismen bleiben dem Benutzer dabei verborgen.
- Inter-Modul Interprozeß-Kommunikation zwischen zusammengehörenden Prozessen, die sich auf verschiedenen Steuergeräten befinden. Sie benutzen zur Kommunikation ein für sie transparentes Bussystem.

- Der dritte Typ, die I/O-Kommunikation, beschreibt Kommunikation zwischen einem Prozeß und einer Eingabe- oder Ausgabeeinheit. Dahinter können sich z.B. Sensoren und Aktoren oder z.B. auch der Treiber für eine serielle Schnittstelle verbergen.

Diese Form der Beschreibung ist ausreichend abstrakt. Für den Programmierer einer Steuergeräteapplikation stellt sich dabei die Netzwerkkommunikation ebenfalls wie eine Kommunikation mit einem IO-Device dar: seine Prozesse kommunizieren strenggenommen mit den Bustreibern, d.h. seine Sicht endet an der Steuergerätegrenze. Die globale Sicht hat in diesem System nur der Netzkoordinator, der vorab bestimmen muß, welche Signale an welcher Stelle innerhalb eines Buspakets transportiert werden. Wie bereits oben beschrieben, erhält der Programmierer dann nur Information über die Buspakete, deren Inhalt er tatsächlich kennen muß, bzw deren Inhalt er selber liefern soll. Sie bilden seine Schnittstelle zum Gesamtnetz.

4 ODL: Eine Sprache zur Beschreibung von Kommunikationsszenarien in Steuergerätenetzwerken

Die Beschreibungssprache ODL (Open Description Language) verwendet eine begrenzte Anzahl von Beschreibungselementen, um ein Kommunikationsszenario in abstrakter Form zu spezifizieren. Es werden dazu die Sender und die Empfänger von Informationseinheiten festgelegt, unabhängig von der zugrundeliegenden Hard- und Software.
Einige Elemente der Sprache sind:

- Signale — Logische Informationseinheiten, die bei allen Kommunikationstypen verwendet werden.
- Prozesse — Im Sinn der Kommunikation die Erzeuger und die Rezipienten von Signalen.
- Container — Elemente, die die Signale zwischen den Prozessen transportieren. Sie repräsentieren jegliche Form an verwendeter Hardware.
- Connection — Elemente, die die Zuordnung zwischen Prozeß und Container realisieren. Hier können dynamische Reaktionen auf das Versenden von Signalen angegeben werden.
- Treiber — Softwarebeschreibung, welche die benutzerdefinierte Ergänzung von unterschiedlichster Hardware in das System ermöglichen.

Die Grundidee der Sprache ist, daß Signale als Informationsträger zwischen Prozessen ausgetauscht werden können. Dazu werden sie von einem Prozeß mittels einer Connection an einen Container gegeben. Handelt es sich um eine IO-Kommunikation, ist der Signalweg dabei zu Ende. Soll ein anderer Prozeß dieses Signal empfangen, dann muß von diesem Container, wieder über eine Connection, das Signal zu diesem Prozeß übertragen werden. (Abbildung 2 veranschaulicht diesen Ablauf). Die entstehenden Verbindungen sind gerichtet. Sollen bidirektionale Beziehungen angelegt werden, dann müssen diese durch zwei Verbindungen dargestellt werden.

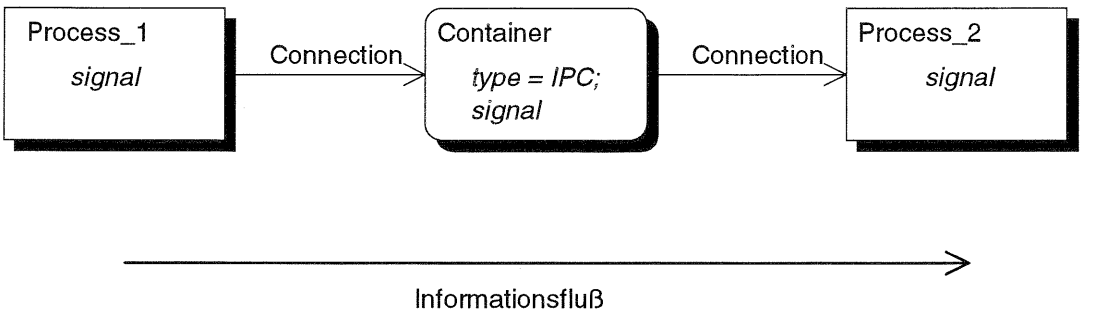

Abb. 2: Gerichtete Kommunikationsverbindung in ODL

Jedes der ODL-Elemente kann weitere Attribute erhalten, die die spezielle Ausprägung dieser Instanz beschreiben. Zum Beispiel wird der gewählte Kommunikationstyp (Intra- oder Inter-Modul IPC, IOC) im jeweiligen Container einer Verbindung mit dem Schlüsselwort „type" festgelegt. Über die Angabe von Completion-Routinen in einer Connection kann der Sender eines Signals über den Abschluß der Sendeaktion unterrichtet werden.
Zusätzlich existieren in ODL noch Objekte, mit denen Timing- und Exception Handling-Funktionen realisiert werden können. Außerdem wurde die Sprache für die Konfigurierung der Prozesse innerhalb des Betriebssystemes erweitert.

5 Codegenerierung für unterschiedliche Zielsysteme

Die Hardware eines automotiven Systems kann, bedingt durch viele Zulieferer, sehr heterogen sein. Verschiedene Prozessoren und unterschiedliche Buscontroller werden in den Steuergeräten eingesetzt. Soll die Methode, mit einer einheitlichen Spezifikationssprache Szenarien zu beschreiben, Akzeptanz finden, dann müssen auch entsprechende Werkzeuge diese Sprache unterstützen und sie auf die verschiedenen Zielsysteme („Targets") abbilden.
Ein Multitarget-Compiler ermöglicht deshalb die Umsetzung der Beschreibungssprache auf die Zielsysteme. Er wandelt dazu die ODL-Szenarienbeschreibung in C-Strukturen um, die dann eine automatische Initialisierung des Kommunikationssystems zur Compilezeit erlauben. Außerdem werden Dateien generiert, die die Initialisierung und Anmeldung der Applikationsprozesse beim Betriebssystem vornehmen. Dies soll hier allerdings nicht weiter ausgeführt werden.
Aufgrund der modularen, objekt-orientierten Architektur des implementierten Compilers können leicht weitere Codegeneratoren hinzugefügt werden. Verfügbar ist die Codegenerierung im Moment für CAN-Controller von Intel, Motorola, Philips und Siemens. Eine Portierung auf andere Bussysteme ist dabei kein Problem, da aufgrund der geschichteten Software im wesentlichen nur die entsprechenden Hardwaretreiber geändert werden müssen.
Das verwendete Kommunikationssubsystem, das skalierbar für die verschiedene Prozessortypen realisiert wurde, basiert auf einem prioritätsgesteuerten Multitasking-Echtzeitbetriebsystem.
Dieses Kommunikationssubsystem und das zugrundeliegende Echtzeitbetriebssystem sind Ergebnisse unserer vorausgegangenen Arbeit.
Der ODL-Compiler ist ein Teil einer Entwicklungsumgebung zur Erstellung von Steuergerätesoftware. Neben der Spezifikation der Kommunikationsbeziehungen können hier

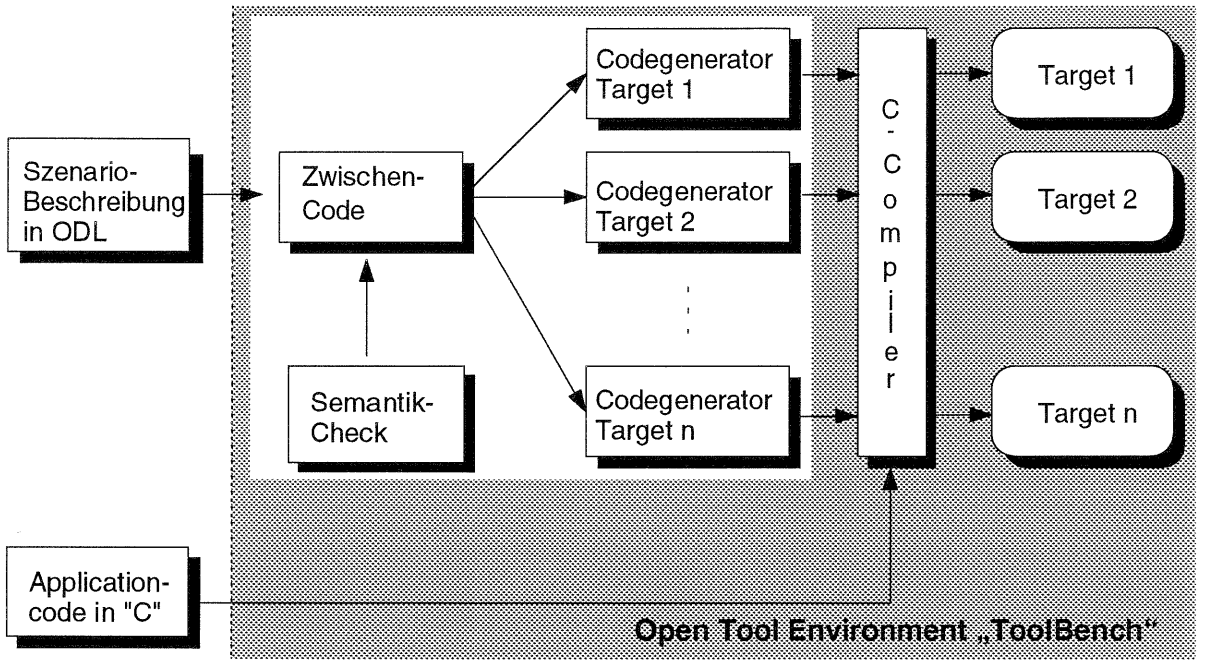

Abb. 3 ODL-Entwicklungssystem. Hier werden nochmals die zwei Beschreibungsstufen, Architektur in ODL und Applikation in C deutlich. Hell hinterlegt ist der Bereich, in dem momentan die ODL-Verarbeitung stattfindet.

auch entsprechende Tests dieser Spezifikation durchgeführt werden. Dazu kann z.B. mit Hilfe eines Semantik-Prüfers überprüft werden, inwieweit die angegebenen Beziehungen der ODL-Beschreibung Sinn machen.
Der Rahmen für diese Entwicklungsumgebung wird durch das an unserem Institut entwickelte Open Tool Environment „ToolBench" für PC-Plattformen gebildet. Damit ist es möglich, beliebige Werkzeuge, die sowohl unter DOS als auch unter Windows erstellt sein können, in einer gemeinsamen Oberfläche zu integrieren und automatische Abläufe dieser Werkzeuge zu veranlassen. Ist z.B. das Editieren der Szenarienbeschreibung beendet, dann werden automatisch die weiteren Stufen Zwischencodegenerierung, Semantiküberprüfung und Compilierung durchlaufen. Die ODL-Werkzeuge stehen außerdem auch in einer UNIX-Version zur Verfügung.

6 Ausblick

Die Beschreibungssprache ODL bildet den ersten Schritt hin zu einem standardisierten Entwicklungsprozeß für automotive Netze und wird bereits an entsprechenden Stellen eingesetzt. Durch die allgemeine Wahl der Beschreibungselemente können auch Kommunikationskonzepte wie Client-Server unterstützt werden.
Momentan wird das Entwicklungssystem um eine Simulationskomponente erweitert. Es ist dadurch möglich, aus der ODL-Beschreibung (und der darin inhärent vorhandenen Kommunikationsmatrix) die Parameter für eine ereignisgesteuerte Simulation zu gewinnen. Es werden dazu sowohl das Bussystem als auch die darüberliegende Betriebssoftware die in geeigneten Modellen vorliegen, in den Simulationsprozeß einbezogen. Mit den

Simulationsergebnissen können dann die zeitlichen Anforderungen (Echtzeitanforderungen an bestimmte sicherheitskritische Signale) an das Netz überprüft und gegebenenfalls ein rechnerunterstützter Optimierungsprozeß eingeleitet werden.
Die Implementierung dieser Simulation erfolgt auf der Basis einer objektorientierten Simulationsbibliothek, was eine einfache Erweiterung für neue Hardwarekomponenten möglich macht.

Literatur

[1] Etschberger, K,; Suters, T.: *„Offene Kommunikation auf CAN-Netzwerken“*, Elektronik 7, Franzis-Verlag, 1993

[2] Färber, G.: *„Feldbus-Technik heute und morgen“*, atp 36, OldenbourgVerlag, 1994

[3] Schwarz, S. *„Es ginge auch anwenderfreundlich - Einsatz von Hochsprachen bei Feldbussen“*, Elektronik 23, Franzis-Verlag, 1994

[4] Kiencke, U. et.al.:*„OSEK - Ein Gemeinschaftsprojekt in der deutschen Automobilindustrie“*, VDI-Berichte 1152, 1994

[5] Leohold, J. *„Das MSR-Projekt: Werkzeugunterstützung für neue Wege der Zusammenarbeit zwischen Automobilhersteller und Zulieferer“* VDI-Berichte 1009, 1992

[6] Stümpfle, M. *„Konzeption und Erstellung von Steuergerätesoftware“*, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart 1993

[7] Controller Area Network, Draft International Standards ISO/DIS 11519-1, ISO/DIS 11898, 1991

[8] Vehicle Area Network, Draft International Standards ISO/DIS 11519-2, 1991

[9] Standard Corporate Protocol - J1850, Draft International Standards ISO/DIS 11519-3, 1991

[10] Stümpfle, M.; Eberspächer, M.: *„Open Description Language“*, Institut für Nachrichtenvermittlung und Datenverarbeitung, Universität Stuttgart, 1993

Anhang

Das nachfolgende Beispiel in ODL realisiert die Beschreibung des in Kapitel drei vorgestellten Szenarios. Es wird dazu zunächst in der Datei `netspec.odl` die Information durch den Netzkoordinator festgelegt. Er benennt die drei Signale, die über das Netz ausgetauscht werden sowie auch den Transportcontainer für diese Signale. Mit der Reihenfolge der Signale im Container legt er implizit die Anordnung der Information im Buspaket fest.
Diese Datei wird dann in die Steuergerätebeschreibungsdatei durch eine Präprozessordirektive eingebunden. In dieser Datei wird zuerst das Steuergerät „example“ (`ecu`, electronic control unit) mit seinen fünf Prozessen A-E beschrieben. Danach folgen die Prozesse, die die Signale produzieren und konsumieren. Die internen Verbindungen werden nun durch die Connections und Container vervollständigt.
Es ist zu bemerken, daß in diesem Beispiel nur die statischen Elemente der Kommunikationsbeschreibung aufgeführt sind. Treiberbeschreibungen und Reaktionen (sog.

Completion-Routinen) und Betriebssystembeeinflussungen werden hier nicht berücksichtigt. Für eine ausführliche Beschreibung siehe [10].

```
/***************************/
/* Network description file*/
/* file  : netspec.odl     */
/* author: M. Stuempfle    */
/* (c)'94 IND-Uni Stuttgart*/
/***************************/
signal signal_1;
signal signal_2;
signal signal_3;

container NetWork {
     signal = signal_1;
     signal = signal_2;
     signal = signal_3;
}
/* EOF */


/***************************/
/* odl description example */
/* file  : example.odl     */
/* author: M. Stuempfle    */
/* (c)'94 IND-Uni Stuttgart*/
/***************************/

#include "netspec.odl"

ecu example {
     process = process_A;
     process = process_B;
     process = process_C;
     process = process_D;
     process = process_E;
     driver  = CAN;
     ...
}

/* processes in ecu: */

process process_A {
     signal = signal_1;
}

process process_B {
     signal = signal_2;
}

process process_C {
     signal = signal_3;
}

process process_D {
     signal = signal_3;
}

process process_E {
     signal = signal_3;
}

/* nwc connections          */
/* container in netspec.odl*/

connection con1 {
     signal = signal_1;
     direction = send;
     task = process_1;
     container = NetWork;
}
connection con2 {
     signal = signal_1;
     direction = send;
     task = process_1;
     container = NetWork;
}
connection con3 {
     signal = signal_1;
     direction = send;
     task = process_1;
     container = NetWork;
}

/* ipc: proc C -> D */
container InternalCD {
     signal = signal_3;
```

```
}

connection con4 {
     signal = signal_1;
     direction = receive;
     task = process_1;
     container = InternalCD;
}

/* ipc: proc C -> E */

container InternalCE {
     signal = signal_3;
}
connection con5 {
     signal = signal_1;
     direction = receive;
     task = process_1;
     container = InternalCE;
}

/* EOF */
```