

Ein flexibles Testfeld für Experimente im Bereich der Gebäudeautomation und -simulation

Andreas Metzger

04/2001

Sonderforschungsbereich 501

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-67653 Kaiserslautern

Ein flexibles Testfeld für Experimente im Bereich der Gebäudeautomation und -simulation

Andreas Metzger

SFB 501 Bericht 4/01

Abstract

Dieser Bericht beschreibt ein im Teilprojekt D1 des Sonderforschungsbereichs 501 entstandenes Testfeld, in welchem Experimente im Bereich der Gebäudeautomation und -simulation durchgeführt werden können. Eine typischer Einsatz für das Testfeld ist z.B. die prototypische Validierung von Gebäudeautomationssystemen.

Neben den Sensoren und Aktuatoren werden in diesem Bericht die verfügbaren Schnittstellen zu dem Testfeld vorgestellt. Diese Schnittstellen stehen, je nach Anwendung, auf unterschiedlichen Abstraktionsebenen zur Verfügung.

Schließlich wird, nach der Darstellung der hardware- und softwaretechnischen Realisierung, eine kurze Beschreibung zweier Fallstudien und deren Ergebnisse gegeben.

Inhaltsverzeichnis

	Einleitung	7
Abschnitt 1	Sensoren und Aktuatoren	9
1.1	Testräume	9
1.2	Flursegment	13
1.3	Sonstige Sensoren	15
1.3.1	Außenhelligkeitssensoren	15
1.3.2	Wetterstation	15
Abschnitt 2	Schnittstellen	17
2.1	Komponenten des Testfelds	17
2.2	Schnittstelle auf Zell- und Managementebene	18
2.2.1	Protokoll	19
2.2.1.1	Kommandos für Sensoren	20
2.2.1.2	Kommandos für Aktuatoren	20
2.2.1.3	Werte der Meldungen	21
2.3	Schnittstelle auf Feldebene	23
2.3.1	Protokoll	23
Abschnitt 3	Realisierung	27
3.1	Schnittstelle auf Zell- und Managementebene	27
3.1.1	Konfiguration	28
3.1.2	Hardware	32
3.2	Schnittstelle auf Feldebene	33
3.2.1	Konfiguration	34
3.2.2	Hardware	36
3.3	Schnittstelle auf Ebene der Messdatenverbindung	39
Abschnitt 4	Experimente und Fallstudien	43
4.1	Fallstudie „Anforderungsanalyse“	43
4.2	Fallstudie „Verteiltes Prototyping“	44

Anhang

51

A Bibliotheksfunktionen für Ican 51

Literaturverzeichnis

55

Einleitung



Im Rahmen des Sonderforschungsbereichs 501: „Entwicklung großer Systeme mit generischen Methoden“ entstehen unter Anderem Gebäudeautomationssysteme und Gebäudesimulatoren. Um realistische Testszenarien für diese reaktiven Systeme durchspielen zu können entsteht in Teilprojekt D1: „Anwendungssystem Gebäude“ ein physikalisches Testfeld. Dieses besteht aus mit Sensoren, Aktuatoren, Rechnern und Kommunikationstechnik ausgestatteten Räumen und Flursegmenten im vierten Stock des Gebäudes 32 der Universität Kaiserslautern.

Der Zugriff auf die Sensoren (ca. 115 Stück) und Aktuatoren (ca. 55 Stück) ist flexibel gestaltet, um vielfältige Experimente mit vertretbarem Aufwand durchführen zu können. Von einfachen Mikrocontrollern bis hin zu leistungsfähigen PCs stehen Rechner zur Verfügung, um Sensordaten aufzubereiten und Aktuatoren zu steuern. Zusätzlich können diese Rechner genutzt werden, um Prototypen oder fertige Produkte ablaufen zu lassen.

Das Testfeld wurde in Experimenten eingesetzt, um Gebäudeautomationssysteme und Simulatoren zu validieren und Konzepte für die Partitionierung von Prototypen zu überprüfen. Weiter leistete D1 Hilfestellung bei der Experimentdurchführung anderer Teilprojekte.

Das aktuelle Testfeld entstand aus einem initialen Testraum (vgl. dazu [MQS96]). Dieser wurde in der zweiten Förderungsperiode des SFB um drei weitere Räume und zwei Flursegmente ergänzt, womit ein größerer Bereich von Szenarien zur Validierung von Gebäudeautomationssystemen und -simulatoren abgedeckt wird. Bei der Installation zusätzlicher Komponenten wurde die für den einzelnen Raum eingesetzte sternförmige Verdrahtung aufgegeben und durch eine hierarchische Kommunikationsstruktur ersetzt.

In den folgenden Kapiteln wird ein Überblick über die eingesetzten Sensoren und Aktuatoren und die Schnittstellen zu diesen Komponenten gegeben. Weiter werden Details der Realisierung der verwendeten Hardware und der Implementierung der benutzten Software

aufgezeigt. Am Ende werden die qualitativen und quantitativen Ergebnisse ausgewählter Experimente und Fallstudien aufgeführt.

Abschnitt 1

Sensoren und Aktuatoren

Das Testfeld befindet sich im vierten Stock des Gebäudes 32 der Universität Kaiserslautern. Abbildung 1 zeigt den aktuellen Ausbaustand. Die schattiert dargestellten Räume und Flursegmente sind vollständig mit Sensorik und Aktuatorik instrumentiert.

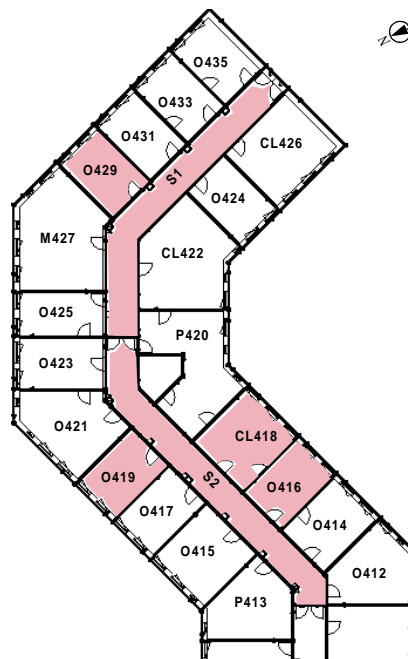


Abbildung 1 Ausbaustand des Testfelds, Gebäude 32, 4. Stock

Für Tests stehen zwei Computerlabors (CL418 und O416), zwei Mitarbeiterbüros (O419 und O429) und zwei Flursegmente (S1 und S2) zur Verfügung.

1.1 Testräume

Alle Testräume verfügen weitgehend über die gleiche Ausstattung. Abbildung 2 zeigt, beispielhaft für alle Testräume, die Topologie des Raums CL418. Dieser Raum ist ein Arbeitsraum für Studenten und wird im normalen Arbeitsbetrieb genutzt.

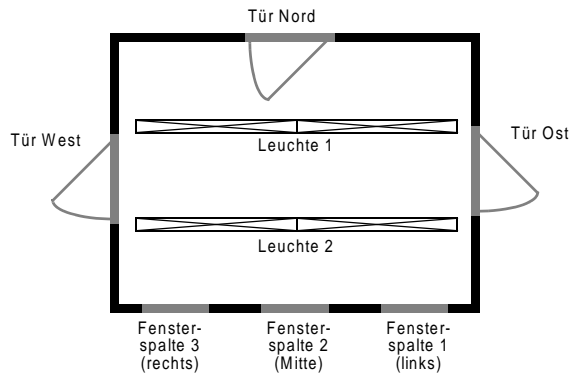


Abbildung 2 Topologie des Testraums CL418

Bei der Instrumentierung wurde er in verschiedene Teilbereiche (Fenster-spalten, Türen, usw.) aufgeteilt, denen jeweils die gleichen Typen von Sensoren und Aktuatoren zugeordnet wurden. Hier sollen zunächst die räumlichen Gegebenheiten betrachtet werden.

- Eine **Fenster-spalte** besteht aus einem manuell bedienbaren Fensterflügel, einem automatisch¹ bedienbaren Oberlicht, einem Radiator (Heizkörper), einer automatisch² bedienbaren Außenjalousie und zwei automatisch bedienbaren Innenjalousien.
- **Türen** existieren zu den benachbarten Räumen und zum Flur.
- Es sind zwei **Leuchtstofflampen**(-gruppen) mit einer Leistung von je 220W vorhanden. Die Gruppen lassen sich getrennt schalten und die Leuchten pro Gruppe jeweils separat dimmen. Es sind zwei³ Leuchten pro Gruppe vorhanden.

In Abbildung 3 ist die Zuordnung der Sensoren und Aktuatoren zu den räumlichen Entitäten an Hand eines an UML angelehnten Klassendiagramms mit Hilfe der Aggregationsrelation gezeigt. Zur besseren Lesbarkeit wurden dabei einige Klassen (z.B. Fensterkontakt) doppelt eingezeichnet.

1 nur für CL418
 2 dito
 3 dito

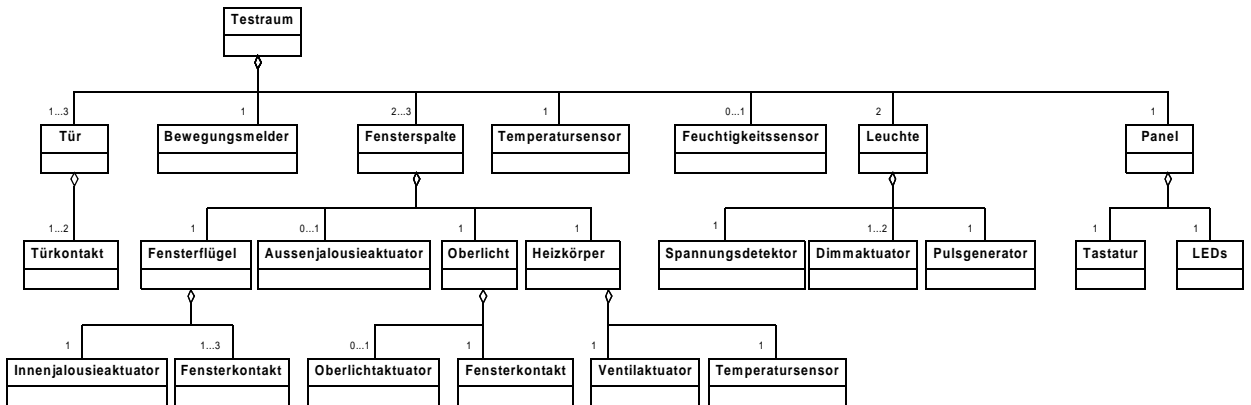


Abbildung 3 Klassendiagramm eines Testraums

Jede Tür verfügt über Türkontakte, welche die Stellung der Tür melden. Die Fensterspalten sind mit Fensterkontakten, Jalousieaktuatoren, Heizkörperventilaktuatoren und Heizkörper temperatursensoren ausgestattet. Die Leuchten lassen sich Dimmen und über Pulsgeneratoren ein- und ausschalten, wobei sich der Einschaltzustand über Spannungsdetektoren feststellen lässt. Eine Bewegung im Raum wird über Bewegungsmelder registriert. Des Weiteren ist ein Raumtemperatursensor vorhanden. Über das Panel lassen sich numerische Eingaben tätigen und Systemzustände über drei verschiedenfarbige LEDs anzeigen.

Tabelle 1 stellt eine vollständige Aufzählung der einzelnen Sensoren und Tabelle 2 eine Auflistung der Aktuatoren dar. Zusätzlich zu einer kurzen Beschreibung findet sich hier auch ein abgekürzter Name, welcher später noch Verwendung finden wird (siehe Abschnitt 3.1.1).

Typ	Abkürzung	Raum	Beschreibung
Türkontakt	dcc	O416, CL418, O419, O429	meldet 1 bei offener Tür
	doc	CL418, O416	meldet 1 wenn Tür um mehr als 30 Grad geöffnet ist

Tabelle 1 Sensoren der Testräume

Typ	Abkürzung	Raum	Beschreibung
Fensterkontakt	wcc	O416, CL418, O419, O429	meldet 1 bei offenem Fensterflügel
	woc	O416, CL418	meldet 1 wenn Fensterflügel um mehr als 30 Grad geöffnet ist
	wlc	CL418	meldet 1 bei verriegeltem Fensterflügel
	wuc	O416, CL418, O419, O429	meldet 1 bei offenem Oberlicht
Temperatursensor	wts	O416, CL418, O419, O429	liefert die Einlasstemperatur des Heizkörpers [°C]
	rts	O416, CL418, O419, O429	liefert die Raumtemperatur [°C]
Helligkeitssensor	rls	CL418	liefert die Raumhelligkeit [lx]
Feuchtigkeitssensor	rhs	CL418	liefert die relative Raumfeuchte [%]
Bewegungsmelder	imd	O416, CL418, O419, O429	meldet 1 bei einer Bewegung im Raum
Spannungsdetektor	sll	O416, CL418, O419, O429	meldet 1 wenn Leuchte mit Spannung versorgt wird
Tastatur	kpd	O416, CL418, O419, O429	erlaubt die Eingabe von Ziffern über eine Zehnertastatur

Tabelle 1 Sensoren der Testräume

Typ	Abkürzung	Raum	Beschreibung
Jalousieaktuator	bop	CL418	fährt die Außenjalousie (Dauer für vollständige Fahrt: 1–1,5 min)
	bld	O416, CL418, O419, O429	fährt die Innenjalousie vor dem Fensterflügel
	bld	O416, CL418, O419, O429	fährt die Innenjalousie vor dem Oberlicht

Tabelle 2 Aktuatoren der Testräume

Typ	Abkürzung	Raum	Beschreibung
Ventil- aktuator	rva	O416, CL418, O419, O429	fährt Stellantrieb für das Heizkörperventil (Dauer für vollständigen Hub: 6min)
Pulsgenerator	lpg	O416, CL418, O419, O429	schaltet eine Stromstoßrelais zur Ansteuerung der Leuchten
Dimm- aktuator	dim	O416, CL418, O419, O429	erlaubt das Dimmen der Leuchten im Bereich von 10% bis 100%
Oberlicht- aktuator	wop	CL418	erlaubt das Öffnen und Schließen der Oberlichter (Dauer für vollständige Fahrt: 30s)
LEDs	led	O416, CL418, O419, O429	erlaubt die Anzeige von Systemzuständen über drei LEDs in den Farben Rot, Grün und Gelb

Tabelle 2 Aktuatoren der Testräume

Der Messbereich der Temperatursensoren liegt zwischen 2°C und 150 °C.

Die Abtastrate der Sensoren ist flexibel anpassbar und liegt in der aktuellen Installation bei ca. 500ms für Binärsensoren und ca. 2000ms für Analogsensoren. Diese Werte gelten auch für die Sensorik in den Flursegmenten.

1.2 Flursegment

Die beiden Flursegmente verfügen über eine weitgehend identische Ausstattung mit Sensorik und Aktuatorik.

Ein Flursegment ist zu beiden Seiten durch Durchgangstüren begrenzt wie in Abbildung 1 gezeigt. Die Durchgangstüren besitzen einen Türkontakt und zwei Bewegungsmelder, um Bewegungen auf beiden Seiten der Tür zu registrieren. Zusätzlich ist das gesamte Flursegment durch Bewegungsmelder abgedeckt. In den Räumen lässt sich das Licht über Pulsgeneratoren schalten und der Zustand über Spannungsdetektoren feststellen. Schließlich existiert eine Alarmsirene zur internen Alarmierung.

Abbildung 4 zeigt das Klassendiagramm zu diesem Teil des Testfelds.

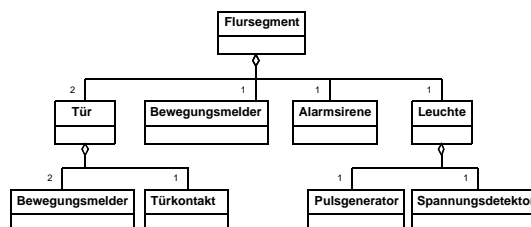


Abbildung 4 Klassendiagramm eines Flursegments

Die folgenden Tabellen geben die aktuell verfügbaren Sensoren und Aktuatoren wieder.

Typ	Abkürzung	Flursegment	Beschreibung
Bewegungsmelder	imd	S1, S2	liefert eine 1 falls eine Bewegung vor einer Zugangstür erkannt wurde
	imd	S1, S2	liefert eine 1 falls eine Bewegung hinter einer Zugangstür erkannt wurde
	imd	S1	liefert eine 1 bei einer Bewegung im Flursegment und zusätzlich einen Vektor, der die auslösenden Bewegungsmelder kennzeichnet
	imd	S2	liefert eine 1 bei einer Bewegung im Flursegment
Spannungsdetektor	sll	S1, S2	liefert eine 1 falls die Flurbeleuchtung mit Spannung versorgt wird

Tabelle 3 Sensoren der Flursegmente

Typ	Abkürzung	Flursegment	Beschreibung
Pulsgenerator	lpg	S1, S2	schaltet ein Stromstoßrelais zur Ansteuerung der Flurbeleuchtung

Tabelle 4 Aktuatoren der Flursegmente

Typ	Abkürzung	Flursegment	Beschreibung
Alarmsirene	fl	S1, S2	schaltet die interne Alarmsirene

Tabelle 4 Aktuatoren der Flursegmente

1.3 Sonstige Sensoren

Neben den Testräumen und den Flursegmenten verfügt das Testfeld über zusätzliche Sensorik. Dazu gehören zum Einen die Außenhelligkeitssensoren, die an der Fassade des Gebäudes montiert sind, und zum Anderen die Sensoren einer Wetterstation.

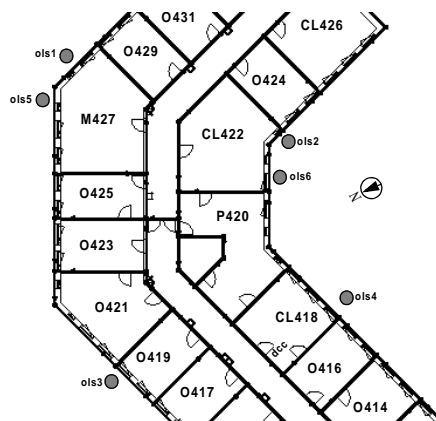


Abbildung 5 Montageorte der Außenhelligkeitssensoren

1.3.1 Außenhelligkeitssensoren

Die Außenhelligkeitssensoren (Abkürzung: ols) sind wie in Abbildung 5 gezeigt an der Fassade des Gebäudes montiert und erlauben den potenziellen Lichteinfall in jeden Raum festzustellen. Die Abtastrate liegt bei ca. 1000ms und der Messbereich zwischen 0Lux und 60000Lux.

1.3.2 Wetterstation

Die Wetterstation stellt zusätzliche Umweltdaten zur Verfügung. Dazu gehören z.B. die Niederschlagsmenge, die Windrichtung- und Geschwindigkeit und die Außenfeuchte.

Tabelle 5 stellt die verfügbaren Daten dar. Diese werden ca. alle 15 Sekunden aktualisiert.

Typ	Abkürzung	Montageort	Beschreibung
Temperatur-sensor	ots	Fassade, Raum CL418	liefert die Außentemperatur [°C]
Feuchtig-keitssensor	ohs	Fassade, Raum CL418	liefert die relative Außen-feuchte [%]
Regensensor	ors	Dach, Gebäude 32	liefert die Niederschlags-menge der letzten 24 Stunden [mm]
Luftdruck-sensor	aps	CL418	liefert den Luftdruck [hPa]
Windsensor	wss	Dach, Gebäude 32	liefert die Windgeschwindig-keit [km/h]
Windrichtung	wds	Dach, Gebäude 32	liefert die Windrichtung [Grad]

Tabelle 5 Sensoren der Wetterstation

Abschnitt 2 Schnittstellen

In diesem Kapitel werden die verschiedenen Schnittstellen zu den Sensoren und Aktuatoren des Testfelds beschrieben. Dazu wird zunächst eine grobe Einteilung des Testfelds in einzelne Teilkomponenten vorgenommen und anschließend die vorhandenen Schnittstellen zu diesen Komponenten dargelegt.

2.1 Komponenten des Testfelds

Das Testfeld besteht aus den in Abbildung 6 abstrakt dargestellten Komponenten. Ein Bereich steht dabei entweder für einen Raum oder ein Flursegment.

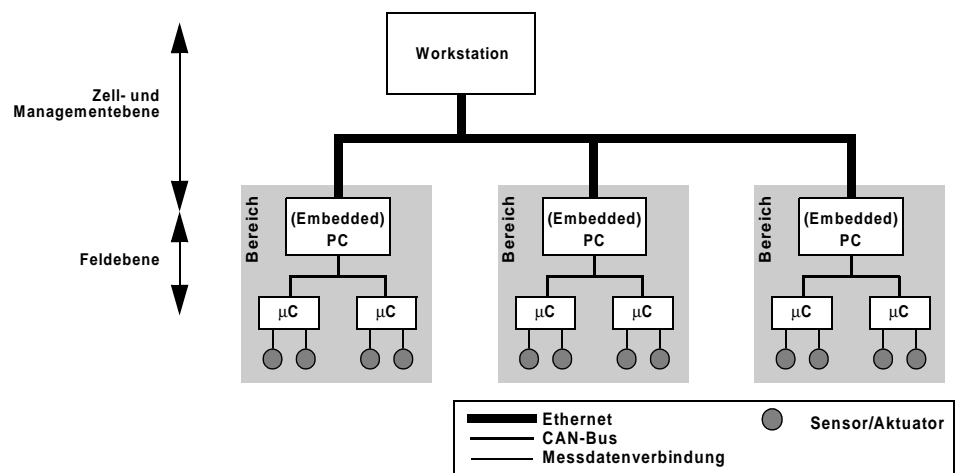


Abbildung 6 Komponenten des Testfelds

Sensoren und **Aktuatoren** bilden die eigentliche Ankopplung an die physikalische Umgebung. Die verfügbaren Sensoren und Aktuatoren wurden bereits in Abschnitt 1 beschrieben.

Unterschiedlich leistungsfähige **Rechner** sammeln Messdaten der verschiedenen physikalischen Sensoren, steuern Aktuatoren und erlauben komplexe Softwareprototypen oder -produkte ablaufen zu lassen. Die im Testfeld eingesetzten Rechner reichen von einfachen

8-Bit Mikrokontrollern (μC) über Embedded PCs zu Standard PCs und Workstations

Ein **hierarchisches Kommunikationsnetzwerk** verbindet die einzelnen Rechner und erlaubt den Datenaustausch zwischen Komponenten. Als lokaler Feldbus kommt CAN (Controller Area Network) zum Einsatz und für die Verbindung der Prozessoren auf Zell- und Managementebene wird Ethernet eingesetzt¹.

Für die unterschiedlichen Ebenen der Kommunikationshierarchie (wie in Abbildung 6 angedeutet) werden passende **Schnittstellen** zu den Sensoren und Aktuatoren realisiert. Auf Feldebene wird ein einfaches CAN-Protokoll verwendet, wohingegen auf Zell- und Managementebene ein Protokoll, das auf TCP/IP-Sockets basiert, eingesetzt wird.

Die Messdatenverbindung zu den Sensoren und Aktuatoren wird erst in Abschnitt 3 beschrieben, da diese Schnittstelle nicht für den Benutzerzugriff konzipiert ist.

2.2 Schnittstelle auf Zell- und Managementebene

Basis für die Schnittstelle zu Sensoren und Aktuatoren auf Zell- und Managementebene bilden TCP/IP-Sockets. Dieser Kommunikationsmechanismus wird von vielen Betriebssystemen (z.B. UNIX, Windows, QNX) unterstützt. Es wird zwischen Client- und Server-Sockets unterschieden. Ein Server-Socket wartet auf einen Verbindungswunsch eines oder mehrerer Clients. Sobald die Verbindung aufgebaut wurde kann die Kommunikation in beide Richtungen erfolgen. Der Server-Socket ist durch die IP-Adresse des Rechners und eine Port-Nummer eindeutig festgelegt. Eine detaillierte Einführung finden man in [Rag93].

Die PCs der einzelnen Bereiche stellen jeweils einen Server-Socket zur Verfügung. Die Port-Nummer ist dabei einheitlich die 32418. Die

¹ Die Bezeichnung der einzelnen Ebenen ist der Begriffswelt des CIM (Computer Integrated Manufacturing) entlehnt [Sch97].

Namen¹ der Rechner und die Zuordnung zu Flursegmenten und Räumen sind Tabelle 6 zu entnehmen.

IP-Adresse	Bereich
hauspc1	S2
hauspc3	S1
bettina	CL418
cordula	O416
daniela	O429
erika	O419

Tabelle 6 Zuordnung von PCs zu Bereichen

Die Verbindung kann von Client-Seite durch Senden von $\wedge B$ (<Ctrl>-) oder durch Schließen des Socket beendet werden.

2.2.1 Protokoll

Das Protokoll für den Austausch von Daten und Befehlen zwischen Client und Server soll in den folgenden Abschnitten dargelegt werden. Prinzipiell erfolgt die Übertragung in einer lesbaren Form – als Strings. Diese Strings sind stets mit $\wedge A$ (<Ctrl>-<A>) terminiert.

Es wird zwischen **Befehlen**, die eine Zustandsänderung im Server bewirken, und **Meldungen**, die eine Reaktion auf Befehle oder asynchrone Ereignisse darstellen, unterschieden.

Die Befehle besitzen das folgende Format:

<identifizier> <kommando> <parameter>

Der Identifizier kennzeichnet eindeutig einen Sensor oder Aktuator. Diese Identifizier können flexibel festgelegt werden (siehe dazu auch Abschnitt 3.1.1). Als Vorschlag dienen die in den Abschnitten 1.1 bis 1.3 verwendeten Abkürzungen. Die möglichen Kommandos und Parameter werden in den folgenden beiden Abschnitten beschrieben.

Für die Meldungen wird das folgende Format verwendet:

¹ Die Domäne ist informatik.uni-kl.de

<identifizier> <wert>

Für den Identifizier gilt das zuvor Gesagte. Die möglichen Ausprägungen der Werte werden in Abschnitt 2.2.1.3 dargestellt.

2.2.1.1 Kommandos für Sensoren

Sowohl für Analogsensoren als auch für Binärsensoren steht der **Request-Mode** zur Verfügung. Damit kann der aktuelle Wert eines Sensors abgefragt werden. Das Kommando lautet `getValue` (die Groß- und Kleinschreibung ist zu beachten!). Um zum Beispiel die Raumtemperatur in CL418 abzufragen würde man den Befehl „`rts1CL418 getValue`“ verwenden.

Zusätzlich zu dem Request-Mode existiert für binäre Sensoren der **Event-Mode**. Er sieht vor, dass der Sensor sich selbsttätig meldet, wenn sich sein Zustand ändert. Es wird dann automatisch eine Meldung an alle¹ Clients verschickt. Das Setzen des Event-Mode geschieht durch das Kommando `setEventMode`, das Rücksetzen durch `resetEventMode`. Möchte man beispielsweise den Türkontakt in O416 in den Event-Mode versetzen würde man den Befehl „`dcc1O416 setEventMode`“ senden.

2.2.1.2 Kommandos für Aktuatoren

Für Aktuatoren ist nur das Kommando `setValue` möglich. Man unterscheidet allerdings die Bedeutung der Parameter je nach Aktuatortyp:

- Ein Aktuator mit drei Bewegungsrichtungen lässt sich über `-1`, bzw. `1` in jeweils eine Richtung fahren und mit `0` stoppen. Beispiel: Zum Öffnen eines Heizkörperventils würde man „`rvalO429 setValue 1`“ benutzen.
- Ein binärer Aktuator lässt sich mit `1` ein- und mit `0` ausschalten.
- Analogaktuatoren werden über einen reellen Wert angesteuert. So erfolgt die Ansteuerung der dimmbaren Leuchten im Bereich von 10% bis 100% durch eine Parameterbelegung zwischen `10.0` und `100.0`.

¹ Einen Mechanismus, der den Event-Mode an einen speziellen Client bindet, existiert nicht. Eine Filterung muss daher bei Bedarf auf der Seite des Clients erfolgen.

- Ein Pulsaktor erzeugt stets einen ca. 250ms langen 0–1–0 Impuls. Um das Befehlsformat einheitlich zu halten muss hier dennoch ein Parameter (mit beliebigem Wert, z.B. 0) angegeben werden.

In Tabelle 7 sind nochmals alle verfügbaren Kommandos zusammengefasst.

Kommando	Beschreibung	zulässiger Sensor-/Aktuortyp
getValue	Abfragen eines Sensorwerts	Binär- und Analogsensor
setEventMode	Aktivieren des Event-Mode	Binärsensor
resetEventMode	Deaktivieren des Event-Mode	Binärsensor
setValue	Ansteuerung eines Aktuators	Aktuator

Tabelle 7 Kommandos der Schnittstelle

2.2.1.3 Werte der Meldungen

Jeder Befehl wird quittiert, d.h. hat eine Meldung zur Folge. Für die Kommandos `setEventMode`, `resetEventMode` und `setValue` ist das der Wert „ACK“ im fehlerfreien Fall¹. Ansonsten wird ein Fehler durch den Wert „ERROR“ gefolgt von einer Fehlerbeschreibung gemeldet. Mögliche Fehler sind in Tabelle 8 aufgelistet.

¹ Diese Bestätigung bedeutet für Aktuatoren lediglich, dass der Befehl empfangen und an die Hardware weitergereicht wurde. Ob der Aktuator wirklich die gewünschte Aktion ausführt muss zusätzlich durch das Abfragen von Sensorinformation überprüft werden.

Auf das Kommando `getValue` erfolgt im fehlerfreien Fall eine Meldung mit dem Wert des abgefragten Sensors. Diese Meldung wird auch im Event-Mode bei einer Wertänderung übertragen.

Fehlermeldung	Beschreibung	Verursacher
component not found	Der angegebene Identifier ist dem System unbekannt; entweder ist der angegebene Identifier falsch oder die Identifier des Systems wurden nicht korrekt konfiguriert	Benutzer/ Sysadmin
syntax error	Der Befehl hatte eine ungültige Syntax	Benutzer
component is of unknown type	Der Sensor/Aktuator wurde als unbekannter Typ konfiguriert	Sysadmin
cannot change event mode for specified sensor	Der Sensor erlaubt keinen Event-Mode	Benutzer
unknown command for specified sensor	Der Sensor existiert, es wurde aber ein unbekanntes Kommando verwendet	Benutzer
no value supplied	Bei <code>setValue</code> wurde kein Wert angegeben	Benutzer
wrong value for specified actuator	Bei <code>setValue</code> wurde ein falscher Wert angegeben	Benutzer
unknown command for specified actuator	Der Aktuator existiert, es wurde aber ein falscher Befehl verwendet	Benutzer
could not change value for specified actuator	Der Befehl wurde richtig verarbeitet, allerdings konnte der Aktuator nicht angesprochen werden (Kommunikation ist fehlgeschlagen, oder Hardwaretreiber nicht aktiv)	System
data for specified sensor is not available	Es liegen im Moment keine Sensordaten vor (Sensor defekt oder noch nicht physikalisch gepollt)	System

Tabelle 8 Fehlermeldungen

Fehlermeldung	Beschreibung	Verursacher
not enough resources available to server	Dem Server steht nicht genügend Arbeitsspeicher oder Plattenkapazität zur Verfügung	System

Tabelle 8 Fehlermeldungen

Bei einigen Fehlermeldungen wird `sys1` als Identifier zurückgeliefert. Dies ist immer dann der Fall, wenn sich kein Identifier im auslösenden Befehl identifizieren ließ.

2.3 Schnittstelle auf Feldebene

Die Schnittstelle auf Feldebene nutzt die Möglichkeiten, die das CAN-Protokoll auf OSI-Schicht 2 anbietet (Object und Transfer Layer [Bos91a]). Das ist zum Einen das Versenden von Datenrahmen (Data-Frames) mit bis zu 8 Bytes Nutzdaten und zum Anderen die Anforderung solcher Datenrahmen mit sogenannten Remote-Frames. Bei CAN handelt es sich um ein „objektorientiertes“ Protokoll, d.h. die Data- und Remote-Frames besitzen keine Zieladresse sondern eine eindeutige Objektidentifikation, den CAN-Identifier.

Die Mikrokontroller reagieren abhängig von der Objektidentifikation auf den Empfang der unterschiedlichen Rahmen mit der Rückgabe von Sensorwerten oder der Ansteuerung von Aktuatoren.

2.3.1 Protokoll

Zur Ansteuerung von Aktuatoren werden Data-Frames eingesetzt. Das verwendete Format ist wie folgt:

```
<CAN-Identifier> <RTR> <LEN> <DATA0> [<DATA1>]
```

Der CAN-Identifier stellt eine eindeutige Identifikation des Aktuators dar. Mit der momentan verwendeten Hardware sind 11-Bit Identifier erlaubt, d.h. zulässige Werte für den CAN-Identifier liegen zwischen 0 (höchste Priorität) und 2048 (niedrigste Priorität).

Das `RTR`-Bit (Remote Transmission Request) ist auf 0 festgelegt und kennzeichnet damit den Rahmen als Data-Frame. Die Länge (`LEN`) der Nutzdaten (`DATAn`) liegt zwischen 1 und 2.

Die Nutzdaten besitzen, abhängig vom Typ des Aktuators, das in Tabelle 9 angegebene Format. Die Dauer legt fest, wie lange der Aktuator angesteuert wird, bis er gestoppt oder ausgeschaltet wird. Eine Dauer von 0 bedeutet die Erzeugung eines kurzen Impulses, eine Dauer von 255 bedeutet ein unendlich langes Ansteuern.

Typ	LEN	DATA0	DATA1
Binäraktuator	2	1 = einschalten 3 = ausschalten	Dauer
Aktuator mit drei Bewegungsrichtungen	2	1 = Richtung 1 2 = Richtung -1 3 = Stopp	Dauer
Pulsgenerator (realisiert in Hardware)	2	1	0
Pulsgenerator (realisiert in Software)	2	4	0
Analogaktuator	1	0% = 0 100% = 255	-
Dezimalaktuator	1	0, ..., 255	-

Tabelle 9 Datenformate für Aktuatoren

Die Unterscheidung zwischen zwei Typen von Pulsgeneratoren ist notwendig, da unterschiedliche Realisierungen für diese Aktuatoren existieren.

Um Sensordaten abzufragen werden Remote-Frames mit folgendem Format verwendet:

<CAN-Identifizier> <RTR> <LEN>

Der CAN-Identifizier identifiziert den gewünschten Sensor. Das RTR-Bit ist auf 1 festgelegt und kennzeichnet damit den Rahmen als Remote-Frame. Die Längenkennung kann beliebig gewählt werden, da sie nicht ausgewertet wird. Um mit zukünftigen Erweiterungen kompatibel zu bleiben, sollte hier allerdings je nach Sensortyp die Länge des als Antwort gesendeten Datenrahmens (siehe nächster Abschnitt) angegeben werden.

Als Antwort auf ein Remote-Frame oder auf Grund einer Änderung eines Sensorwertes senden die Mikrokontroller Datenrahmen, die das folgende Format besitzen:

<CAN-Identifizier> <RTR> <LEN> [<DATA0>] ... [<DATA7>]

Da es sich um Datenrahmen handelt ist auch hier das RTR-Bit auf 0 festgelegt. Die Länge und das Datenformat sind abhängig von dem Typ des Sensors:

- Binärsensor: Die Länge ist auf 1 festgelegt und `DATA0` kann 0 oder 1 sein.
- Analogsensor: Die Länge ist auf 3 festgelegt. Der Sensorwert v wird als Festkommazahl übertragen. Es gilt die folgende Umrechnung:

$$v = \frac{1}{100} \cdot (\text{DATA0} \cdot 2^{16} + \text{DATA1} \cdot 2^8 + \text{DATA2})$$

- Stringsensor: `LEN` kodiert die Länge des Strings (maximal 8 Zeichen). `DATA n` beinhaltet jeweils ein Zeichen in ASCII-Kodierung.

Schnittstelle auf Feldebene

Abschnitt 3 Realisierung

Kenntnisse über die Realisierung der Schnittstellen sind notwendig, um diese flexibel an neue Anforderungen anzupassen. Für die Schnittstelle auf Zell- und Managementebene bedeutet dies z.B. eine Festlegung der Typen für Sensoren und Aktuatoren, für die Schnittstelle auf Feldebene eine Vergabe von CAN-Identifiern und für die Messdatenverbindung die Ankopplung neuer Arten von physikalischer Sensoren und Aktuatoren.

3.1 Schnittstelle auf Zell- und Managementebene

Die Schnittstelle auf Zell- und Managementebene wird durch mehrere Betriebssystemprozesse für das Echtzeitbetriebssystem QNX realisiert. Der Hauptprozess, genannt **lsockd**, kommuniziert dabei mit den Socket- und Hardwaretreiberprozessen mittels Betriebssystemnachrichten und einem Shared-Memory. Abbildung 7 zeigt die Prozessstruktur für eine typische Konfiguration.

Von **lsockd** wird **lhardwd** gestartet, der seinerseits die Hardwaretreiberprozesse je nach Systemkonfiguration initialisiert. Die Hardwaretreiberprozesse kommunizieren mit den Sensoren und Aktuatoren über die Schnittstelle auf Feldebene. Eine Ausnahme ist die Wetterstation, die direkt über den seriellen Port angeschlossen ist.

Alle Sensordaten werden, nachdem sie von den Hardwaretreiberprozessen empfangen wurden, im Shared Memory **lshds** abgelegt, von wo sie von den anderen Prozessen ausgelesen werden können. Des Weiteren enthält **lshds** die Konfigurationsdaten der Sensoren und Aktuatoren, Referenzen auf initialisierte Hardwaretreiberprozesse und weitere globale Daten.

Erhält der Hauptprozess **lsockd** einen Verbindungswunsch an seinem Server-Socket, dann wird ein neuer Socketprozess **lsockchⁿ**¹

¹ *n* ist eine laufende Nummer, die von **lsockd** vergeben wird

erzeugt, welcher die Kommunikation mit dem Client selbstständig abwickelt.

Empfängt ein Socketprozess einen Befehl zur Ansteuerung eines Aktuators, so sendet dieser eine Nachricht an den Hardwaretreiberprozess, der dann seinerseits die Steuerung übernimmt. Für die Abfrage von Sensordaten greift ein Socketprozess lediglich auf die in der zentralen Datenstruktur **lshds** vorhandenen aktuellen Sensordaten zu.

Für den Fall, dass sich ein Sensor im Event-Mode befindet und dessen Daten sich geändert haben, sendet der Hardwaretreiberprozess eine Betriebssystemnachricht an alle angemeldeten Socketprozesse. Diese reagieren auf diese Nachricht und senden eine entsprechende Meldung an ihre Clients.

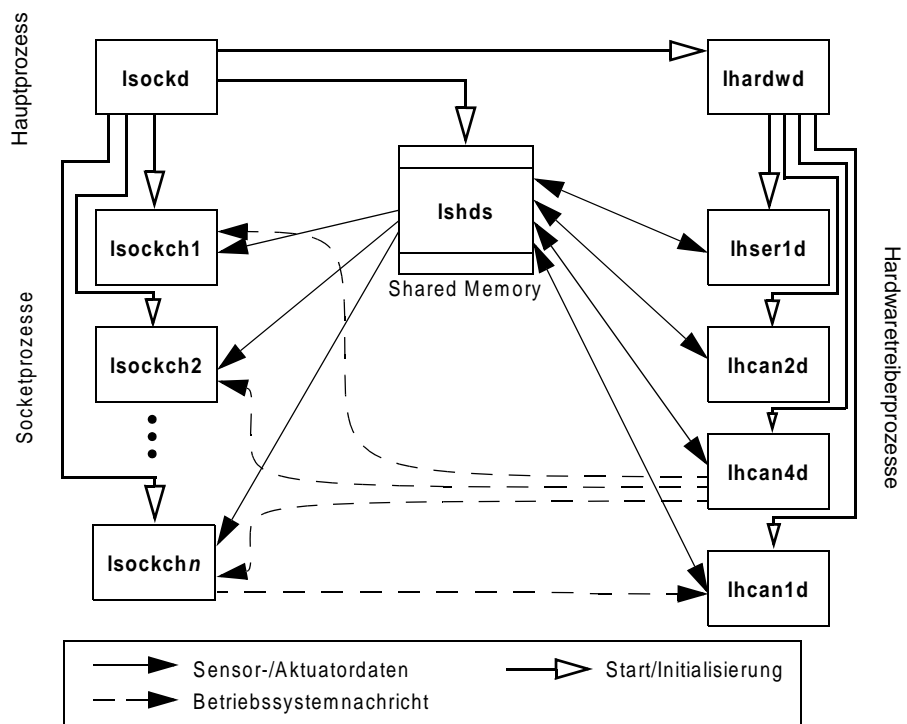


Abbildung 7 Prozessstruktur von Isockd

3.1.1 Konfiguration

Da das Testfeld für verschiedene Experimente umgebaut werden kann, muss sich auch die Schnittstelle an die veränderte Konfigura-

tion anpassen lassen. Für die Schnittstelle auf Zell- und Managementebene bedeutet dies im Speziellen eine Vergabe von Identifiern, eine eindeutige Abbildung auf das Protokoll der Schnittstelle auf Feldebene und eventuell die Integration neuer Hardwaretreiberprozesse.

Die (Re-)Konfiguration der Schnittstelle erfolgt in zwei Schritten. Zuerst werden die benötigten Konfigurationsdateien angepasst und danach die Prozesse neu gestartet.

Es existieren drei Konfigurationsdateien, die auf allen Rechnern einheitlich benannt sind: `devices`, `sensors` und `actuators`.

Konfigurationsdatei `devices`

Diese Datei beinhaltet die Namen der Prozesse, die bei Systemstart initialisiert werden sollen. Der Aufbau einer solchen Datei ist beispielhaft in der folgenden Abbildung gezeigt.

```
room 32418                                #1 Portnummer des Server-Sockets
# Name      Typ      Beschreibung
lhcan1d     A        # Ansteuerung der Aktuatoren
lhcan2d     A        # Abfrage von Sensorwerten
```

Abbildung 8 `devices`

Die erste Zeile dient der eindeutigen Kennzeichnung der Konfigurationsdaten und entspricht gleichzeitig der Portnummer des Server-Sockets.

Dann folgen zeilenweise der Name des Prozesses (bzw. der ausführbaren Datei, die den Prozesscode enthält) und eine Kennzeichnung, wie die gesteuerte Hardware initialisiert werden soll. Das `A` bedeutet, dass die Initialisierung automatisch geschieht. Wird ein `M` angegeben, muss bei Systemstart die Hardware vom Benutzer in einen definierten Grundzustand gebracht werden.

¹ In allen Dateien werden Kommentare durch `#` eingeleitet.

Konfigurationsdatei sensors

Alle über die Schnittstelle verfügbaren Sensoren werden in der Datei `sensors` definiert. An Hand des Beispiels in Abbildung 9 soll deren Aufbau erklärt werden.

```

room 32418                # Portnummer des Server-Sockets

# Name   Typ   Hardwaretreiber  ID  Beschreibung
dcc1     B    lhcan2d          700 # Tuerkontakt
rts1     R    lhcan2d          702 # Temperatursensor
kpd1     S    lhcan2d          703 # Tastatur

```

Abbildung 9 sensors

Die erste Zeile entspricht der Zeile in der Datei `devices`. Dann folgen zeilenweise Einträge, die den Identifier des Sensors, dessen Typ (siehe Tabelle 10), den für den Zugriff auf den Sensor zuständigen Hardwaretreiberprozess und einen Parameter für die Abbildung auf die Schnittstelle auf Feldebene beinhalten.

Kürzel	Beschreibung	Beispiel
B	Binärsensor (liefert die Werte 0 und 1)	Türkontakt (normally open)
I	Binärsensor, der die vom physikalischen Sensor gelieferten Werte zusätzlich invertiert (liefert die Werte 1 und 0)	Türkontakt (normally closed)
R	Analogsensor (liefert eine reelle Zahl)	Temperatursensor
S	Stringsensor (liefert einen Zeichenstring)	Tastatur
Y	Bytestringsensor (liefert einen String der Form $n_p_1_ \dots _p_n$, wobei n die Anzahl der Parameter p_i angibt, die jeweils durch das Zeichen <code>_</code> getrennt sind)	wird für die direkte Abbildung von Nachrichten auf Feldebene auf Meldungen auf Zell- und Managementebene eingesetzt

Tabelle 10 Sensortypen

Konfigurationsdatei actuators

Diese Datei ist analog zur Datei `sensors` aufgebaut. Der einzige Unterschied besteht in den Typen, die zur Spezifikation der Aktuator-

ren angegeben werden können. Tabelle 11 stellt die möglichen Ausprägungen dar.

Kürzel	Beschreibung	Beispiel
B	Binäraktuator (erlaubt die Parameter 0 und 1)	Alarmsirene
3	Aktuator mit drei Bewegungsrichtungen (erlaubt die Parameter -1, 0 und 1)	Jalousie
R	Analogaktuator (erlaubt als Parameter eine reelle Zahl)	Dimmaktuator
P	Pulsgenerator (verlangt beliebigen Parameter)	Leuchte
U	Pulsgenerator (unterscheidet sich von P nur durch die Realisierung auf Feldebene)	Leuchte
D	Dezimalaktuator (erlaubt als Parameter eine natürliche Zahl)	LEDs
Y	Bytestringaktuator (verlangt als Parameter einen String der Form $n_p_1_ \dots _p_n$, wobei n die Anzahl der Parameter p_i angibt, die jeweils durch das Zeichen $_$ getrennt sind)	wird für die direkte Abbildung von Befehlen auf Zell- und Managementebene auf Nachrichten auf Feldebene eingesetzt

Tabelle 11 Aktuatortypen

CAN-Treiberprozess

Der Zugriff auf den CAN-Bus erfolgt in der aktuellen Implementierung der Schnittstelle durch mindestens drei Hardwaretreiberprozesse. Neben diesen Prozessen müssen möglicherweise auch von der Schnittstelle entkoppelte Prozesse auf die CAN-Hardware zugreifen. Da dies zu Konflikten bei dem Zugriff auf die physikalische Hardware (die Schnittstellenkarte im PC) führt, wurde ein zusätzlicher CAN-Treiberprozess (**lcan**) eingeführt, der von der Hardware abstrahiert. Dieser Prozess reicht empfangene CAN-Nachrichten an registrierte Prozesse weiter und führt Sendewünsche von Prozessen aus. Abbildung 10 zeigt die Einbindung dieses Prozesses in das System.

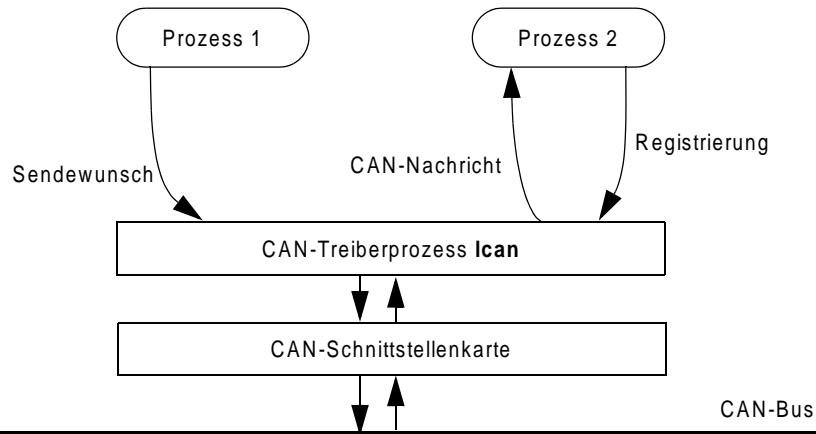


Abbildung 10 CAN-Treiberprozess

In Anhang A werden die Bibliotheksfunktionen, welche die Kommunikation mit lcan unterstützen aufgeführt.

3.1.2 Hardware

Die PCs, auf welchen die Implementierung der Schnittstelle abläuft, besitzen genügend Ressourcen, um neben den Schnittstellenprozessen auch weitere Prozesse ablaufen zu lassen. Daher wird hier die verfügbare Hardware und deren Leistungsmerkmale aufgelistet.

Es werden zwei Varianten von PCs eingesetzt: Standard PCs und Embedded PCs im PC/104-Format. Auf beiden Varianten ist das Betriebssystem QNX 4.25 mit TCP/IP-Bibliothek und dem Watcom C/C++-Compiler installiert.

Standard PC:

- Intel Pentium Prozessor, 90 bzw. 120MHz
- 16 bzw. 32MB RAM
- 2GB Festplatte
- 3,5" Diskettenlaufwerk
- 10/100MBit/s Ethernet-Karte, Medium: Twisted-Pair
- CAN-Schnittstellenkarte, Philips 82C200 Controller (CAN 2.0A [Bos91a])
- SVGA-Grafikkarte

- zwei serielle Schnittstellen (UART)
- parallele Schnittstelle (bidirektional)
- PS/2-Anschluss

Embedded PC:

- 8x486 Prozessor (AMD Élan 400), 66MHz
- 16MB RAM
- 8 bzw. 24MB Flashspeicher
- 10MBit/s Ethernet-Karte, Medium: Twisted-Pair/Koaxial
- CAN-Schnittstellenkarte, Intel 82527 Controller [Int95] (CAN 2.0B [Bos91b])
- optionale VGA-Grafikkarte
- optionale Festplatte
- optionales Diskettenlaufwerk
- drei serielle Schnittstellen (UART)
- parallele Schnittstelle (bidirektional)
- IrDA-Anschluss

Ein Benchmark, der die Leistungsfähigkeit der Rechner auf der Basis von aus Zustandsdiagrammen generiertem Applikationscode vergleicht, ist in Tabelle 16 zu finden.

3.2 Schnittstelle auf Feldebene

Die Schnittstelle auf Feldebene wird durch die 8-Bit Mikrokontroller realisiert. Auf der Basis eines Echtzeitkernels (KEIL RTX-51 [Kei96]) wird die Schnittstelle durch mehrere Tasks implementiert. Abbildung 11 zeigt die bei der Erfassung von Sensordaten und Ansteuerung von Aktuatoren beteiligten Tasks und deren Kommunikation untereinander und mit dem CAN-Medium.

Der RTX-51 CAN-Task ist Teil des Echtzeitkernels und nimmt eine Vorverarbeitung der CAN-Nachrichten vor. Empfängt dieser Task einen Datenrahmen, so wird dieser als Aktuatorbefehl gedeutet und an den zentralen MASTER-Task weitergereicht. Hier wird die eigentliche Filterung nach dem CAN-Identifizierer vorgenommen. Existiert ein Aktuator, der an die empfangene Objektidentifikation

(EPROMs) untergebracht ist. Die Modifikation erfolgt mit Hilfe eines Konfigurationswerkzeuges unter QNX auf den PCs.

Die Konfigurationsdateien, die als Eingabe in dieses Werkzeug dienen, haben dabei das in Abbildung 12 exemplarische aufgezeigte Format.

```
Modul
  Config
    CanID  102
    BTR0   67
    BTR1   47
    CTRL   202
  EndConfig

  Task    1
    Comment Heizungventilaktuator
    CanID  206
    Port   1
  EndTask

  Task    2
    Comment Analogsensoren
    CanID  401
    Port   4
    Delay  10

    Sensor
      Comment      Heizkörpertemperatur
      Nummer       1
      Intervall    10
      Offset       0
      Faktor       4097
    EndSensor
  EndTask

  Task    4
    Comment Fensterkontakte
    Port   5
    CanID  404
    Count  4
  EndTask
EndModul
```

Abbildung 12 Konfigurationsdatei für Mikrokontroller

Im `Config`-Abschnitt werden grundlegende Parameter für die CAN-Kommunikation festgelegt. Das ist zum Einen ein CAN-Identifizier, unter dem der Mikrokontroller selbst ansprechbar ist. Dieser wird z.B. verwendet um eine Neukonfiguration zu starten. Zum Anderen sind die den Registerinhalten des CAN-Controllers entsprechenden Werte `BTR0` und `BTR1` (Busgeschwindigkeit) und `CTRL` definiert.

Die Parameter der für die Sensoren und Aktuatoren zuständigen Tasks werden innerhalb von `Task`-Abschnitten festgelegt. Die Zahl

hinter dem Schlüsselwort `TASK` kennzeichnet dabei den Typ des Tasks, der abhängig vom zu steuernden Aktuator, bzw. auszulesendem Sensor ist. Tabelle 12 zeigt die möglichen Werte. Danach folgt der zuzuordnende CAN-Identifizierer. Erscheint zusätzlich das Schlüsselwort `COUNT`, bedeutet dies, dass mehrere CAN-Identifizierer beginnend bei dem angegebenen Wert linear vergeben werden.

Typ (Nr.)	Verwendung
1	Binäraktuator oder Aktuator mit drei Bewegungsrichtungen
2	bis max. 8 Analogsensoren
3	Analogaktuatoren (speziell Dimmer)
4	bis max. 4 Binärsensoren
5	bis max. 8 Binärsensoren (in Vektordarstellung)
6	– nicht genutzt –
7	Stringsensor (speziell Tastatur)
8	Dezimalaktuator (speziell LEDs)

Tabelle 12 Tasktypen für Sensoren und Aktuatoren

Bei Analogsensor-Tasks kann zusätzlich die Abtastrate und für jeden Sensor einzeln dessen Abtastdauer und die zur Umrechnung in physikalische Größen benötigte Parameter angegeben werden.

Schließlich wird nach dem Schlüsselwort `PORT` die Messdatenverbindung, über welche die Komponenten angeschlossen ist, angegeben (siehe dazu Abschnitt 3.3).

3.2.2 Hardware

Als Mikrokontroller kommt der Philips 8xC592 zum Einsatz (siehe auch [Phi96]). Dieser besitzt einen integrierten CAN-Controller mit einem identischen Registersatz zum Philips 82C200. Die weiteren Daten sind der folgenden Aufzählung zu entnehmen:

- bis zu 64kB Programmspeicher, i. d. R. ROM (Harvard-Architektur)
- bis zu 64kB Datenspeicher, RAM (bei Bedarf können davon bis zu 32kB als Programmspeicher verwendet werden, von-Neumann-Architektur)

- CAN-Kontroller nach Protokoll Spezifikation 2.0A (11-Bit Identifier), max. Bitrate 1MBit/s.
- drei 16-Bit Timer/Counter, ein 10-Bit Analog/Digital-Konverter mit acht gemultiplexten Eingängen und zwei 8-Bit pulsbreitenmodulierte Ausgänge
- fünf 8-Bit Ein-/Ausgabeports und ein 8-Bit Eingabeport
- serielle Schnittstelle (UART)
- Watchdog Timer
- Taktfrequenz 16MHz.

physikalische CAN-Anbindung

Die gewählte Steckerbelegung entspricht der Spezifikation der CiA (CAN in Automation [CiA94]). Die folgende Abbildung und Tabelle 13 zeigen die Belegung der verwendeten neunpoligen D-Sub-Stecker.

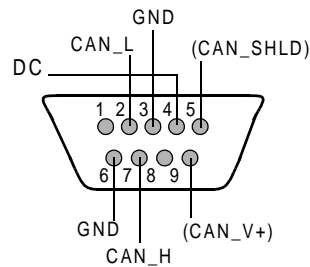


Abbildung 13 D-Sub-Steckerbelegung (männlich) für CAN-Busankopplung

Pin	Signal	Beschreibung
1	-	reserviert
2	CAN_L	CAN_L Bus Leitung (dominant low)
3	GND	Masse
4	DC	+12V DC Stromversorgung
5	(CAN_SHLD)	opt. CAN Abschirmung
6	(GND)	opt. Masse
7	CAN_H	CAN_H Bus Leitung (dominant high)
8	-	reserviert (Fehlerleitung)

Tabelle 13 Pinbelegung der D-Sub-Stecker

Pin	Signal	Beschreibung
9	(CAN_V+)	opt. ext. CAN Stromversorgung (für Transceiver und Optokoppler)

Tabelle 13 Pinbelegung der D-Sub-Stecker

Für die flexible Verkabelung des Testfelds wurde die in Abbildung 14 schematisch dargestellte Verbindungstechnik eingesetzt. Einzelne Bussegmente werden durch Buskabel realisiert, die an einem Ende einen männlichen am anderen Ende einen weiblichen Konnektor besitzen. Mittels T-Verbindern können die Busteilnehmer, die einen männlichen Konnektor besitzen, an den Hauptstrang angekoppelt werden. Dabei können bei Bedarf zusätzliche Buskabel (Stichleitungen) eingesetzt werden.

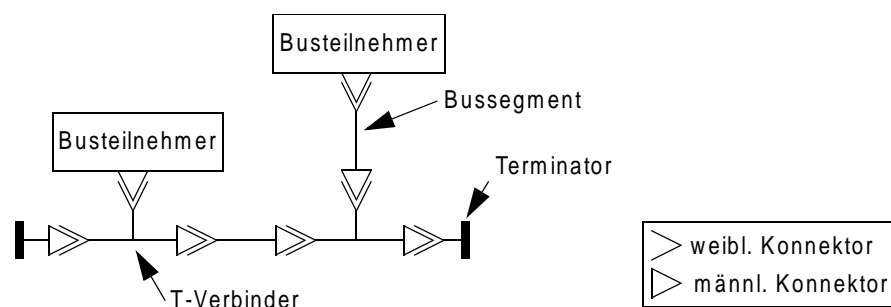


Abbildung 14 Verbindungstechnik (schematisch)

Der Hauptstrang ist an beiden Enden mit Terminatoren zu versehen, um Leitungsreflexionen zu vermeiden. Dazu werden die Abschlusswiderstände, wie in Abbildung 15 gezeigt, im „Split Termination“-Prinzip aufgebaut (siehe auch [Phi00], S. 15).

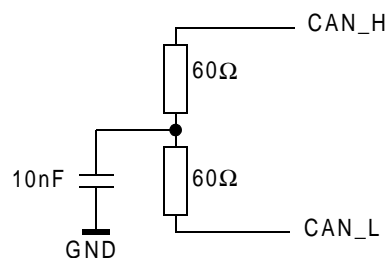


Abbildung 15 Aufbau eines Busterminals

3.3 Schnittstelle auf Ebene der Messdatenverbindung

Auf der Ebene der Messdatenverbindungen wurde ein modulares Konzept zur Realisierung einer flexiblen Schnittstelle eingesetzt. Die Ein-/Ausgabepins der Mikrokontroller wurden dazu so zusammengefasst, dass vorgefertigte komplexere Module zur Ansteuerung von Aktuatoren und zum Messen von Sensordaten angekoppelt werden können. Die technischen Details dieses Konzeptes sind in [Dis99] zu finden und sollen hier zusammengefasst dargestellt werden.

Der Philips 8xC592 Mikrokontroller erlaubt in der verwendeten Hardwareeinbettung die Ankopplung an die Umwelt über 16 digitale Ein-/Ausgabepins (**I/O-Pins**) und 8 Eingabepins (**I-Pins**), welche digitale oder analoge Eingangssignale verarbeiten können. Wie in Tabelle 14 gezeigt, werden diese Pins in 4er Gruppen zu insgesamt sechs **Ports** zusammengefasst.

Port	Pin	Pinbezeichnung P8xC592	Porttyp
1	0	P1.0	I/O
	1	P1.3	I/O
	2	P1.5	I/O
	3	P3.4	I/O
2	0	P3.0	I/O
	1	P3.1	I/O
	2	P3.2	I/O
	3	P3.5	I/O
3	0	P4.0	I/O
	1	P4.1	I/O
	2	P4.2	I/O
	3	P4.3	I/O

Tabelle 14 Gruppierung von Pins zu Ports

Port	Pin	Pinbezeichnung P8xC592	Porttyp
4	0	P4.4	I/O
	1	P4.5	I/O
	2	P4.6	I/O
	3	P4.7	I/O
5	0	P5.0/ADC0	I
	1	P5.1/ADC1	I
	2	P5.2/ADC2	I
	3	P5.3/ADC3	I
6	0	P5.4/ADC4	I
	1	P5.5./ADC5	I
	2	P5.6/ADC6	I
	3	P5.7/ADC7	I

Tabelle 14 Gruppierung von Pins zu Ports

In Tabelle 15 ist die Belegung der verwendeten Pfostenstecker, die auf der Grundplatine des Mikrokontrollermoduls angebracht sind, beschrieben. Jeder Port besitzt eine +5 V Stromversorgung und Masse (GND). Bei den digitalen I/O-Ports ist zusätzlich ein allen Ports gemeinsamer Zählereingang (T2) und Interrupteingang (Int1) vorhanden. Die reinen Eingabeports (I-Ports) verfügen über zwei Referenzspannungen (Vref+ und Vref-) und einen Triggereingang zum Start der Analog-Digital-Wandlung (StADC).

Pin	digitaler I/O-Port	analog/digitaler I-Port
1	Pin 0	Pin 0
2	T2	Vref-
3	Pin 1	Pin 1
4	Int1	Vref+

Tabelle 15 Belegung der Pfostenstecker

Pin	digitaler I/O-Port	analog/digitaler I-Port
5	Pin 2	Pin 2
6	GND	GND
7	Pin 3	Pin 3
8	+5V	+5V
9	nicht belegt	nicht belegt
10	nicht belegt	nicht belegt

Tabelle 15 Belegung der Pfostenstecker

Abschnitt 4

Experimente und Fallstudien

Das in den vorherigen Abschnitten vorgestellte Testfeld wurde innerhalb des SFB 501 in verschiedenen Experimenten und Fallstudien¹ eingesetzt. Diese werden in den folgenden Unterabschnitten vorgestellt und qualitative und quantitative Ergebnisse präsentiert.

4.1 Fallstudie „Anforderungsanalyse“

Im Rahmen der Fallstudie CS3/2 [QuZ99] wurde von einem Team aus acht Entwicklern die Anforderungsspezifikation eines komplexen Gebäudeautomationssystem erarbeitet. Ein Ergebnis dieser Fallstudie ist ein SDL-Modell, welches die Anforderungen operational mit Hilfe von Prozessen und Zustandsdiagrammen beschreibt. Dieses Modell wurde mit Hilfe der Prototyping-Umgebung **ProtoEnv** (siehe Abschnitt 4.2) des Teilprojekts D1 in einen ablauffähigen Prototyp überführt, wobei die Generierung eines solchen Prototyps jeweils ca. zehn Minuten benötigte.

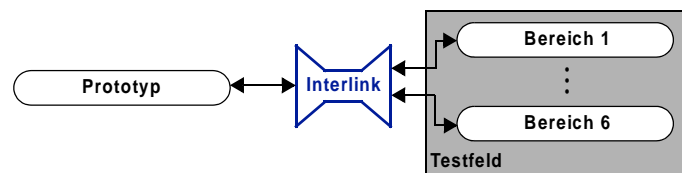


Abbildung 16 Interlink

Nach der durch Prototyping ermöglichten Validierung und der Verifikation mit Hilfe von Simulatoren wurde der Prototyp über die Schnittstelle auf Zell- und Managementebene an das Testfeld angekoppelt. Um diese Ankopplung zu erreichen, wurde die Komponente **Interlink** [Met99] entwickelt, welche, wie in Abbildung 16 angedeutet, eine Verbindung der Socketschnittstelle des Prototyps zu den

¹ Eine exakte Differenzierung von Experiment und Fallstudie wollen wir hier nicht vornehmen.

Sockets im Testfeld erlaubt. Zusätzlich wurde dieses Werkzeug genutzt, um Instanzennamen des Prototyps auf Identifier des Testfelds abzubilden und die im SDL-Modell zur Vereinfachung eingefügten Kopien von Sensoren aufzulösen.

Ergebnisse

Da das Testfeld als „living laboratory“ im täglichen Betrieb genutzt wurde, ließen sich viele Fehler oder Verbesserungspunkte identifizieren, die beim Testen mit Hilfe von Simulatoren nicht festgestellt wurden. Exemplarisch sollen hier einige dieser Punkte aufgezählt werden:

- Das Verhalten der Raum- und der Flurhelligkeit war fehlerhaft modelliert. Erst bei Aufenthalt in einem automatisierten Raum wurde festgestellt, dass die eingestellte Helligkeit nicht sinnvoll war. Die Fehlerursache lag in einer falsch umgesetzten Strategie.
- Eine Strategie in der Spezifikation sah vor, in Abhängigkeit vom Lichteinfall in den Flur das Flurlicht bei Bedarf ein- bzw. auszuschalten. Dies führte im Testbetrieb zu der Extremsituation, dass bei kurzem Öffnen einer Bürotür das Licht aus und dann sofort wieder eingeschaltet wurde. Dies hat sich als sehr störend herausgestellt und war auch für die Lebensdauer der Leuchten nicht sehr zuträglich.
- Viele der Zeiten, die z.B. zur Feststellung einer Raumbelugung verwendet werden, waren nicht passend gewählt. Die Optimierung dieser Parameter durch Prototyping wurde bewusst forciert, da nicht genügend Erfahrung bezüglich sinnvoller Voreinstellungen vorhanden war.
- Als störend wurde auch das Zurücksetzen einer vom Benutzer eingestellten Lichtszene auf eine Default-Lichtszene empfunden.

Desweiteren wurden noch ca. 10 Fehler identifiziert, die erst auf Grund der komplexeren „Testszenarien“ zu Tage traten. Es waren oftmals inkorrekt modellierte Zustandsübergänge und besonders bei der Initialisierung Annahmen über Signalreihenfolgen, die zu einem Fehlverhalten führten.

4.2 Fallstudie „Verteiltes Prototyping“

Um die sich an die Anforderungsanalyse anschließende Entwurfsphase zu unterstützen, soll Prototyping auch verteilt ermöglicht wer-

den. Deswegen wurde mit Hilfe des auf die physikalisch existierenden Räume (siehe Abbildung 1) reduzierten SDL-Modells aus Fallstudie CS3/2 eine weitere Fallstudie durchgeführt. Diese hatte zum Ziel, das Modell zu partitionieren und daraus einen verteilten Prototyp zu erzeugen, welcher auf den PCs des Testfelds ablauf-fähig ist.

Bei dieser Fallstudie handelte es sich zunächst einmal um eine Mach-barkeitsanalyse, die mit fest vorgegebenen Annahmen und einfachen technischen Mechanismen die Erzeugung eines verteilten Prototyps zum Ziel hatte. Ein Teil dieser Erzeugung wurde manuell durchge-führt und es wurde untersucht in wie weit sich Schritte automatisie-ren lassen. Besonders die Probleme, die sich bei der Verteilung auf kleinere Hardwarekomponenten ergaben, sollten untersucht werden.

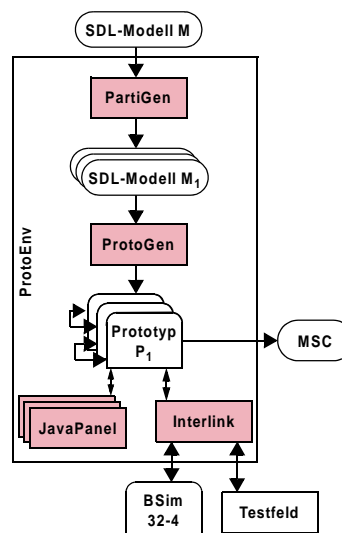


Abbildung 17 ProtoEnv

In Abbildung 17 ist die Prototyping-Umgebung **ProtoEnv** mit den einzelnen Aktivitäten, bzw. Werkzeugen, und Dokumenten gezeigt. Mit dieser Umgebung wurde die Erstellung der verteilten Prototypen durchgeführt. Die Partitionierung des SDL-Modells wird dabei von **PartiGen** übernommen. Nachdem einzelne Teilmodelle erzeugt wurden, werden mit Hilfe von **ProtoGen** ausführbare Prototypen erzeugt, welche dann mit Hilfe des Interlink und der **JavaPanels** an die Umgebung angekoppelt werden können.

Zunächst wurde mit Hilfe einer einfachen Kostenfunktion eine Partitionierung durchgeführt, und diese Partitionierung mit Hilfe des Prototyps evaluiert. In einem anschließenden Durchlauf konnten dann Schwachstellen eliminiert werden und eine bessere Partitionierung erhalten werden.

Für die erste Partitionierung sollten die folgenden Aspekte berücksichtigt werden:

- **Topologie des Testfelds:** Die einzelnen Prototyp-Partitionen sollen auf den Embedded PCs ablaufen und können daher nur auf lokale Sensoren und Aktuatoren zugreifen
- **Struktur des Steuerungssystems:** Eine Modifikation der Struktur (d.h. eine Auflösung oder Umordnung der existierenden Objekte) wäre zu aufwändig und soll daher nicht durchgeführt werden
- **Kommunikationsaufkommen:** Die Kommunikation zwischen den einzelnen Partitionen sollte möglichst gering sein
- **Ausfallsicherheit:** Auch bei Ausfall einer oder mehrerer Partitionen sollte eine Minimalfunktionalität der anderen Partitionen gewährleistet sein.

Das Ergebnis war eine Partitionierung in sieben Partitionen wie sie in Abbildung 18 an Hand eines abgewandelten UML Klassendiagramms gezeigt sind.

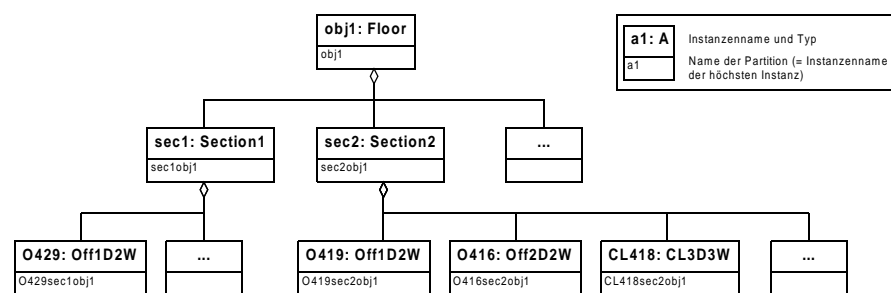


Abbildung 18 Initiale Partitionierung (Instanzendiagramm)

Dieses so partitionierte System ließ sich wegen der in der Modellierung eingeführten Kopien von Sensoren und wegen des notwendigen Zugriffs auf entfernte Komponenten auf Feldebene nicht im Testfeld zum Ablauf bringen. Zur Verdeutlichung zeigt Abbildung 19 das Problem der Kopien am Beispiel eines Türkontakts.

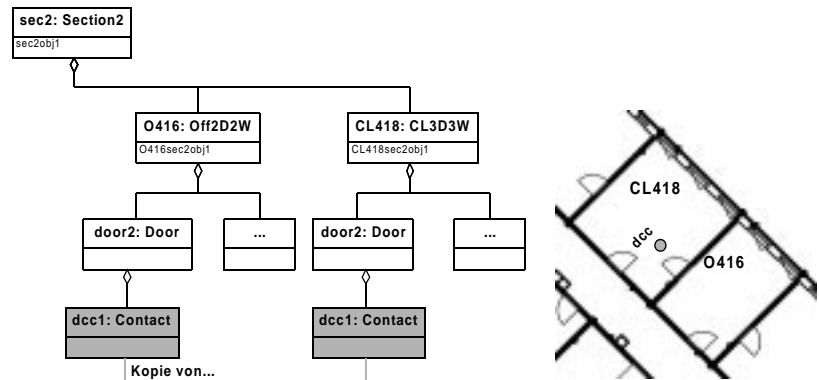


Abbildung 19 Kopien von Sensoren

Der Türkontakt existiert doppelt im SDL-Modell. Physikalisch jedoch befindet sich der Türkontakt in CL418 und ist daher dort über den Feldbus an den Embedded PC angebunden. Raum O416 hat keinen Zugriff auf diesen Sensor, da die Feldbusse beider Räume physikalisch nicht verbunden sind. Da die Kommunikation der Partitionen untereinander jedoch über Ethernet auf Zell- und Managementebene erfolgt, lässt sich durch geschicktes (Re-)partitionieren dieses Dilemma lösen.

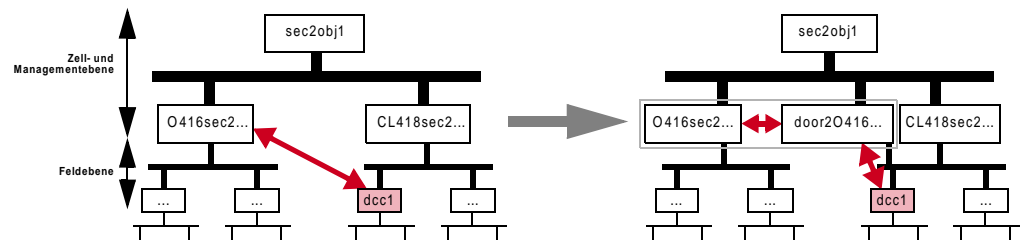


Abbildung 20 (Re-)partitionierung zur Lösung des Zugriffproblems auf entfernten Bus

Abbildung 20 zeigt die prinzipielle Vorgehensweise am Beispiel des Türkontakts. (Die einzelnen Partitionen sind dabei jeweils Rechnerkomponenten der entsprechenden Ebenen zugeordnet). Der gewünschte direkte Kanal von Partition *O416sec2obj1* zu Sensor *dcc1* wird ersetzt durch zwei Kanäle, die von *O416sec2obj1* via *door2O416sec2obj1* zu *dcc1* verlaufen. Dabei entstehen *O416sec2obj1* und *door2O416sec2obj1* durch Partitionierung der ursprünglichen Partition *O416sec2obj1*. Die Partition

door2O416sec2obj1 erhält durch Zuordnung zu einem Embedded PC in CL418 den Zugriff auf den lokalen Feldbus.

Wendet man dieses Vorgehen auf alle Sensorkopien und entfernte Komponenten an, erhält man eine Partitionierung wie sie in Abbildung 21 gezeigt ist.

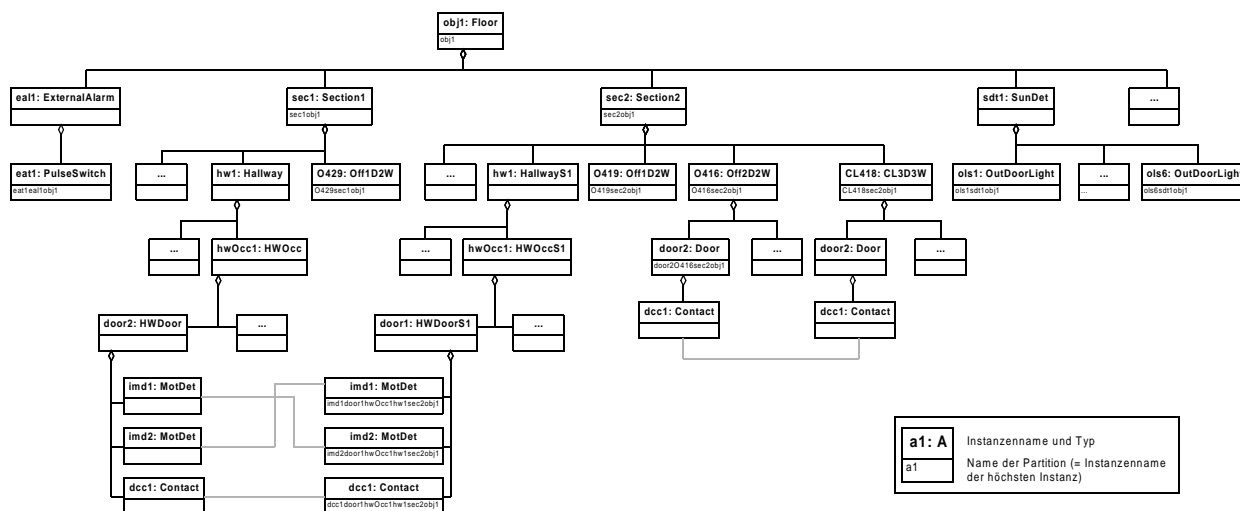


Abbildung 21 (Re-)partitionierung zur Lösung des Kopien-Problems

Aus dem so partitionierten Modell wurden anschließend Prototypen generiert und im Testfeld zum Ablauf gebracht.

Ergebnisse

Der Aufwand für die manuelle Partitionierung des SDL-Modells betrug insgesamt 674 Minuten. Abbildung 22 zeigt die Aufwandsverteilung für die einzelnen Komponenten. *Floor* beinhaltet den Aufwand für die Integration der für die Verteilung benötigten neuen Kommunikationskanäle in die existierenden Komponenten des Systems. Man erkennt die Lernkurve, welche durch Reuse schon modellierter Komponenten entsteht.

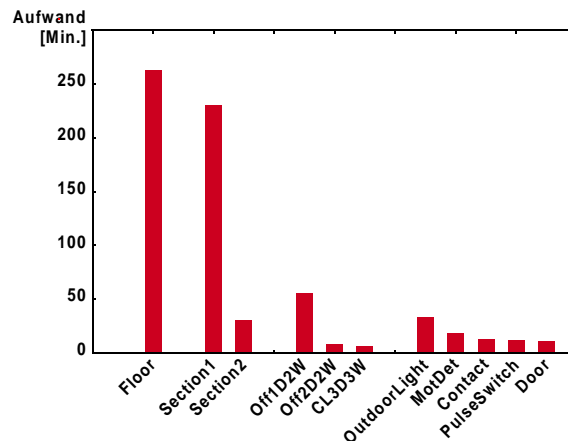


Abbildung 22 Aufwandsverteilung der manuellen Partitionierung

Aus dem hohen Aufwand für diese Partitionierung lässt sich folgern, dass eine Werkzeugunterstützung unumgänglich ist, wenn man Prototyping für den Entwurfsschritt „Partitionierung“ sinnvoll einsetzen und die Zykluszeiten in einer vernünftigen Größenordnung halten will.

Als deutliches Problem dieser „initialen“ Partitionierung hat sich die Trägheit der Prototypen herausgestellt. Auf Grund der relativ rechen-schwachen PCs und der trotz Aufteilung noch immer komplexen Partitionen liegt die Reaktionszeit des Systems im Extremfall bei bis zu zwei Sekunden. Der Vergleich der Rechenleistung mit anderen Systemen zeigt dramatische Leistungsdifferenzen (siehe Tabelle 16). Als Benchmark wurde die bei mehreren Prototypen erreichte durchschnittliche Anzahl an Transitionen pro Sekunde gewählt. Transitionen treten typischerweise bei der Abarbeitung der SDL-Zustandsdiagramme auf.

Plattform	Transitionen/Sekunde (Durchschnitt)
Embedded PC (8x486, 66MHz)	35
PC (Pentium 120MHz)	605
Workstation (HP-PA RISC)	1830

Tabelle 16 Leistungsvergleich

Man muss sich nun Gedanken machen, ob man schnellere Hardware einsetzt, oder aber ob die Hardware auf die endgültige Zielhardware skaliert und deshalb die Ausführungszeiten der Prototypen die Zeiten für das Endprodukt widerspiegeln. Ist letzteres der Fall, so muss die Rechenleistung der PCs in die Kostenfunktion für die Partitionierung aufgenommen werden und eine Repartitionierung durchgeführt werden.

Als problematisch hat sich zusätzlich das Fehlen eines auf die Anwendung angepassten Kommunikationssystems herausgestellt. Arbeiten zum Einsatz eines solchen Systems wurden daher in Verbindung mit Teilprojekt B4 initiiert.

Anhang

A Bibliotheksfunktionen für `lcan`

In diesem Abschnitt werden die Funktionen und Datenstrukturen für den Zugriff auf den CAN-Treiberprozess `lcan` aufgeführt.

Die zu übertragenden oder zu empfangenden CAN-Nachrichten werden in der Datenstruktur `CANMSG` gespeichert:

```
typedef struct {
    unsigned short int ID ;          /* 11-bit identifier */
    unsigned char    RTR ;          /* true, if remote request */
    unsigned char    LEN ;          /* data length code (0..8) */
    unsigned char    DATA[8] ;     /* data: 0...7 bytes */
} CANMSG;
```

Diese Datenstruktur beinhaltet den CAN-Identifizierer (`ID`) mit 11 Bit Länge, das Remote-Transmission-Request-Bit (`RTR`), die Anzahl der CAN-Daten (`LEN`) und die Nutzdaten selbst (`DATA[n]`).

A.1 `CANDriver_Init()`

```
int CANDriver_Init()
```

Diese Funktion muss ausgeführt werden bevor die anderen Bibliotheksfunktionen verwendet werden können. Sie initialisiert die Verbindung zum CAN-Treiberprozess `lcan`.

Rückgabewert: `-1`: ein Fehler ist aufgetreten, der Treiberprozess kann nicht angesprochen werden

`0`: kein Fehler

A.2 `CANDriver_Write`

```
int CANDriver_Write(CANMSG *msg)
```

Mit dieser Funktion können CAN-Nachrichten versendet werden. CAN-Nachrichten, die mit `CANDriver_write()` an **lcan** geschickt werden, werden automatisch auch an alle angemeldeten Prozesse weitergereicht (siehe Abschnitt A.3).

Rückgabewert: -1: Der CAN-Treiberprozess ist nicht initialisiert oder die Kommunikation mit diesem ist fehlgeschlagen

0: kein Fehler

A.3 CANDriver_Register

```
int CANDriver_Register()
```

Ruft ein Prozess diese Funktion auf, so wird der Prozess bei **lcan** registriert, d.h. empfängt **lcan** eine CAN-Nachricht, so wird diese an den angemeldeten Prozess weitergereicht.

Rückgabewert: -1: Der CAN-Treiberprozess ist nicht initialisiert oder die Kommunikation mit diesem ist fehlgeschlagen

0: kein Fehler

A.4 CANDriver_Unregister

```
int CANDriver_Unregister()
```

Ruft ein Prozess diese Funktion auf, so wird der Prozess bei **lcan** aus der Registrierung entfernt. CAN-Nachrichten werden nicht mehr an diesen Prozess gesendet.

Rückgabewert: -1: Der CAN-Treiberprozess ist nicht initialisiert oder die Kommunikation mit diesem ist fehlgeschlagen

0: kein Fehler

A.5 CANDriver_Receive

```
int CANDriver_Receive(CANMsg *msg)
```

Hat sich ein Prozess bei **lcan** registriert kann mit `CANDriver_Receive()` auf die Ankunft einer CAN-Nachricht gewartet werden. Der Funktionsaufruf ist blockierend, d.h. die Funktion kehrt solange nicht zurück, bis eine Nachricht eintrifft.

In der `msg` Datenstruktur wird die empfangene CAN-Nachricht gespeichert.

Rückgabewert: -1: Der CAN-Treiberprozess ist nicht initialisiert oder die Kommunikation mit diesem ist fehlgeschlagen

0: kein Fehler

A.6 CANDriver_Creceive

```
int CANDriver_Creceive(CANMsg *msg)
```

Hat sich ein Prozess bei **lcan** registriert kann mit `CANDriver_Creceive()` auf die Ankunft einer CAN-Nachricht gewartet werden. Der Funktionsaufruf ist *nicht*-blockierend, d.h. liegt keine Nachricht an, kehrt die Funktion sofort zurück und liefert -1.

War eine Nachricht vorhanden, so wird in der `msg` Datenstruktur die empfangene CAN-Nachricht gespeichert.

Rückgabewert: -1: Der CAN-Treiberprozess ist nicht initialisiert, die Kommunikation mit diesem ist fehlgeschlagen oder keine Nachricht liegt an

0: kein Fehler

Literaturverzeichnis

- [Bos91a] *CAN Specification Version 2.0 – Part A*. Stuttgart: Robert Bosch GmbH. 1991
<http://www.can-cia.de/CAN20A.pdf>
- [Bos91b] *CAN Specification Version 2.0 – Part B*. Stuttgart: Robert Bosch GmbH. 1991
<http://www.can-cia.de/CAN20B.pdf>
- [CiA94] *CiA Draft Standard 102 Version 2.0 – CAN Physical Layer for Industrial Applications*. Erlangen: CAN in Automation (CiA). 1997
<http://www.can-cia.de/DS102.pdf>
- [Dis99] T. Disper. *Gebäudeautomation mit CAN auf Basis des Philips P8xC592*. Projektarbeit. Fachbereich Informatik. Universität Kaiserslautern. 1999
- [Int95] *82527: Serial communications controller – Controller Area Network*. Data Sheet. Santa Clara, Calif., USA: Intel Corporation. 1995
- [Kei96] *RTX-51, RTX-251 Real-Time Multitasking Executives for the 8051 and MCS[®] 251 Microcontrollers*. Users' Guide. Grasbrunn: Keil Elektronik GmbH. 1996
- [MQS96] A. Metzger, S. Queins, B. Schürmann. *Installation eines Testraums zur Gebäudeautomation und deren Schnittstelle*. SFB 501 Bericht Nr. 13/96. Fachbereich Informatik. Universität Kaiserslautern. 1996
- [Met99] A. Metzger. *An Interlink of Building Control System Prototypes and the Lighting Simulation Lumina*. Bericht 8/99. Universität Kaiserslautern. 1999
- [Phi96] *P8xC592 8-bit microcontroller with on-chip CAN*. Data Sheet. Eindhoven, Niederlande: Philips Semiconductors. 1996
<http://www-eu.semiconductors.philips.com/acrobat/3052.pdf>
- [Phi00] *TJA 1050 – CAN High-Speed Transceiver*. Application Note AN00020. Hamburg: Philips Semiconductors. 2000
<http://www.semiconductors.com/acrobat/applicationnotes/AN00020.pdf>

-
-
- [QNX01] *QNX System Architecture*. Online Document. QNX Software Systems. 2001
http://www.qnx.com/literature/qnx_sysarch/index.html
- [QuZ99] S. Queins, G. Zimmermann. *A First Iteration of a Reuse-Driven, Domain-Specific System Requirements Analysis Process*. SFB 501 Bericht 13/1999. Universität Kaiserslautern. 1999
- [Sch97] B. Schürmann. *Rechnerverbindungsstrukturen – Bussysteme und Netzwerke*. Braunschweig, Wiesbaden: Friedr. Vieweg & Sohn Verlagsgesellschaft. 1997