# Datenkommunikation

# zwischen

# Roboter und Computer

---

Diplomarbeit im Fach Informatik

vorgelegt von

**Daniel Regenass**
Reinhold Frei-Str. 62
8049 Zürich

Matrikelnummer 91-712-794

**Angefertigt am**
**Institut für Informatik**
**der**
**Universität Zürich**

**Prof. Dr. R. Pfeifer**

Betreuer: Marinus Maris

Abgabe der Arbeit: 2. April 1996

# VORWORT

Die vorliegende Arbeit entstand am Institut für Informatik in der Fachgruppe für Künstliche Intelligenz als Teil eines umfangreicheren Projektes zur Entwicklung von Robotern, die in Vorlesungen und Übungen an der Universität Zürich eingesetzt werden sollen. Sie basiert daher auch auf Arbeiten anderer Personen, die ich an dieser Stelle erwähnen möchte.

Allen voran Marinus Maris, der Entwickler der Roboter, die für diese Arbeit verwendet wurden. Seine Hardware, seine vorbehaltlose Unterstützung sowie seinen ungebrochenen Enthusiasmus halfen mir diese Arbeit erfolgreich fertigzustellen. Auch die bestehende Systemsoftware von René Schaad gaben mir eine gute Basis für umfangreiche Erweiterungen.

Nicht zuletzt gebührt mein Dank auch Prof. Dr. R. Pfeifer, Leiter der Fachgruppe für Künstliche Intelligenz. Durch ihn wurde diese Diplomarbeit erst ermöglicht und konnte in diesem angenehmen Umfeld durchgeführt werden.

*'Und seh dass wir nichts wissen können,*
*Das will mir schier das Herz verbrennen.*
*Zwar bin ich gescheiter als alle die Laffen*
*Docktors, Professors, Schreiber und Pfaffen*
*Mich blagen keine Skrupel noch Zweifel*
*Fürcht mich weder vor Höll noch Teufel.*
*Dafür ist mir auch all Freud entrissen*
*Bild mir nicht ein ich könnt was lehren*
*Die Menschen zu bessern und zu bekehren'*

*(Urfaust, J. W. Goethe)*

# INHALTSVERZEICHNIS

# ABBILDUNGSVERZEICHNIS

# 1 EINFÜHRUNG

Der heutige Stand der Kommunikationstechnik bietet viele Möglichkeiten, um den Datenaustausch zwischen einzelnen Teilnehmern eines Netzes zu ermöglichen. Es ist dabei von entscheidender Bedeutung, welche Auswahl von technischen Verfahren zur Anwendung gelangen, da dadurch die Leistungsmerkmale eines Kommunikationsnetzes bestimmt werden. Das Ziel besteht nun, möglichst die gesetzten Leistungsmerkmale zu erreichen.

Dieses Dokument soll nun genau diesen Entscheidungsprozess im Rahmen dieses Projektes widerspiegeln. Dazu ist als erstes eine Analyse der Ausgangslage notwendig. Diese gibt Auskunft über Variablen, die sich zur Zielerreichung verändern lassen, sowie über Konstanten, die als vordefinierte Eigenschaften des Systems angesehen werden müssen (Kapitel 2).

Ist die Ausgangslage nun ausreichend bekannt, so muss als nächstes die Aufgabenstellung (Kapitel 3) betrachtet werden. Sie definiert, welche Ziele am Ende des Projektes erreicht werden sollen.

Nun muss geklärt werden, auf welchem Wege diese Ziele erreicht werden können. Es gilt, eine Aufzählung der möglichen Alternativen zu geben, aus deren Menge am Ende jeweils eine gewählt werden muss und bildet damit die Grundlage (Kapitel 4) für den Designentscheid. Unter der Voraussetzung, dass die Ziele realisierbar sind, muss mindestens eine Kombination der Variablenwerte zum gewünschten Ziel führen.

Die Kombination der Variablen findet während des Systemdesigns (Kapitel 5) statt. Jede Variable wird analysiert, die verschiedenen Möglichkeiten einander gegenübergestellt und eine Entscheidung getroffen, die am Ende implementiert wird. Um das effektiv implementierte Programm transparent und erweiterbar zu halten, müssen die internen Abläufe beschrieben werden (Kapitel 6).

Dieser Vorgang läuft nicht wie hier in diesem Dokument Schritt für Schritt ab, vielmehr ist es ein Vor und Zurück, bis eine befriedigende Lösung erreicht wird. Doch damit schliesst der Entwicklungsprozess noch nicht ab. Am Ende sollten noch Schlussfolgerungen gezogen werden, dessen Erfahrungsschatz bei neuen Projekten Zeit sparen helfen. Daher folgt diesem Entwicklungsprozess eine Schwachstellenanalyse (Kapitel 7), die dann die Grundlage bildet, um das Projekt definitiv zu beurteilen (Kapitel 8). Zum Schluss soll noch ein Ausblick gegeben werden, um Erweiterungsmöglichkeiten aufzuzeigen (Kapitel 9).

# 2 AUSGANGSLAGE

In dieser Arbeit geht es darum, ein bestehendes Entwicklungssystem eines Roboters zu erweitern. Doch bevor die genaue Aufgabenstellung dieses Projektes gezeigt wird, sollte ein Überblick über das existierende Entwicklungswerkzeug gegeben werden. Es besteht grob aus zwei Komponenten, nämlich dem Roboter selbst sowie einer Entwicklungsumgebung, die auf einem IBM PC kompatiblen Computer mit Windows 3.1 bzw. Windows 95 läuft. Im folgenden werden diese beiden Komponenten vorgestellt.

## 2.1 DER DIDABOT

Für die Fachgruppe Künstliche Intelligenz an der Universität Zürich sollte eine neue Familie von Robotern entworfen werden. Das Ziel bestand darin, einen kleinen, handlichen Allzweck-Roboter zu entwickeln, der sich über einen Computer programmieren lassen sollte. Dadurch sollte ein Werkzeug geschaffen werden, an dem sich Studenten und Forscher in der Roboterprogrammierung üben können. Der Name dieses Roboters, Didabots, ist daher auch abgeleitet vom Englischen ‘didactical robot’ [MS95].

Der Didabot (Abbildung 2.1) besteht im wesentlichen aus 3 Komponenten. Erstens, aus der Prozessorplatine von Intel, bestückt mit einem 20 Mhz getakteten 80C196KD Mikrokontroller [INT92]. Es handelt sich dabei um einen Prozessor mit einem 16 Bit Adressbus, womit er maximal 64 KB ansprechen kann. Im Moment befinden sich jedoch 32 KB nicht-flüchtiges RAM auf dem Prozessorboard, was auch für grössere Programme ausreichen sollte. Desweiteren befindet sich eine serielle Schnittstelle auf dieser Platine, wodurch eine Verbindung zu einem PC hergestellt werden kann. Eine Bootsoftware auf dem Didabot namens ERISM [RE95] ermöglicht die Übertragung von Programmen und stellt ausserdem Debugger Funktionen zur Verfügung.

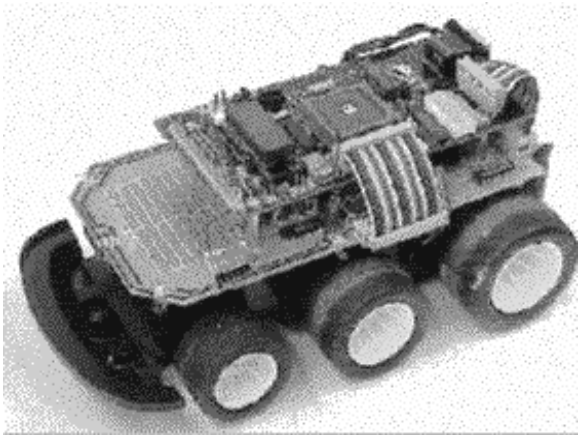

*Abbildung 2.1- Der Didabot*

Zweitens besteht der Didabot aus einer selbstentwickelten Platine, die die nötigen Schaltungen zur Steuerung der Motoren sowie die Sensoren enthält. Über 6 Infrarotsender und -empfänger rund um diese Platine ermöglichen Distanzmessungen zu Hindernissen in 6 verschiedene Richtungen. Ausserdem ist es mit diesen Sensoren möglich, die Raumhelligkeit zu messen. Weiter befindet sich ein Lautsprecher sowie eine Lampe auf dieser Platine, die zu Kontrollzwecken verwendet werden können. Wenige Modelle des Didabot wurden zusätzlich mit 9 Berührungssensoren ausgerüstet, die ebenfalls rund um die Platine angebracht sind und für spezielle Anwendungen gedacht sind.

Die dritte Komponente des Didabots ist die Chassis, wo zwei Motoren für den Antrieb sorgen. Ein Motor treibt dabei jeweils die beiden hinteren Räder einer Seite an und ermöglichen Geschwindigkeiten von bis zu 5 m/s. Die ganze Chassis wurde von einem von Tyco vertriebenen ferngesteuerten Spielzeugmodell namens 'Scorcher' entnommen, worauf die beiden Platinen befestigt und verkabelt wurden.

Der Didabot wurde durch einen Mitarbeiter der Fachgruppe Künstliche Intelligenz, Marinus Maris, in kleinen Stückzahlen entwickelt. Die Systemsoftware des Roboters wurde programmiert von René Schaad, einen weiteren Mitarbeiter obiger Fachgruppe.

## 2.2 DAS DIDABOT ENTWICKLUNGSSYSTEM

Die Idee des Didabot Entwicklungssystems (kurz DDS für 'Didabot Development System') [RE95] besteht darin, dass auch während der Ausführung eines Programmes auf dem Didabot Informationen über den aktuellen Status dieses Programmes auf dem PC verfügbar sind. Dies macht es zwangsläufig notwendig, dass ein Programm des Didabots während seiner Ausführung Daten an den PC schicken kann. Da aber die Bedeutung dieser Daten extrem variieren kann, ist es nicht möglich, ein einziges PC Programm mit dem Didabot kommunizieren zu lassen, das die ganze Kommunikation mit dem Roboter übernimmt. Um mehreren PC Anwendungen die Kommunikation mit dem Didabot zu ermöglichen, ist also ein Multiplexing System vonnöten. Dieses existiert in Form des DDS Servers (Abbildung 2.2), das die zu übertragenden Daten von den Clientapplikationen entgegennimmt und über die serielle Schnittstelle gemäss einem Multiplexing Protokoll überträgt. Genauso werden Daten vom Roboter an die entsprechende Applikation weitergeleitet. Die Kommunikation der einzelnen Applikationen mit dem DDS Server geschieht über DDE ('Dynamic Data Exchange'), ein Systemdienst von Windows.

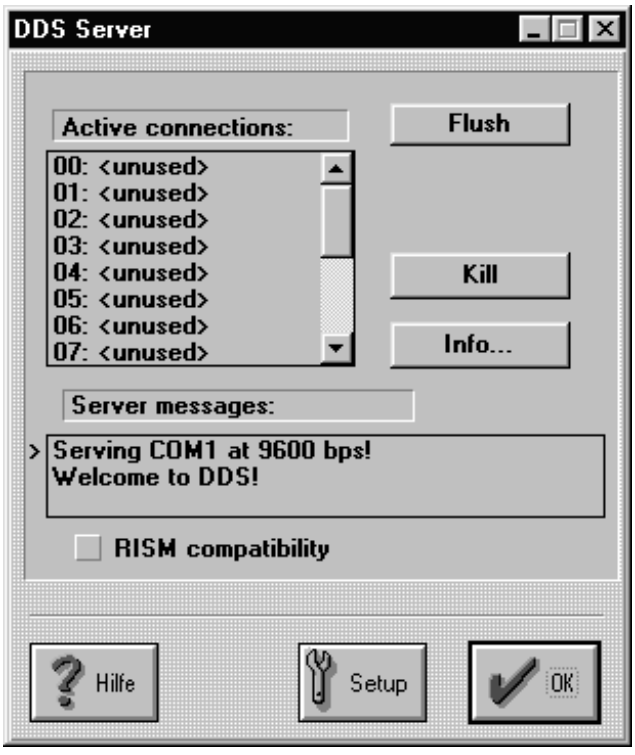


*Abbildung 2.2 - Der DDS Server*

Das Multiplexing basiert auf der Teilung des physikalischen Kanals in 16 virtuelle Ports, die unabhängig voneinander jeweils einen physikalischen Kanal vortäuschen. Jede Applikation kann nun einen Port für sich reservieren und darüber seine Daten

ungestört übertragen. Gewisse Ports sind bereits für bestimmte Zwecke reserviert, z.B. die Kommunikation mit der Systemsoftware des Didabots (Port 0) oder die C Standard I/O Funktionen eines Didabot Programmes (Port 1). Die Windowsapplikationen, die diese Ports benutzen, gehören standardmässig zum DDS. Besteht die Nachfrage nach eigenen, spezifischen Clientapplikationen, z.B. zur Visualisierung der Sensordaten, so existiert dafür eine Softwarebibliothek, die die Kommunikation mit dem DDS Server über DDE ermöglicht.

Der Didabot wird gewöhnlich in C programmiert, wofür ein Editor mit entsprechender Projektverwaltung existiert (Abbildung 2.3). Die Standard I/O Routinen wurden so angepasst, dass sie auf dem entsprechenden Port an den PC übermittelt werden. Ausserdem existieren Funktionen zur Nutzung der unbelegten Ports sowie der Steuerung des Roboters.

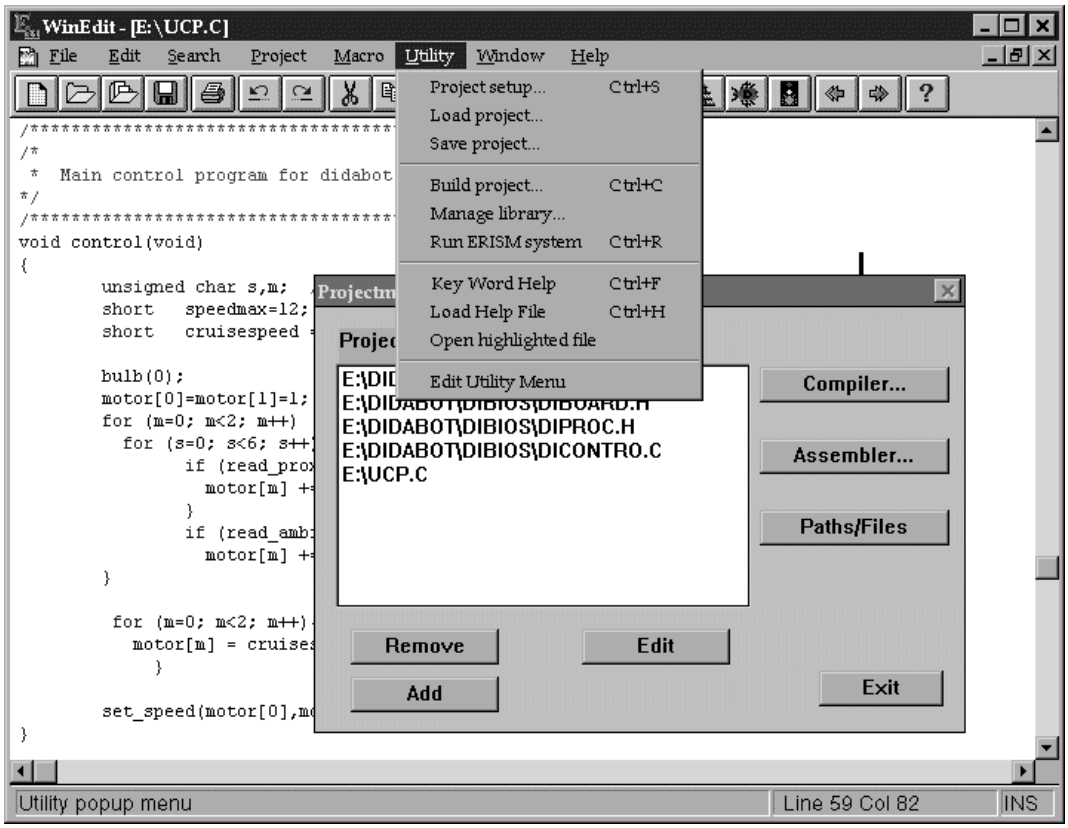


*Abbildung 2.3 - Screenshot von Editor mit Projektverwaltung*

DDS selbst ist in Borland Pascal v7.0 für Windows programmiert und liegt als Quellcode der Entwicklungsumgebung bei. DDS wurde im Rahmen einer Semesterarbeit vom Autor dieses Dokumentes entwickelt [RE95].

# 3 AUFGABENSTELLUNG

Im vorherigen Kapitel wurde die Ausgangslage für dieses beschrieben. Jetzt ist es an der Zeit, die eigentlichen Zielsetzungen zu betrachten.

Für eine adäquate Analyse des Verhaltens eines Roboters ist es notwendig, dass die Sensordaten und Motorzustände auf einem Computer visualisiert werden. Zur Erhöhung der Flexibilität ist es wünschenswert, die Daten mittels eines Radiomodems zu übertragen. Hierzu müssen Fehlererkennungsverfahren eingesetzt werden, damit ein zuverlässiger Transfer erreicht wird. Das Vorgehen sieht dabei folgendermassen aus:

- In einem ersten Teil gilt es, eine Radioverbindung zwischen Roboter (Didabot) und PC zu erstellen. Für die Kommunikation soll ein Fehlererkennungsverfahren eingesetzt werden.
- Dazu muss es möglich sein, die vom Roboter empfangenen Daten auf dem PC darzustellen. Hierzu ist eine Softwarebibliothek zu erstellen. Diese muss dergestalt sein, dass auch interne Variablen und zusätzliche interne Module eingeführt und miteinander vernetzt werden können.
- Ziel ist es, die sensomotorischen Daten des Roboters systematisch zu überwachen und zu verarbeiten. Dazu sollen die Daten optional als eine Zeitreihe gespeichert werden können.
- Das Protokoll soll es ermöglichen, dass mehrere Roboter innerhalb des Funknetzes miteinander kommunizieren können.

Die notwendigen Radiomodems wurden vor Beginn dieser Arbeit beschafft und auf den Robotern installiert. Es handelt sich dabei um das 'Low Power UHF Data Transceiver Module' [RM95] der Firma Radiometrix Ltd mit folgenden Merkmalen:

- Übertragung im halbduplex Betrieb
- Übertragungsraten bis 40 Kbit/Sek.
- Reichweite: 30m in Gebäuden, 120m im Freien.
- Die Umschaltung von Empfangen auf Senden erfolgt durch die DTR Leitung der seriellen Schnittstelle. Diese Einstellung wurde vom Entwickler der Didabots vor Beginn dieser Arbeit vorgenommen.

# 4 GRUNDLAGEN

Dieses Kapitel beschäftigt sich mit grundlegenden Überlegungen, welche Designmöglichkeiten für ein drahtloses Datenübertragungssystem bestehen. Es geht dabei nur um eine Zusammenstellung und Vergleich der verschiedenen Varianten, auf deren Basis in einem späteren Kapitel eine Designentscheidung getroffen wird. Das Vorgehen wird dahingehend sein, dass zunächst sehr allgemeine Überlegungen über Telemetriesysteme angestellt und im Laufe der Betrachtungen immer mehr die Teilbereiche wie Netzwerkarchitektur, Kanalzugriffsverfahren usw. fokussiert werden. Dabei wird kein Anspruch auf Vollständigkeit erhoben. Es werden jedoch die bekanntesten Verfahren erwähnt, die für diesen Fall angewendet werden könnten.

## 4.1 TELEMETRIESYSTEME

Telemetrie kann als eine Spezialform der Datenverarbeitung betrachtet werden. Oft werden Telemetriesysteme in der Forschung und Entwicklung angewendet, um Informationen über Orte oder Gegenstände zu erhalten, die für die Wissenschaftler selbst nur unter Gefahren oder gar nicht erreichbar sind. Ein Beispiel dafür wäre etwa die Raumsonde Voyager (Abbildung 4.1), die auf ihrem Flug aus unserem Sonnensystem neue Informationen über die äusseren, grossen Planeten geliefert hat. Dabei setzten sich die gesammelten Informationen aus verschiedenen Messgeräten zusammen. Diese wurden von der Sonde aufbereitet und an die Bodenstation auf der

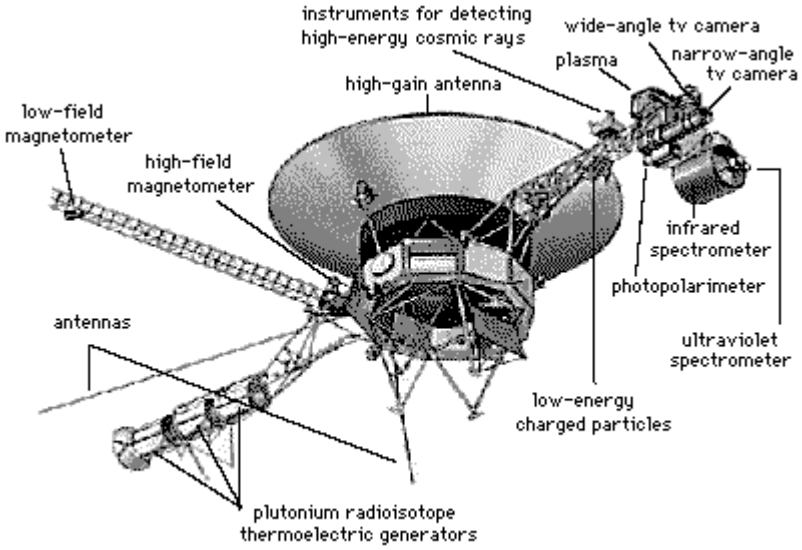


*Abbildung 4.1- Die Raumsonde Voyager*

Erde übermittelt. Erst dort wurden die gesammelten Daten ausgewertet und konnten zu neuen Erkenntnissen in der Weltraumforschung führen.

In diesem Beispiel fand der Datentransport nicht nur in eine Richtung - von der Sonde zur Bodenstation - statt. Vielmehr war es auch möglich, den Kurs der Raumsonde durch ihre 16 Düsen ferngesteuert zu ändern. Dadurch erreichte man beinahe die gleichen Möglichkeiten, wie wenn ein Mensch an Bord dieser Sonde gewesen wäre. Durch die Technik der Telemetrie konnte aber ein derart riskantes Unterfangen vermieden werden. Doch ist die Weltraumforschung bei weitem nicht das einzige Gebiet, wo Telemetriesysteme Anwendung finden. Auch bei der Erforschung des Meeres werden, vor allem für grosse Tiefen, unbemannte Unterseeboote eingesetzt, die bequem von der Oberfläche aus gesteuert werden


*Abbildung 4.2 - Das AUSS*

können. Das AUSS ('Advanced Unmanned Search System') (Abbildung 4.2) hilft zum Beispiel beim Absuchen des Meeresgrundes nach versunkenen Schiffen und Flugzeugen. Es ist dabei nicht auf eine Kabelverbindung angewiesen, sondern kommuniziert mit Hilfe von akustischen Signalen mit der Steuerzentrale. Da das AUSS [ASI92] mit Sonar, Kompass sowie einer Videokamera ausgerüstet ist, kann der Navigator das System problemlos von einem Schiff aus steuern. Dabei besitzt das System selbst ausreichend Intelligenz, um sich zu vorgegebenen Koordinaten zu lenken. Dadurch muss der Navigator das Unterseeboot nicht selbst steuern, sondern kann lediglich die gewünschten Koordinaten programmieren.

*Abbildung 4.3 - Dante II im Einsatz*

Im Falle von DANTE II (Abbildung 4.3), der zur Erforschung des Vulkans Mount Spurr eingesetzt wurde, stellen sich wiederum ganz andere Anforderungen an das Telemetriesystem. Sein Fortbewegungsapparat ist auf das unwegsame Gelände eines Berges ausgelegt und beansprucht dazu auch eine intelligente Steuerung, die die Koordination der Beine übernimmt. Mechanik sowie Elektronik muss den äusseren physikalischen Bedingungen eines Vulkans widerstehen, um eine einwandfreie Funktion zu gewährleisten. Zur Übermittlung der Messdaten wird ein Satellitenlink

benutzt, der ausreichend Kapazität besitzt, um auch Daten von 8 Kameras zu übertragen.

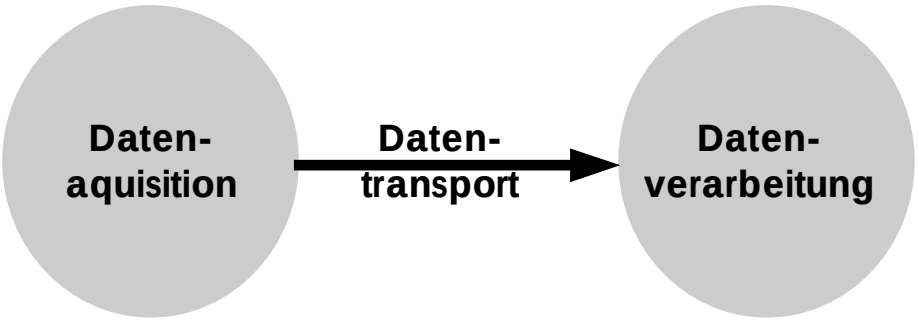


*Abbildung 4.4 - Hauptkomponenten von Telemetriesystemen*

Diese Beispiele sollen zeigen, welche Vorteile Telemetriesysteme bieten, jedoch auch welche technischen Faktoren eine wesentliche Rolle spielen. Verallgemeinert man nämlich dieses Beispiel, so können im wesentlichen zwei grundlegende Merkmale von Telemetriesystemen ausgemacht werden. Erstens, findet die Datenaquisition an einem anderen Ort statt als wo die Daten ausgewertet werden und zweitens, werden die Daten mittels eines Transportmediums übertragen (Abbildung 4.4). Wie auch in unserem Beispiel ist es möglich, dass es mehrere Datenquellen unterschiedlichster Art gibt. Diese müssen sich unter Umständen einen einzigen Kommunikationskanal teilen, wodurch ein Multiplexing System eingeführt werden muss, dass Regeln und Protokolle für den Transport verwendet. Oft ist dies auch aus rein wirtschaftlichen Gründen notwendig. Dadurch ist jedoch die Übermittlung noch nicht gewährleistet. Abhängig vom gewählten oder vorhandenen physikalischen Übertragungsmedium muss ein geeignetes technisches System für die Modulation bestimmt werden. Kommunikation unter Wasser stellt andere Anforderungen an das Modulationsverfahren als terrestrische und diese wiederum andere als interplanetare, was auch die Wahl von Sender und Antenne wesentlich beeinflusst. Auf der Gegenstelle müssen auch die entsprechenden Empfangsanlagen sowie das entsprechende De-Multiplexing System vorhanden sein. Erst jetzt ist eine umfassende Datenverarbeitung und -auswertung möglich. Wie diese auszusehen hat,

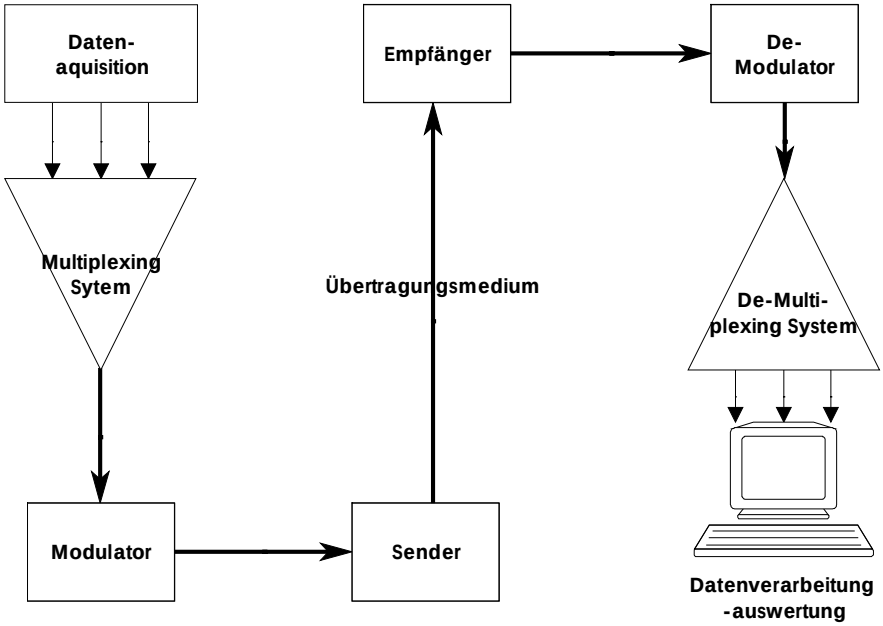


*Abbildung 4.5 - Die 9 Komponenten eines Telemetriesystems*

hängt hauptsächlich von der Art der erfassten Daten ab und kann sehr unterschiedlich sein. Zusammenfassend können also 9 grundlegende Elemente [CAR95] eines Telemetriesystemes identifiziert werden (Abbildung 4.5). Diese werden in den folgenden 6 Unterkapiteln einzeln betrachtet, wobei korrespondierende Elemente wie Sender/Empfänger und De-/Modulator als Einheit angesehen werden.

### 4.1.1 Die Datenaquisition

Die Datenaquisition ist dafür verantwortlich, dass die zu übertragenen Daten korrekt bestimmt werden. Diese Daten beruhen im allgemeinen auf von Sensoren ermittelten physikalischen Werten. Welche Grössen relevant sind, hängt dabei von der gewählten Anwendung ab. Folgendes wäre denkbar:

- **Geschwindigkeit**
- **Beschleunigung**
- **Entfernung**
- **Temperatur**
- **Helligkeit**
- **Druck** (Schalldruck, Luftdruck usw.)
- **Schallwellen** (Ton, Sprache, Frequenz, Ultraschall usw.)
- **Elektromagnetische Strahlung** (sichtbares Licht, UV, IR usw.)
- **Feuchtigkeit**
- **Radioaktive Strahlung**
- **Zeit**

Auf Methoden und Geräte zur Ermittlung obiger Messwerte soll hier nicht näher eingegangen werden [STR87] [EVR95]. Beim Design eines Telemetriesystems sollten jedoch folgende Faktoren berücksichtigt werden:

- **Betriebsanforderungen**: Es sollte ermittelt werden, unter welchen physikalischen Bedingungen ein Messgerät arbeiten muss und ob diese die Betriebsanforderungen des Gerätes erfüllen. Nur so kann eine zuverlässige Funktion gewährleistet werden.
- **Genauigkeit**: Je nach Anwendung ist eine unterschiedliche Genauigkeit erforderlich. Eine zu hohe Genauigkeit ist unwirtschaftlich, eine zu geringe kann zu Fehlverhalten des Systems führen.
- **Messfehler**: Dies kann zu falschen Interpretationen in der Datenauswertung führen. Eine Berücksichtigung von Messfehlers in der Analyse ist daher unabdingbar.
- **Grösse / Gewicht**: Abhängig vom Einsatz des Telemetriesystems müssen hohe Anforderungen an Grösse und Gewicht gestellt werden. Bei der Auswahl ist daher diesem Faktor Rechnung zu tragen. In engen räumlichen Verhältnissen darf z.B. ein Telemetriesystem eine gewisse Grösse nicht überschreiten, daher muss dies bereits beim Entwurf berücksichtigt werden.
- **Energieverbrauch**: Häufig handelt es sich bei Telemetriesystemen um autonome Geräte mit eigener Energieversorgung. Diese ist meist nicht unbegrenzt. Ein zu hoher Energiekonsum der Sensoren kann daher die Autonomiedauer markant beeinträchtigen.

- **Zuverlässigkeit**: Zu geringe Zuverlässigkeit eines Sensors kann zum Totalausfall oder sogar zum Verlust des gesamten Systems führen. Ist also während des Betriebes kein Zugriff auf das Gerät möglich, sollte eine hohe Zuverlässigkeit gewählt werden. Dies kann beispielsweise dadurch erreicht werden, dass mehrere Sensoren gleiche Aufgaben erfüllen, damit bei Ausfall einer Einheit die Daten immer noch von einem anderen Sensor gemessen werden.
- **Abtastfrequenz**: Die Abtastfrequenz sollte entsprechend der Geschwindigkeit, mit der sich die Werte verändern, gewählt werden. Nicht zu vergessen ist aber die maximale Geschwindigkeit des Übertragungsmediums. Ist die Übertragungsrate zu klein, so ist keine Echtzeitdatenverarbeitung mehr möglich. Dies ist aber für manche Anwendungen unumgänglich. Eine genauere Betrachtung dieses Dilemmas folgt im Kapitel über das Multiplexing System (Kapitel 4.1.2), das sich ebenfalls mit dieser Problematik konfrontiert sieht.

Die Sensoren werden auch nach elektrotechnischen Gesichtspunkten unterschieden. Es würde jedoch den Rahmen dieses Dokumentes sprengen, auch alle elektrotechnischen Eigenschaften zu untersuchen. Es wird daher auf das Literaturverzeichnis verwiesen, wo entsprechende Literatur aufgelistet ist [STR87].

### 4.1.2 Das De-/Multiplexing System

Das De-/Multiplexing System ist eines der Schlüsselkomponenten eines jeden Telemetriesystems. Multiplexing bedeutet die gleichzeitige Übertragung von mehreren verschiedenen Datenmeldungen über eine einzige Verbindung. Im Falle eines Telemetriesystems wären dies die verschiedenen Messergebnisse der Sensoren. Die Komplexität eines solchen Systems variiert dabei stark je nach Anwendung. Allen gemeinsam ist jedoch eines oder eine Kombination der folgenden Multiplexing Verfahren.

- **FDM** ('Frequence Division Multiplexing'): Bei FDM teilen sich die einzelnen logischen Kanäle das Frequenzspektrum unter sich auf (Abbildung 4.6). Dadurch erhält jeder Kanal einen ausschliesslich für sich bestimmten

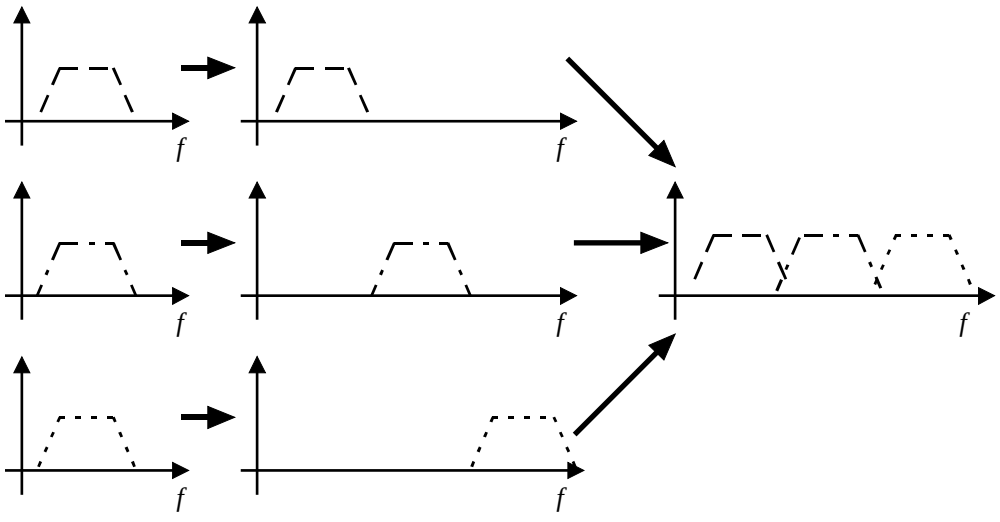


*Abbildung 4.6 - Frequence Division Multiplexing*

Frequenzbereich. Das Problem von FDM besteht darin, dass die gleichen Installationen (Modulator, Sender usw.) für jeden Kanal vorhanden sein müssen. Dies ist nur bis zu einer bestimmten Anzahl Kanäle sinnvoll. Erst bei

Verwendung von vielen Kanälen und digitaler Verarbeitung werden intelligente, programmierbare Prozessoren eingesetzt, um den Datentransfer effizienter zu gestalten oder die Flexibilität des Systems zu erhöhen. Daher trifft man FDM meist in einfacheren Systemen, dessen Verwendung auf bestimmte Funktionen begrenzt ist. Dafür bietet FDM den Vorteil, dass jeder Sensor unabhängig voneinander mit unterschiedlichen Abtastraten betrieben werden können.

- **TDM** ('Time Division Multiplexing'): Im Unterschied zum FDM teilen sich die einzelnen logischen Kanäle nicht die Frequenz, sondern die Zeit, in der sie auf einen einzigen Kanal voll zugreifen dürfen. Jeder logische Kanal bekommt in bestimmten Abständen einen Zeitslot, wo er vollen Zugriff für

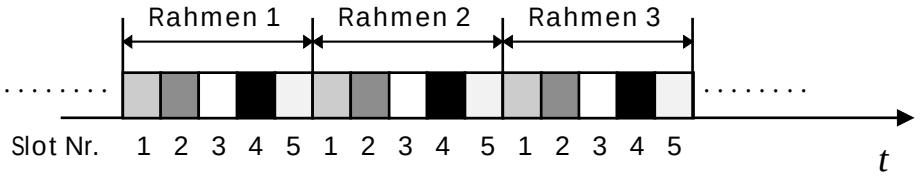


*Abbildung 4.7 - Time Division Multiplexing*

eine vorgegebene Dauer auf den Kanal bekommt (Abbildung 4.7). Die Grundaufgaben des TDM Multiplexers bestehen dabei in der Bildung der Rahmen sowie der Synchronisation. Damit der Demultiplexer den Beginn eines Rahmens erkennt, muss an dieser Stelle ein Synchronisationszeichen gesendet werden. Eine weitere Schwierigkeit tritt bei TDM auf, wenn die Sensoren unterschiedliche Abtastraten besitzen. Grundsätzlich sind zwei Situationen denkbar [STR87]:

  - Die Abtastrate eines Sensors ist höher als die der restlichen. In diesem Falle können zwei Slots innerhalb des gleichen Rahmens für diesen Sensor reserviert werden, was einer doppelten Abtastrate entspricht.

  - Die Abtastrate eines Sensors ist niedriger als die der restlichen. Dadurch würden in manchen Rahmen Lücken entstehen, da keine neuen Daten für diesen Sensor vorhanden sind. Praktisch wird dieses Problem so gelöst, dass sich mehrere, langsamere Sensoren einen Slot teilen und diesen in jedem Rahmen abwechslungsweise teilen.

  Häufig wird zur Generierung der Rahmen ein Computer verwendet. Dadurch ist eine wesentlich effizientere Übertragung möglich, da auch verfeinerte TDM Verfahren verwendet werden können, die durch intelligente Software bessere Leistungen erreichen. Solche Verfahren werden im Kapitel 4.4 genauer besprochen.

Beim Entwurf des De-/Multiplexers muss sich der Entwickler darüber Gedanken machen, mit welcher Frequenz er die einzelnen Sensorwerte abfragen will. Dies hängt mitunter davon ab, mit welcher Frequenz sich der zu messende physikalische Zustand ändert. Gemäss dem Satz von Nyquist muss die Abtastfrequenz doppelt so gross sein wie die maximale, korrekt zu erfassende Frequenz [TAN92]. Die Summe der anfallenden Daten darf im Durchschnitt die der Übertragungskapazität nicht überschreiten.

Beim Design des Multiplexers muss ein weiterer Grundsatzentscheid getroffen werden. Die Verarbeitung der Sensordaten kann auf analoger wie auch auf digitaler Basis geschehen. Im letzteren Falle müssen Analog-Digitalwandler dafür sorgen, dass ein analoges Signal eines Sensors in einen entsprechenden digitalen Wert umgewandelt wird. Dies hat ebenfalls Konsequenzen für das Multiplexing System.

Häufig wird dann ein Computer verwendet, der diese Arbeit übernimmt. Diese Lösung ist zwar wesentlich kostenintensiver, die Software bringt aber auch mehr Flexibilität.

### 4.1.3 Der De-/Modulator

Es existieren verschiedene Modulationsverfahren für die Kommunikation über unterschiedliche Übertragungsmedien. Da dem Telemetriesystem grösstmögliche Autonomie gewährt werden möchte, wird in den meisten Anwendungen eine drahtlose Übertragung bevorzugt. Dieses Kapitel beschränkt sich daher auf die Modulationsverfahren bei der Funkübermittlung. Folgende Liste nennt die bekanntesten Verfahren:

- **Amplitudenmodulation** (AM)
- **Frequenzmodulation** (FM)
- **Phasenmodulation** (PM)
- **Pulse Code Modulation** (PCM)

Diese Modulationstechniken weisen sehr unterschiedliche Leistungscharakteristiken auf (Tabelle 4.1). Wofür man sich entscheidet, hängt von den gesetzten Schwerpunkten ab.

| | AM | FM | PM | PCM |
|---|---|---|---|---|
| **Reichweite** | hoch | gering | gering | gering |
| **Frequenzbereich** | niedrig | hoch | hoch | sehr hoch |
| **Störungsanfälligkeit** | gross | klein | klein | sehr klein |

*Tabelle 4.1 - Charakteristiken von Modulationstechniken*

Der Modulator ist ausserdem dafür verantwortlich, dass die Daten entsprechend für den Transport über das physikalische Medium vorbereitet werden. In den meisten Fällen ist das Medium nicht rauschfrei, wodurch Fehler und Störungen auftreten können. Es werden daher spezielle Codierungsverfahren verwendet, um die Zuverlässigkeit der Übermittlung zu verbessern:

- **Manchester- oder Bi-Phase Codierung:** Jedes Bit wird in zwei Bits aufgeteilt. Das Erste entspricht dem eigentlichen Wert des Bits, das Zweite dem Komplementwert [CAR95]. Durch den zusätzlichen Flankenwechsel wird die Übertragungssicherheit verbessert, jedoch die Übertragungsrate halbiert. Trotzdem ist es ein häufig verwendetes Verfahren (Abbildung 4.8).

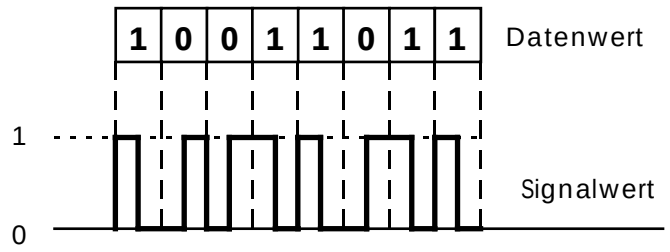


*Abbildung 4.8 - Die Bi-Phase Codierung*

- **FEC Kodierung:** Bei der Übermittlung eines Bytes wird nachfolgend das Komplement dazu transferiert. Dadurch wird gewährleistet, dass durch Flankenwechsel genügend Synchronisationsmerkmale vorhanden sind. Jedoch wird auch die Übertragungsrate halbiert [RM95].

- **Delay Modulation Code:** Eine 1 wird repräsentiert durch einen Flankenwechsel in der Mitte des Bits [CAR95]. Folgt einer Null wieder eine Null, so tritt ein Flankenwechsel am Ende der ersten Null ein (Abbildung 4.9).

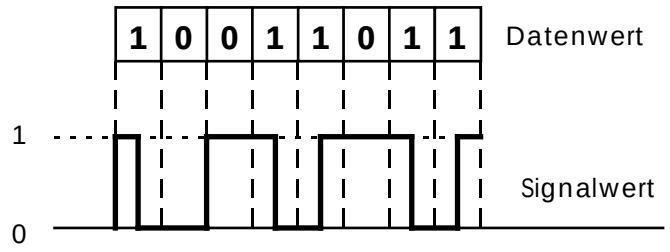


*Abbildung 4.9 - Der Delay Modulation Code*

Es existiert beinahe eine unzählbare Menge verschiedener Codierungsverfahren. Die oben erwähnten sollen nur kurze Beispiele darstellen, wie solche Methoden aussehen können. Es hängt jedoch stark auch von der verwendeten Sendetechnik ab, welche Verfahren wo anzuwenden sind.

### 4.1.4 Der Sender/Empfänger

Der Sender ist dafür verantwortlich, dass das modulierte Signal ins Übertragungsmedium eingespiesen wird, sich dort ausbreitet und dadurch den Empfänger erreicht. Antennenabstimmung sowie Sendeleistung sollten beim Design eines Telemetriesystems berücksichtigt werden. Ausserdem sollte überprüft werden, ob die geforderten Ansprüche an Reichweite und Signalqualität erreicht werden können.

### 4.1.5 Das Übertragungsmedium

Im Übertragungsmedium findet der eigentliche Transport der Daten statt. Wichtig ist dabei zu wissen, wie lange der Transport von Sender zu Empfänger dauert. Dies ist vor allem dann wichtig, wenn grosse Distanzen überwunden werden müssen, wie z.B. bei Satellitenkommunikation. Um hohe Übertragungskapazitäten sicherzustellen, müssen spezielle Multiplexingprotokolle verwendet werden. In den meisten terrestrischen Systemen ist die Transportdauer jedoch so gering, dass sie vernachlässigt werden kann.

### 4.1.6 Die Datenverarbeitung und -auswertung

Wurde das Signal korrekt empfangen und die Daten vom Demultiplexer wieder auf die einzelnen Kanäle verteilt, kann nun die Datenverarbeitung und -auswertung folgen. Die Art und Weise der Datenverarbeitung ist stark anwendungsspezifisch, so

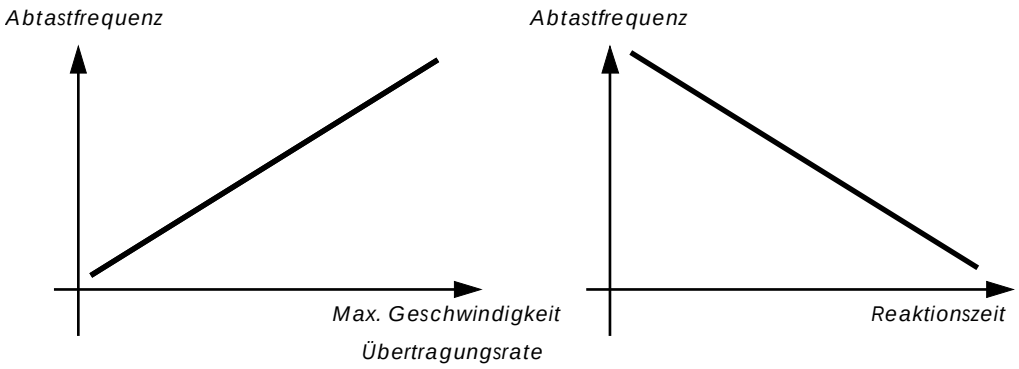


*Abbildung 4.10 - Abhängigkeiten eines Echtzeittelemetriesystems*

dass kaum Annahmen darüber gemacht werden können. Muss jedoch mit Hilfe der

Telemetriedaten zum Beispiel ein Fahrzeug gesteuert werden, so ist eine Echtzeitdatenverarbeitung vonnöten. Die Auswertung der Daten ist dadurch sehr zeitkritisch und muss so schnell wie möglich erfolgen, wodurch meist leistungsfähige Computer zum Einsatz kommen, die diese Aufgabe übernehmen. Je schneller sich das Fahrzeug bewegt, desto schneller werden sich auch die Sensorwerte ändern, was kürzere Reaktionszeiten voraussetzt. Diese verlangen wiederum höhere Abtastraten für die Sensoren sowie eine höhere Übertragungskapazität (Abbildung 4.10). Aus diesen Gründen sollte eine optimale Kombination von Abtastfrequenz, maximaler Geschwindigkeit und Übertragungskapazität erreicht werden.

## 4.2 ZUSAMMENFASSUNG

Zum Abschluss dieses Kapitels über Telemetriesysteme soll noch einmal eine Zusammenfassung helfen, eine Klassifizierung vorzunehmen. Die Bandbreite der technischen Möglichkeiten sowie die Einsatzgebiete von Telemetriesystemen sind äusserst umfangreich, so dass sich nur wenige Elemente finden lassen, die allen

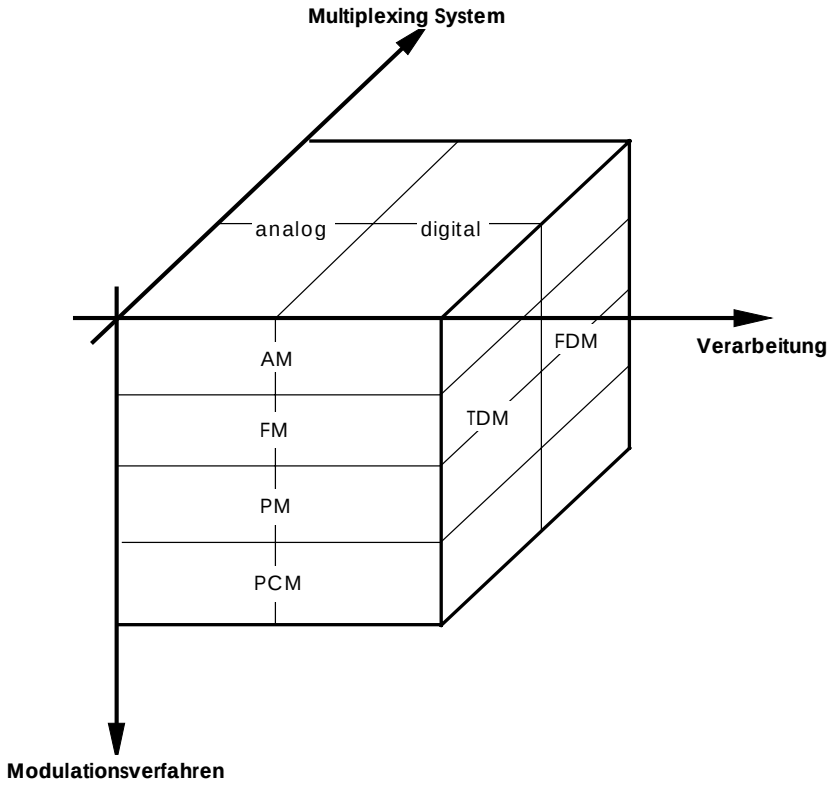


*Abbildung 4.11 - Designmöglichkeiten für ein Telemetriesystem*

Systemen gemeinsam sind und eine Klassifizierung zulassen. Damit die neun Hauptkomponenten eines Telemetriesystems zusammenarbeiten, müssen sie gemeinsame Nenner besitzen, die dies ermöglichen. Digitale oder analoge Verarbeitung hat Auswirkungen auf sämtliche Teile und kann als eine grundlegende Eigenschaft angeschaut werden. Das Multiplexing System funktioniert immer auf der Basis von FDM, TDM oder einer Kombination von beiden. Zum Schluss definiert das Modulationsverfahren die Charakteristiken für De-/Modulator, Sender und Empfänger (Abbildung 4.11). Nicht alle Kombinationen dieser drei Eigenschaften sind möglich und nur wenige wirklich sinnvoll.

Die Möglichkeiten eines digitalen Telemetriesystems führen zu einem fliessenden Übergang zu den Verfahren der Kommunikationstechnik der Informatik. Das Telemetriesystem wandelt sich dabei zu einem Teilnehmer eines Kommunikationsetzwerkes, der von einem Computer gesteuert wird. Dem

Mulitplexing System werden neue Aufgaben auferlegt wie z.B. Routing, Fehlerkorrektur oder Verschlüsselung. Dieser Übergang soll nun auch in diesem Dokument vollzogen werden. Die folgenden Kapitel sollen daher Aufschluss für die Wahl eines geeigneten Netzwerkes geben.

## 4.3 DIE NETZWERKARCHITEKTUR

Die Netzwerkarchitektur ist die Menge von Schichten und Protokollen, über die mehrere miteinander verbundene Computer kommunizieren können. Bereits ein einfaches Telemetriesystem kann als degeneriertes Netzwerk betrachtet werden, da es die Grundelemente der obigen Definition enthält. In manchen Fällen kommt es vor, dass Daten nicht nur von einem Gerät gemessen und an eine Station übermittelt werden, sondern mehrere geographisch verteilte Messstationen ein richtiges Telemetrienetz aufbauen, deren Messdaten an eine oder mehrere Datensammelstelle übermittelt werden. Im folgenden soll die Architektur solcher Netzwerke studiert werden.

Die Kommunikation innerhalb eines Netzwerkes findet über Kanäle statt. Jeder Teilnehmer kann sich einen solchen Kanal reservieren, um anschliessend seine Daten einzuspeisen. Grundsätzlich existieren zwei Kanaltypen [TAN92]:

- **Punkt-zu-Punkt Kanäle:** Das Netzwerk enthält dabei eine Vielzahl von Leitungen, die jeweils zwei Teilnehmer miteinander verbindet. Sind Sender und Empfänger nicht direkt verbunden, so müssen die Daten von anderen Netzteilnehmern bis zum Empfänger weitergeleitet werden. Diese Vermittler werden Schalteinheiten genannt. Nach welchem Prinzip diese Schalteinheiten miteinander verbunden sind, kann sehr unterschiedlich aussehen. Abbildung 4.12 zeigt mögliche Topologien für ein derartiges Netzwerk.

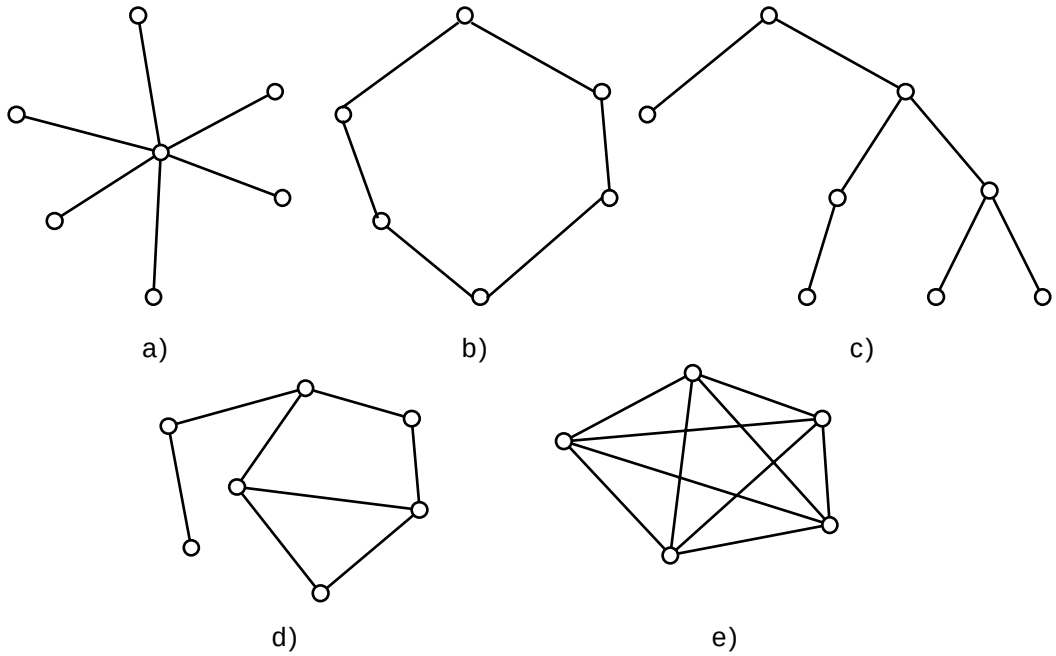


*Abbildung 4.12 - a)Stern-, b)Ring-, c)Baum-, d)heterogen oder e)komplette Struktur*

- **Rundsendekanäle:** Alle Netzteilnehmer sind am gleichen, einzigen Kanal angeschlossen. Nachrichten, die von einem beliebigen Teilnehmer abgeschickt werden, werden von allen anderen erhalten. Daher muss ein Adressfeld innerhalb der Nachricht bestimmen, welche Station der

Empfänger ist. Wenn eine Station eine Nachricht empfängt, überprüft sie das Adressfeld. Ist die Nachricht für einen anderen Empfänger bestimmt, so wird sie einfach ignoriert. Abbildung 4.13 zeigt drei mögliche Formen für Netze mit Rundsendekanälen.

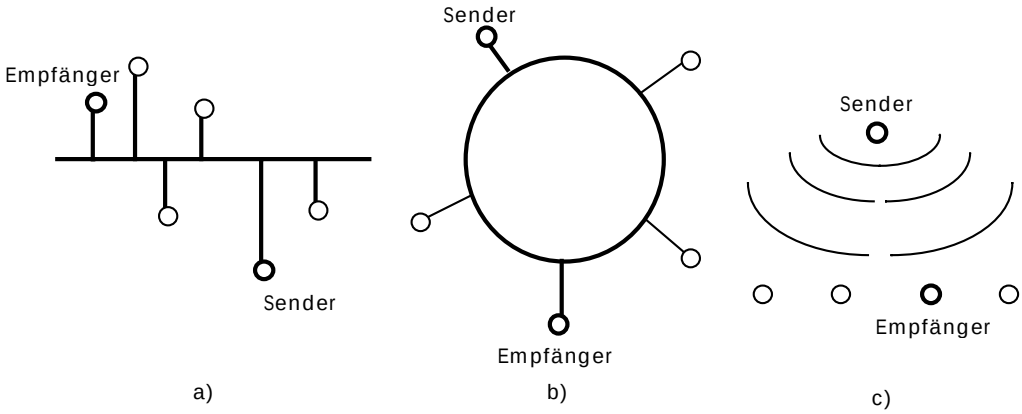


*Abbildung 4.13 - Rundsendekanäle a)Bus, b)Ring, c)Satellit oder Funk*

Diese zwei Kanaltypen sind natürlich nicht strikte voneinander getrennt. Es gibt Varianten, die Eigenschaften aus beiden Klassen mischen. Die weiteren Betrachtungen sollen sich nun darauf beschränken, welche spezifischen Probleme bei der Netzwerkarchitektur im Gebrauch mit Telemetriesystemen gelöst werden müssen. Da nicht alle Arten von Netzwerkarchitekturen diskutiert werden können, wird eine Annahme für die Telemetriesysteme getroffen, die im folgenden studiert werden sollen. Häufig ist es nämlich notwendig, diese so mobil wie möglich zu halten, womit praktisch nur Kommunikation über Funk in Frage kommt. Dadurch wird auch präjudiziert, dass vornehmlich das Prinzip der Rundsendekanäle zur Anwendung gelangt. Jedoch ist damit die Frage noch offen, nach welchem Präferenzvergabeverfahren die einzelnen Stationen Zugriff auf den Funkkanal bekommen sollen.

## 4.4 KANALZUGRIFFSVERFAHREN

Leistung und Effizienz eines Netzwerkes hängen im wesentlichen vom verwendeten Kanalzugriffsverfahren ab. Da wir uns hier auf Netzwerke mit Rundsendekanälen beschränken, entstehen beim Kanalzugriff zusätzliche Probleme. Für alle Teilnehmer eines Netzes existiert nur ein einziger Kanal, der immer nur von einem Teilnehmer zu einer bestimmten Zeit reserviert bzw. benutzt werden kann. Die Verfahren, die diesen Zugriff regeln, nennt man Mehrfachzugriffsprotokolle. Diese können grob in zwei Klassen eingeteilt werden:

- **Zentral gesteuerte Zugriffsregelung:** Der Zugriff auf den Kanal wird für alle Netzteilnehmer von einer zentralen Instanz, meist von einem bestimmten Teilnehmer, geregelt. Dadurch wird auch ein bestimmter Anteil der Bandbreite für die Regelung der Zuteilung des Kanals benötigt, was den effektiven Datendurchsatz etwas vermindert. Die Idee besteht jedoch darin, durch die Vermeidung von Kollisionen bei gleichzeitigem Senden mehrerer Stationen insgesamt mehr Bandbreite für Daten verwenden zu können.
- **Dezentral gesteuerte Zugriffsregelung:** Der Zugriff auf den Kanal folgt bestimmten Regeln, nach denen sich alle Stationen richten. Damit erreicht man, dass keine Bandbreite für die Regelung des Kanalzugriffs verwendet werden muss, nimmt dafür aber gewisse Ineffizienzen in der Übertragung in Kauf.

Welcher dieser zwei Grundtypen im Einzelfall zur Anwendung gelangt, hängt hauptsächlich von der erwarteten Auslastung sowie von der Richtung des Datenflusses ab. Meist sind auch wirtschaftliche Faktoren ausschlaggebend, da so nur in der zentralen Leitstelle komplexe Hardware verwendet werden muss. Die Gründe für oder gegen eine zentrale Steuerung sind sehr weit gefächert und können hier nicht allumfassend behandelt werden. Im folgenden beschränkt sich die Diskussion daher auf dezentral gesteuerte Mehrfachzugriffsprotokolle. Vieles kann aber auch für zentral gesteuerte Protokolle angewendet werden.

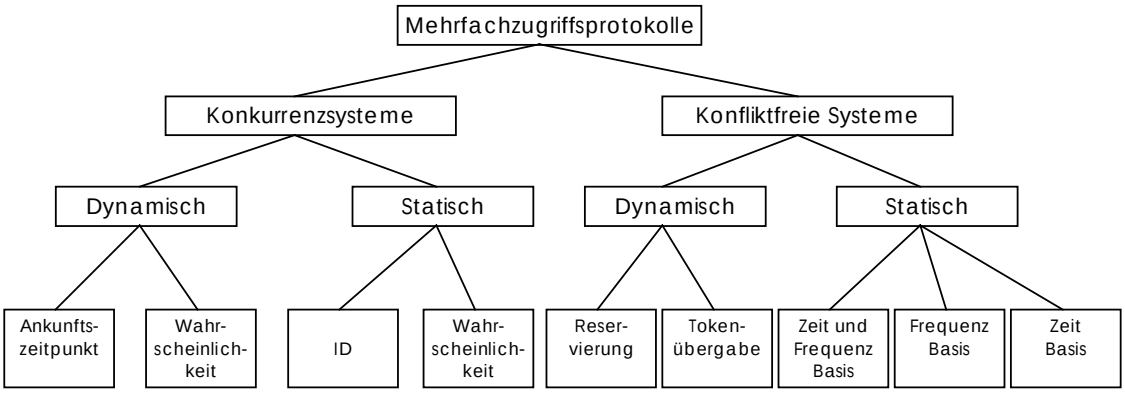


*Abbildung 4.14 - Klassifizierung der Mehrfachzugriffsprotokolle*

Auf der obersten Stufe werden Mehrfachzugriffsprotokolle unterschieden in Konkurrenz- und konfliktfreie Systeme (Abbildung 4.14) [RS90]. Konfliktfreie Protokolle stellen sicher, dass eine Übermittlung immer erfolgreich durchgeführt wird, egal zu welchem Zeitpunkt sie gestartet wird. Eine Störung durch andere Stationen wird verhindert. Dies wird erreicht durch eine statische oder dynamische Zuordnung des Kanals an eine Station. Zu diesem Zweck wird der Kanal entweder in der Zeit (TDMA - 'Time Division Multiple Access') oder in der Frequenz (FDMA - 'Frequency Division Multiple Access') oder einer Kombination (CDMA - 'Code Division Multiple Access') von beiden geteilt. Das Ziel der dynamischen Kanalzuordnung besteht darin, dass versucht wird, die sonst verschwendeten Kanalresourcen einer nicht sendenden Station anderen zur Verfügung zu stellen. Dies kann durch verschiedene Reservierungsverfahren geschehen, z.B. dort wo eine Station zuerst eine Sendeabsicht ankündigen muss. Alle diese Stationen dürfen in einer bestimmten Reihenfolge senden, bevor neue Stationen ein Senderecht erhalten können. Weiter sind Verfahren denkbar, wo ein sogenanntes Token von Station zu Station weitergegeben wird, das derjenigen das Senderecht zuspricht, die das Token gerade besitzt (MSAP - 'Mini Slotted Alternating Priority', BRAM - 'Broadcast Recognition Access Method').

Konfliktfreie unterscheiden sich von Konkurrenzsystemen darin, dass der Erfolg einer Übertragung bei letzteren nicht garantiert werden kann. Das System muss Wege definieren, die bei Konflikten bzw. Kollisionen allen Parteien ermöglichen, ihre Daten erfolgreich zu übertragen. Dieser sogenannte Auflösungsprozess benötigt natürlich Kanalresourcen. Die Idee besteht nun darin, dass die benötigte Bandbreite zur Auflösung von Konflikten kleiner ist als diejenige, welche bei konfliktfreien Systemen zur Vermeidung von Kollisionen verloren geht. Die Häufigkeit von Kollisionen hängt im wesentlichen von der Anzahl Teilnehmer im Netz sowie der Intensität des Datenverkehrs ab.

Immer wenn Konkurrenzsysteme verwendet werden, besteht die Notwendigkeit eines Verfahrens, das Konfliktsituationen auflösen kann. Wie bei konfliktfreien Systemen existieren statische wie auch dynamische Auflösungsverfahren. Statische Auflösung bedeutet, dass das Systemverhalten nicht situativ beeinflusst wird, sondern immer gleich abläuft. Diese Auflösung kann basieren auf der Stations-ID oder einer anderen fixen Prioritätsanordnung, die bestimmt, wer bei Konflikten höhere Priorität

geniesst. Statische Auflösung kann auch mit Hilfe der Wahrscheinlichkeit erreicht werden. Die betroffenen Stationen lösen den Konflikt mit Hilfe eines fixen Zufallsprinzips unabhängig von der Anzahl der betroffenen Stationen (ALOHA, CSMA - 'Carrier Sense Multiple Access').

Dynamische Auflösung kann erfolgen aufgrund der Ankunftszeit, wo die älteste Nachricht die höchste oder niedrigste Priorität erhält. Weiter kann die Auflösung ebenfalls auf Wahrscheinlichkeit beruhen, mit dem Unterschied, dass die statistischen Erwartungswerte dynamisch dem System angepasst werden.

Eine umfassende Diskussion aller dieser verschiedenen Verfahren kann an dieser Stelle nicht erfolgen und wird daher auf die beiden bekanntesten Vertreter aus dieser Gruppe beschränkt.

#### 4.4.1.1 ALOHA

ALOHA wurde um 1970 als erstes Mehrfachzugriffsprotokoll entwickelt. Wegen seines schon beachtlichen Alters ist es gleichzeitig auch die grösste Familie von Protokollen. Die Stärken von ALOHA bestehen vor allem in seiner Einfachheit und wurde ursprünglich für den bodengestützten Rundfunk entwickelt, findet heute aber auch in speziellen Varianten in lokalen Netzwerken Anwendung.

Die Hauptidee von ALOHA besteht darin, dass eigentlich gar kein Kanalzugriffsverfahren definiert wurde. Jede Station kann senden, wann immer sie das für nötig hält. Dadurch treten selbstverständlich Kollisionen auf, wodurch die kollidierenden Pakete zerstört werden. Da aber empfangene Pakete immer rückbestätigt werden, kann der Sender somit herausfinden, ob seine Meldung beim Empfänger angekommen ist. Trat also eine Kollision auf, so wird das Paket nach einer bestimmten Zeit wiederholt. Da alle von der Kollision betroffenen Stationen dies tun, würde bei einer fix festgelegten Wartezeit wieder eine Kollision auftreten. Daher wartet jede Station eine zufällig gewählte Zeitspanne, womit sich die Wahrscheinlichkeit einer erneuten Kollision verringert. ALOHA ist somit ein Mitglied der Konkurrenzsysteme.

Die zwei bekanntesten Mitglieder der ALOHA Familie sind [TAN92][RS90]:

- **Reines ALOHA** ('pure ALOHA'): Dies entspricht dem oben beschriebenen, ursprünglichen ALOHA Protokoll.
- **Unterteiltes ALOHA** ('slotted ALOHA'): Der Unterschied zur ursprünglichen Variante besteht darin, dass hier Stationen nur zu bestimmten Zeitpunkten, die in einem festen Intervall $T$ auftreten, zu senden beginnen dürfen. Die Sendedauer für ein Paket beträgt ebenfalls $T$. Eine Übertragung ist folglich immer erfolgreich, wenn genau eine Station zu Beginn eines solchen Rahmens zu senden beginnt.

Die Schwäche der ALOHA Protokolle beruht auf der Tatsache, dass die einzelnen Stationen einem sehr simplen Zugriffsverfahren folgen, was zu chaotischen Zuständen auf dem Kanal führt, entsprechend viele Kollisionen verursacht und Effizienzverluste zur Folge hat. Ausserdem ist die Stabilität von ALOHA nur bedingt gewährleistet. Gemeinhin wird nämlich von der Annahme ausgegangen, dass die Zahl der das System verlassenden (korrekt übertragenen) Pakete grösser oder gleich der Zahl der ins System eintretenden (noch zu übertragenden) Pakete ist. Dies kann aber ab einem bestimmten Verkehrsaufkommen nicht mehr gewährleistet werden, wodurch das System instabil wird und unweigerlich zusammenbricht.

Interessant ist eine genauere Leistungsanalyse der ALOHA Protokolle. Nehmen wir an, in einem Netzwerk mit einem reinen ALOHA Protokoll existiert eine unendliche Anzahl Teilnehmer, die pro Zeiteinheit $\lambda$ neue Pakete produzieren. Da manche Pakete durch Kollisionen nicht übertragen werden, ist das effektive Verkehrsaufkommen von

$g$ Paketen grösser als $\lambda$. Jedes dieser Pakete benötigt ausserdem $T$ Zeit, um übertragen zu werden, unabhängig davon, ob erfolgreich oder nicht. Als Näherung geht man nun davon aus, dass der Zeitpunkt des Übertragungsbeginns einer Poissonverteilung folgt. Damit ein Paket erfolgreich übertragen werden kann, darf innerhalb der Zeitspanne $2T$, in dessen Mitte dieses eine Paket übertragen wird, kein anderes Paket zu senden beginnen. Die Wahrscheinlichkeit also, mit der ein Paket erfolgreich übertragen wird, ist

$$P_{suc} = \frac{e^{-2Tg} \cdot (2Tg)^0}{0!} = e^{-2Tg}$$

Von diesen $g$ Paketen werden somit nur $P_{suc}g$ erfolgreich übertragen. Gemäss Definition ist Datendurchsatz der Bruchteil der Zeit, in der sinnvolle bzw. nützliche Information übertragen wird. Da ein Paket in unserem Falle die Zeit $T$ benötigt, um übertragen zu werden ergibt sich

$$S = gT \cdot e^{-2Tg}$$

als Datendurchsatzrate für das reine ALOHA Protokoll. Gebräuchlich ist die Definition von $G$, des normalisierten Verkehrsaufkommens, das aus dem Produkt von $g$ und $T$ hervorgeht. Die Endformel [RS90] lautet damit

$$S = G \cdot e^{-2G}$$

Die Formel für die Datendurchsatzrate des unterteilten ALOHA ist beinahe identisch mit obiger. Der Unterschied besteht jedoch darin, dass nur zu Beginn eines jeden Slots ein Paket gesendet werden darf. Dadurch verringert sich die Zeitspanne, in der ein Paket durch ein anderes zerstört werden kann von $2T$ auf $T$. Mit anderen Worten heisst das, wenn zu Beginn eines Slots nur eine Station zu senden beginnt, so wird die Übertragung erfolgreich sein. Die Formel ändert sich dadurch nun folgendermassen [RS90]:

$$S = G \cdot e^{-G}$$

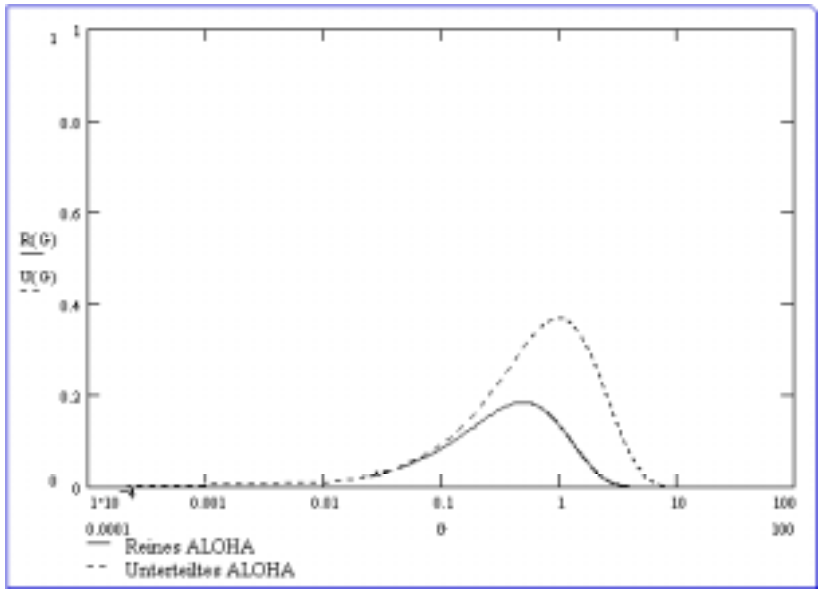


*Abbildung 4.15 - Leistungsanalyse der ALOHA Protokolle*

Nun ist leicht einzusehen, dass das unterteilte ALOHA in Sachen Durchsatz dem reinen ALOHA überlegen ist. Beim reinen ALOHA liegt der maximale Durchsatz bei $G$=0.5 ($S$≈0.18), beim unterteilten ALOHA bei $G$=1 ($S$≈0.36) (Abbildung 4.15).

#### 4.4.1.2 CSMA

Die eher mässige Leistung von ALOHA führte zur Entwicklung von CSMA ('Carrier Sense Multiple Access'). Die Leistungssteigerung wird dadurch erzielt, indem sendewillige Stationen erst überprüfen, ob bereits eine andere Station am senden ist und warten, bis diese ihre Übertragung beendet hat. Das Problem besteht nun darin, dass unter Umständen mehrere Stationen das Ende dieser Übertragung abgewartet haben und nun Regeln vorhanden sein müssen, auf welche Art und Weise die einzelnen Stationen ihre Meldungen absetzen dürfen. Verschiedene Varianten von CSMA haben sich entwickelt, die sich dadurch unterscheiden, wie in solchen Situationen zu verfahren ist [TAN92]:

- **unterbrochenes CSMA** ('nonpersistent CSMA'): Stellt eine Station bei der Prüfung des Kanals fest, dass dieser frei ist, so beginnt diese zu senden. Ist hingegen der Kanal besetzt, wartet die Station eine zufällige Zeitspanne und beginnt dann wieder von vorne mit der Prüfung des Kanals.
- **1-ständiges CSMA** ('1-persistent CSMA'): Eine Station sendet mit der Wahrscheinlichkeit von 1, sobald der Kanal frei ist. Tritt dabei eine Kollision auf, so wartet die Station eine zufällige Zeitspanne und fängt dann nochmal von vorn an.
- **p-ständiges CSMA** ('p-persistent CSMA'): Dieses Protokoll arbeitet ähnlich dem unterteiltem ALOHA mit einem getakteten Kanalzugriff. Wird zu Beginn eines Rahmens ein freier Kanal festgestellt, so wird mit einer Wahrscheinlichkeit von p gesendet. Mit der Wahrscheinlichkeit von 1-p wartet die Station auf den nächsten Slot, wo sich der ganze Vorgang mit gleichen Wahrscheinlichkeiten wiederholt.

Die Leistungsfähigkeit von CSMA hängt stark von der Zeitdauer $a$ ab, bis das erste Zeichen eines Pakets den Empfänger erreicht. In dieser kurzen Zeitspanne glauben andere Stationen noch, einen freien Kanal zu haben und beginnen möglicherweise ebenfalls zu senden, wodurch trotz der Kanalprüfung Kollisionen entstehen können. Es handelt sich daher bei CSMA um ein Protokoll der Konkurrenzsysteme. Selbst wenn die Zeitspanne Null wäre, könnte es sein, dass zwei Stationen exakt gleichzeitig zu senden beginnen. Damit ist aber der Kanal für die Dauer dieser Pakete hinsichtlich der Effizienz ungenutzt. Daher wurde CSMA so erweitert, dass Kollisionen schon zum Zeitpunkt des Sendens erkannt werden können (CSMA/CD - 'Carrier Sense Multiple Access/Collision Detection'). Dies geschieht, indem eine Station während des Sendens gleichzeitig auch empfängt und prüfen kann, ob die eigenen Daten wieder korrekt empfangen werden oder ob eine andere Station Störungen verursacht.

CSMA/CD gilt als eines der leistungsfähigsten Protokolle der Familie der Konkurrenzsysteme. Es ist heutzutage weit verbreitet und findet in vielen Netzwerken Anwendung.

Um einen effektiven Vergleich mit den ALOHA Protokollen vornehmen zu können, wird auch hier eine Leistungsanalyse für die drei oben erwähnten Fälle von CSMA vorgenommen, jedoch ohne eine Herleitung der Formeln zu geben. Diese ist zwar ähnlich, aber wesentlich komplexer als bei ALOHA. Weiterführende Informationen sind im Literaturverzeichnis angegeben [ABR93]. Der Durchsatz hängt bei allen CSMA Protokollen wesentlich von der Zeit $a$ ab, die ein Paket benötigt, um alle Stationen des Netzes zu erreichen. Bei kurzen Verzögerungszeiten ist CSMA wesentlich leistungsfähiger als ALOHA (Abbildung 4.16 und Abbildung 4.17). Da aber der Durchsatz bei ALOHA nicht abhängig ist von $a$, ist es bei grossen Verzögerungszeiten dem CSMA Protokoll überlegen. Beim Vergleich der CSMA Protokolle untereinander schneidet das p-ständige CSMA bezüglich Datendurchsatz

am besten ab. Unter Umständen sind jedoch noch andere Faktoren entscheidend, die für das eine oder andere Protokoll sprechen, z. B:

- Homogenität der Verteilung der Datenvolumen auf die einzelnen Stationen
- Minimierung der Verzögerungszeiten
- Fehlerrate des Kanals

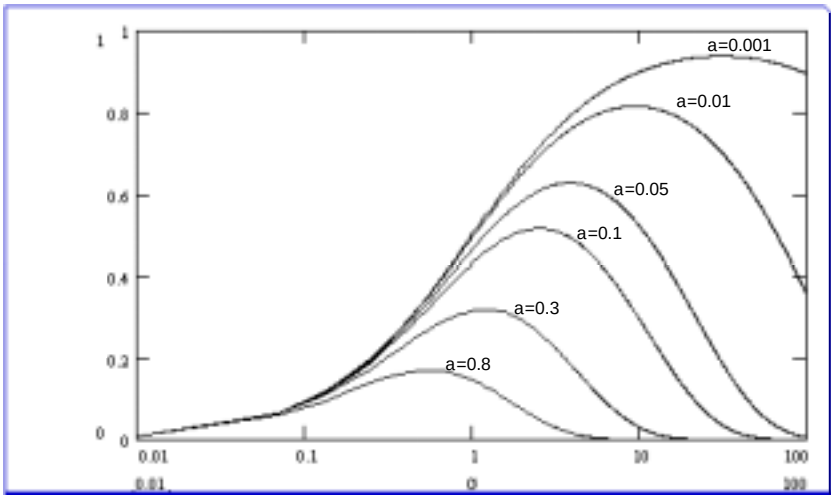


*Abbildung 4.16 - Leistungsanalyse unterbrochenes CSMA*

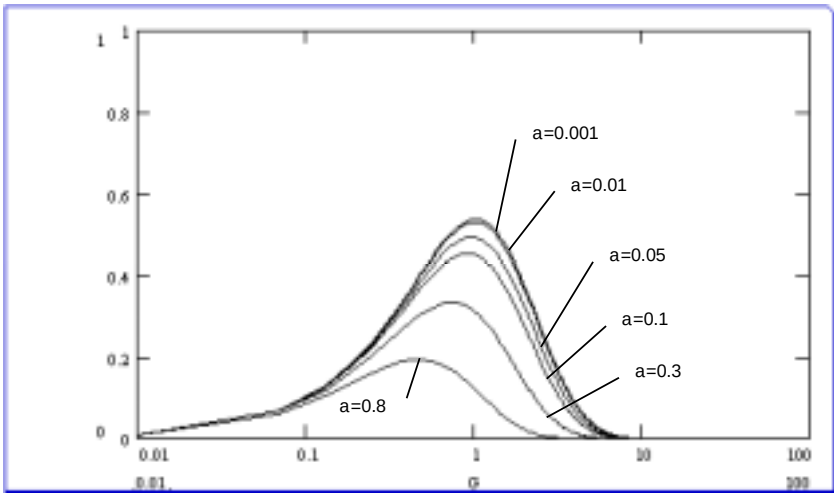


*Abbildung 4.17 - Leistungsanalyse 1-ständiges CSMA*

## 4.5 FEHLERERKENNUNGSVERFAHREN

In jeder Form der elektrischen Datenübertragung muss das Problem des Kanalrauschens berücksichtigt werden. Rauschen ist ein ungewolltes, zufälliges Signal und kann verschiedenen Ursprungs sein. Dies führt bei digitaler Datenübertragung nun dazu, dass auf dem physikalischen Medium einzelne Bits durch das Rauschen gestört bzw. verändert werden. Trotz der gewöhnlich kurzen Dauer einer solchen Störung reicht bereits auch schon ein einziges manipuliertes Bit aus, um die Bedeutung einer Nachricht vollkommen zu verändern [MCN88]. Ein Netzwerkprotokoll muss jedoch Mittel und Wege finden, um Fehlerfreiheit garantieren zu können. Da es aber keinen physikalischen, rauschfreien Kanal gibt, müssen Verfahren gefunden werden, die eine Erkennung, eventuell sogar eine Korrektur von Fehlern ermöglichen. Fehlerkorrekturverfahren gehen also gegenüber von Fehlererkennungsverfahren im Ziel noch wesentlich weiter. Sie sollen es dem Empfänger ermöglichen, eine teilweise gestörte Nachricht mit obengenanntem Verfahren wieder zu korrigieren. Auf den ersten Blick mag Fehlerkorrektur- dem Fehlererkennungsverfahren überlegen sein. Es sind jedoch zusätzliche Überlegungen nötig, um zu entscheiden, welches Verfahren in einem Netzwerk zur Anwendung kommen soll. Beide Verfahren fügen der zu übertragenen Information redundante Daten bei, damit der Empfänger mit dessen Hilfe mögliche Fehler erkennen oder eben gleich korrigieren kann. Diese zusätzlichen Daten benötigen natürlich eine gewisse Bandbreite, wobei leicht einzusehen ist, dass zur Korrektur mehr redundante Information nötig ist als zur blossen Erkennung von Fehlern [ML85]. Welches Verfahren nun gewählt werden soll, hängt damit davon ab, welche Arten von Fehlern auftreten und welcher Anteil eines Informationspaketes davon betroffen wäre. Die Antworten darauf werden durch bestimmte Eigenschaften eines Netzwerkes beeinflusst:

- **Fehlerhäufigkeit**: Wie häufig tritt ein Fehler bei der Übertragung eines Informationspaketes auf? Normalerweise bewegen sich diese Werte im Bereich von einem fehlerhaften auf 100'000 übertragene Bits. Je höher die Fehlerrate ist, desto eher sollte ein Fehlerkorrekturverfahren erwogen werden. Da in den meisten Fällen eine Wiederholung eines Informationspaketes mehr Bandbreite benötigt als die zusätzlichen redundanten Daten, kann sich eine Korrektur als lohnenswert herausstellen.

- **Störungsdauer**: Je länger eine einzelne Störung dauert, desto eher wird sich eine Wiederholung des Paketes lohnen, da auch Daten für die Fehlerkorrektur betroffen sein könnten, was eine Rekonstruktion der ursprünglichen Daten verunmöglicht. In diesem Sinn können auch Kollisionen von Paketen verschiedener Stationen als Störungen angesehen werden, vor allem wenn das Netzwerk ein Mehrfachzugriffsverfahren verwendet. Derartige Störungen dauern lange im Vergleich zur Übertragungszeit eines einzelnen Bits, nämlich die ganze Sendedauer eines Paketes.

Es sind jedoch auch andere Gründe denkbar, wieso das eine oder andere Verfahren bevorzugt verwendet werden soll. Zum Beispiel ist es denkbar, dass zeitkritische Informationen übermittelt werden, die ihren Wert bei einer Wiederholung wegen eines aufgetretenen Fehlers verlieren, da bereits zuviel Zeit verstrichen ist. Sinnvollerweise sollten dann die Daten verworfen und wieder die zeitlich aktuellsten Werte übermittelt werden. Damit trotzdem die Chance besteht, die fehlerhaften Daten zumindest teilweise auszuwerten, ist in dieser Situation eine Fehlerkorrektur sinnvoll. Manchmal kommt es auch vor, dass der exakte Wert nicht unbedingt benötigt wird und durch eine Interpolation korrekter Daten ausreichend angenähert werden kann.

Ein Netzwerk mit Mehrfachzugriffsprotokoll muss vor allem das Problem der möglicherweise auftretenden Kollisionen mehrerer Pakete lösen. Da Kollisionen im Grunde den gleichen Effekt haben wie eine Störung anderer Ursache, muss sie

innerhalb des Protokolles auch als solche behandelt werden. Dies bedeutet, dass eine Störung extrem lang sein kann (um genau zu sein, die Übertragungsdauer eines Paketes), weshalb Fehlerkorrekturverfahren kaum sinnvoll sind. Aus diesem Grund werden im folgenden nur noch Fehlererkennungsverfahren behandelt. Für Fehlerkorrekturverfahren wird auf das Literaturverzeichnis verwiesen [VO89] [ML85].

Die einfachste und bekannteste Variante eines Fehlererkennungsverfahren ist das Paritätsbit [MCN88]. Dabei wird jeweils am Ende einer bestimmten Anzahl Datenbits ein zusätzliches Bit übemittelt, das die Anzahl der gesetzten Bits nach Definition entweder gerade oder ungerade macht. Ist also eine gerade Zahl von Datenbits gesetzt und ungerade Parität verlangt, so muss das Paritätsbit gesetzt werden. Die Schwäche der Paritätsprüfung besteht nun darin, dass die Fehlererkennung zwar bei einem einzigen Bitfehler funktioniert, jedoch schon bei zwei geänderten Bits versagt, da die Anzahl gesetzter bzw. gelöschter Bits gleich bleibt. Ein solcher Fehler würde also übersehen werden. Daher wurden weitere Verfahren entwickelt, die bessere Wahrscheinlichkeiten der Erkennung eines Fehlers bieten. In der Praxis hat sich besonders ein Verfahren als äusserst effizient erwiesen. Es soll im folgenden als Beispiel vorgestellt werden.

CRC ('Cyclic Redundancy Check') [MCN88] ist eines der am häufigsten verwendeten Verfahren in der Kommunikation, da es einen guten Kompromiss an Leistung bei minimalem Aufwand bietet. Eine CRC wird mit Hilfe zweier Polynome $G(x)$ und $P(x)$ berechnet. Diese Polynome werden in binärer Form repräsentiert und zwar so, dass jedes Bit einem Koeffizienten entspricht, zum Beispiel stellt also '1100110001' das Polynom $x^9+x^8+x^5+x^4+1$ dar. $G(X)$ nennt man das Datenpolynom und wird mit einer bestimmten Anzahl $k$ Datenbits gefüllt. $P(X)$ ist das sogenannte Generatorpolynom. Es wird einem beliebigen, vordefinierten Initalwert zugewiesen. Wie später ersichtlich sein wird, muss dieser aus $m+1$ Bits bestehen, damit eine CRC von $m$ Bits berechnet werden kann. Im weiteren Verlauf wird nun $G(x)$ mit $x^m$ multipliziert und durch $P(x)$ dividiert. Der daraus entstehende Quotient kann ignoriert werden - nur der Rest ist relevant, denn dieser stellt die eigentliche CRC dar. Da durch $P(x)$ geteilt wurde, das ja aus $m+1$ Bits besteht, kann also der Rest maximal $m$ Bits besitzen.

Die berechnete CRC wird zusammen mit dem Datenwert an den Empfänger übermittelt. Dieser führt den gleichen Vorgang ebenfalls durch. Erhält er den gleichen Rest wie die übermittelte CRC, so sind die Daten korrekt übertragen worden. Das folgende Beispiel soll die CRC-Berechnung verdeutlichen.

| | | | |
|---|---|---|---|
| Datenpolynom: | $G(x)=110011$ | $(x^5+x^4+x+1)$, | $k=6$ |
| Generatorpolynom: | $P(x)=11001$ | $(x^4+x^3+1)$, | $m=4$ |

$$\frac{G(x)\cdot x^m}{P(x)} = \frac{x^4(x^5+x^4+x+1)}{x^4+x^3+1} = \frac{x^9+x^8+x^5+x^4}{x^4+x^3+1}, \quad \left(\frac{1100110000}{11001}\right)$$

```
1100110000:11001
11001
0000010000
     11001
```
**CRC=01001**

Die Berechnung einer CRC ist vor allem in Hardware äusserst einfach zu implementierten, weswegen sie auch sehr häufig angewendet wird. Die Einfachheit kommt daher, dass obige Polynommultiplikation nicht anderes als eine Verschiebung der Bits nach links darstellt. Die Division kann ausserdem durch eine einfache Exklusiv-Oder Funktion durchgeführt werden.

Der genaue mathematische Beweis kann in angegebener Literatur detailliert nachgelesen werden [MCN88] [PFT86].

# 5 SYSTEMDESIGN

Dieses Projekt hat die Aufgabe, an einem bestehenden System Erweiterungen vorzunehmen. Diese beschränken sich jedoch auf die Softwareebene, wenn man die Installation der Radiomodems mal ausser acht lässt. Es gilt nun, ein vernünftiges Mass an Wiederverwendung und Neuentwurf zu finden. Das Projekt kann grob in zwei grosse Teile zerlegt werden:

- Kommunikation über Radiomodem: Diese stellt einen Grossteil der Arbeit dar. Die Kommunikation fand zuvor über ein serielles Kabel statt, wodurch hier wahrscheinlich die meisten Kompromisse in Sachen Wiederverwendung gemacht werden müssen, um nicht gleich alles neu implementieren zu müssen. Wiederverwendung hat daher zur Folge, dass das Systemdesign nicht optimal ausfallen kann, da damalige Designentscheide aufgrund anderer Kriterien erfolgten. Diese Modifikationen beschränken sich hauptsächlich auf die Kommunikationsfunktionen des Didabot Development System (DDS) [RE95] und der DDE Interapplikationskommunikation des DDS Servers. Das Re-Design dieser Komponenten wird im anschliessenden Kapitel behandelt.
- Das Didabot Control Interface (DCI): Es handelt sich dabei um eine komplette Neuentwicklung, die ein zusätzlicher Teil des Didabot Development System (DDS) sein wird. Es verwendet daher auch DDS Dienste zur Kommunikation mit dem Didabot. Das Design von DCI wird in Kapitel 5.6 genauer betrachtet.

## 5.1 NETZWERKARCHITEKTUR

Die bisherige Netzwerkarchitektur beschränkte sich auf ein degeneriertes Zweiernetz PC - Didabot, verbunden über die serielle Schnittstelle. Die Kommunikation konnte in beiden Richtungen gleichzeitig stattfinden bei einer Übertragungsrate von 9600 Bit/Sek. Die Richtung des Datenflusses beschränkte sich auf die Übertragung von Programmen vom PC auf den Didabot. Danach wurde die serielle Verbindung getrennt und das Programm ausgeführt. Die Software erlaubte es theoretisch, Rückmeldungen während der Ausführung an den PC zu übermitteln. Da aber die Kabelverbindung die Autonomie des Roboters stark einschränkte, kam diese Möglichkeit kaum zur Anwendung.

Das Netzwerk, das nun auch neu aus mehreren Teilnehmern bestehen kann, bekommt nun neue Aufgaben. Um die Architektur optimal an die neuen Bedürfnisse anzupassen, muss man diese zuvor einer Analyse unterziehen. Diese Analyse betrifft

- **Datenvolumen:** Welche Datenmengen sind zu erwarten?
- **Dauer:** Wieviel Zeit wird das in Anspruch nehmen?
- **Richtung:** In welche Richtung werden die Daten hauptsächlich fliessen?
- **Art der Daten:** Welche Annahmen können über die Art der Daten gemacht werden?
- **Geographische Anordnung:** Wie sind die Stationen geographisch verteilt?

Jedem zu erwartenden oder verlangten Bedürfnis müssen all diese Fragen gestellt werden, deren Gesamtheit dann die Entscheidungsgrundlage bilden. Folgende alte oder neu dazugekommene Bedürfnisse existieren:

- **Programme übertragen:** Es soll möglich sein, auf dem PC geschriebene Programme auf den Didabot zu übertragen und anschliessend auszuführen.

- **Roboter steuern:** Der Roboter soll über Steuerbefehle vom PC aus komplett ferngesteuert werden können.
- **Sensordaten auslesen:** Zur Fernsteuerung des Roboters vom PC aus muss es möglich sein, die Sensordaten auszulesen.
- **Debugging Hilfen:** Übertragene Programme müssen während der Ausführung analysiert werden können.
- **Koordination der Roboter:** Der Datenaustausch zwischen den Robotern im Netz soll möglich sein zur gegenseitigen Koordination.

| | Daten-volumen | Dauer | Richtung | Art der Daten | Geogr. Anordnung |
|---|---|---|---|---|---|
| Programme übertragen | **hoch** | **kurz** | **PC-Didabot** | **beliebig binär** | **beliebig** |
| Roboter Steuern | **klein** | **lang** | **PC-Didabot** | **Steuer-befehle** | **beliebig** |
| Sensordaten auslesen | **hoch** | **lang** | **Didabot-PC** | **beliebig binär** | **beliebig** |
| Debugging Hilfen | **mittel** | **mittel** | **alle** | **beliebig binär** | **beliebig** |
| Koordination der Roboter | **mittel** | **lang** | **Didabot-Didabot** | **beliebig binär** | **beliebig** |

*Tabelle 5.1 - Bedürfnisanalyse*

Aus Tabelle 5.1 ist ersichtlich, dass die zu erwartende Netzwerkarchitektur mittleres bis hohes Datenvolumen für eine längere Zeitdauer bewältigen muss. Dabei ist die Richtung des Datenflusses nur leicht zentralisiert (PC als Zentrum). Die Netzwerktopologie kann daher nicht höher strukturiert werden, da damit keine Effizienzgewinne gemacht werden können - eher das Gegenteil würde der Fall sein. Eine heterogene Topologie mit einem Rundsendekanal für alle Teilnehmer scheint daher am sinnvollsten. Aufgrund der verwendeten Hardware ist nur die Lösung mit Rundsendekanälen möglich, da alle Radiomodems auf der gleichen Frequenz arbeiten. In der weiteren Diskussion wird noch häufiger auf die Bedürfnisanalyse verwiesen, da sie auch für die Wahl des Kanalzugriffsverfahrens sowie der Paketstruktur relevant ist.

## 5.2 KANALZUGRIFFSVERFAHREN

Das Kanalzugriffsverfahren ist das Herz eines jeden Netzwerkes. Um zu entscheiden zwischen einer zentralisierten oder dezentralisierten Zugriffsregelung, sollte die Richtung der Datenflüsse näher betrachtet werden. Eine zentralisierte Zugriffsregelung ist dann vorteilhaft, wenn der grösste Teil der Kommunikation zwischen einer normalen Station und dieser zentralen Stelle erfolgt. Weiter bedeutet dies, dass ohne diese Instanz das Netzwerk nicht funktionsfähig ist. Aufgrund dieser Tatsache sowie aus Tabelle 5.1 aufgelisteten Richtungen der Datenflüsse scheint eine dezentrale Zugriffsregelung vorteilhafter. Das Übertragen von Programmen stellt nur ein punktuelles Ereignis dar, dessen Effizienz nicht zentral im Vordergrund steht. Vielmehr zeigen die restlichen Anwendungsgebiete, dass je nach Situation unterschiedliche Richtungen Priorität geniessen, in welche die Daten fliessen.

Es stellt sich nun die Frage, welches der Mehrfachzugriffsprotokolle für diesen Fall am besten geeignet ist. Die vorhandene Hardware diktiert jedoch bereits, dass es sich dabei um eine Variante von TDM ('Time Division Multiplexing') handeln muss. Die

installierten Modems lassen kein anderes Verfahren zu. Folgende übrige Alternativen wären denkbar:

| | Vorteile | Nachteile |
|---|---|---|
| **Konfliktfreie Systeme** | | |
| reines TDM | **einfach, zuverlässig** | **Resourcenverschwendung bei ungenützen Slots.** |
| Tokenübergabe | **guter Durchsatz, einfach** | **Lange Wartezeiten bei einseitiger Netzauslastung** |
| Reservierung | **effizient, einfach, zuverlässig** | **zweiter Kanal zum Anzeigen von Reservierungen notwendig** |
| **Konkurrenzsysteme - Auflösung durch** | | |
| Wahrscheinlichkeit (dyn.) | **sehr effizient, flexibel** | **sehr komplex, Abstimmung der Variablen bei vielfältiger Netzstruktur schwer** |
| ID | **einfache Prioritätenregelung** | **evtl. ineffiziente Kollisionsauflösung, zusätzl. Bandbreite für Kollisionshandling nötig** |
| Ankunftszeitpunkt | **Berücksichtigung des Faktors Zeit als Prioritätenordnung für Echtzeitsysteme günstig.** | **Zeitpunkt nicht immer leicht feststellbar** |
| Wahrscheinlichkeit (stat.) | **effizient, flexibel** | **Effizienzverlust bei extremen Spitzen des Datenaufkommens** |

*Tabelle 5.2 - Vor- und Nachteile von Mehrfachzugriffsprotokollen*

Wie wichtig der Faktor Effizienz ist, hängt davon ab, wie gross die Reserven bei der Übertragungskapazität sind. Die Radiomodems besitzen eine maximale Rate von 40 Kbit/Sek. Da jedoch die Modems selbst kein Codierungsverfahren benützen und dieses vom Hersteller selbst zur Verbesserung der Übertragungssicherheit empfohlen wird, muss diese Rate bereits auf ca. die Hälfte reduziert werden. Weiter ist zu berücksichtigen, dass zusätzliche Bandbreite für die Regelung des Kanalzugriffs bzw. zur Auflösung von Kollisionen verwendet werden muss. In Anbetracht dessen und unter Berücksichtigung der Datenvolumen in Tabelle 5.1 sind die Reserven der Übertragungskapazität eher als gering zu beurteilen. Systeme mit grösseren Ineffizienzen wie reines TDM und ID können schon vorweg ausgeschlossen werden. Ein System mit Kanalreservierung ist nicht möglich, da die Hardware mit einem Netzkanal als gegeben angesehen werden kann. Da die Zahl der Netzstationen stark variieren kann, ist das System der Tokenübergabe wenig sinnvoll, da bei jeder Veränderung der Netzstruktur Änderungen der Netzkonfiguration vorgenommen werden müssen. Es sollte möglich sein, ohne Änderungen an den bestehenden Netzteilnehmern neue, zusätzliche Didabots ins Netz aufzunehmen, um ein angenehmes Arbeiten für den Anwender zu gewährleisten. Unter solchen sich ändernden Bedingungen ein System mit Variablen zur Optimierung des Netzes dynamisch anzupassen ist nur schwer lösbar, wobei der Effizienzgewinn je nach Situation wohl sehr unterschiedlich ausfällt.

Übrig bleiben somit noch Konkurrenzsysteme mit statischer Kollisionsauflösung durch die Stations-ID oder Wahrscheinlichkeit (Tabelle 5.2). Denkbar wären wohl beide, doch haben sich letztere in der Praxis bereits vielfach bewährt. Die bekanntesten Vertreter dieser Gruppe sind ALOHA sowie das CSMA ('Carrier Sense Multiple Access') Protokoll. Auf ihre Leistungsmerkmale und Eigenschaften wurde in einem früheren Kapitel eingegangen und soll hier nicht wiederholt werden. Bei kurzen Verzögerungszeiten, bzw. bei hoher Ausbreitungsgeschwindigkeit ist das CSMA Protokoll eindeutig überlegen.

Es existiert eine Vielzahl von Varianten des CSMA-Protokolles. Drei davon wurden in Kapitel 4.4.1.2 behandelt. Die Unterschiede in Sachen Datendurchsatz sind als eher gering zu betrachten, daher wären wohl alle 3 Kandidaten möglich. Dieses Projekt weist jedoch einige wichtige Punkte auf, die speziell für das 1-ständige CSMA sprechen. In der Praxis ist es nämlich häufig der Fall, dass nur ein Didabot zusammen mit dem PC in einem Zweiernetz miteinander kommunizieren. Dadurch ist die Wahrscheinlichkeit von Kollisionen relativ gering. Da nun das 1-ständige CSMA jedes in Auftrag bekommene Datenpaket sofort absendet, treten keine Verzögerungen im Datenfluss auf, was eine effizientere Auslastung des Netzes bei nur zwei Knoten verspricht.

## 5.3 PAKETSTRUKTUR

Bei jedem TDM Verfahren müssen die Daten eines Senders in kleinere Einheiten unterteilt, dann übertragen und am Ankunftsort wieder in der richtigen Reihenfolge in einen einzigen Datenstrom zusammengefügt werden. Die Paketstruktur muss den Bedürfnissen von Multiplexer, Modulator, Fehlererkennung und Effizienz Rechnung tragen.

- **Multiplexer:** Der Sender muss dem Paket zusätzliche Information mitgeben, damit klar ist, an wen dieses Paket übermittelt wird und von wem es abgeschickt wurde.
- **Modulator:** Unter Umständen muss ein Paket speziell durch ein Codierungsverfahren vorbereitet werden, damit es den Ansprüchen des Modulators genügt. Häufig nimmt dies jedoch der Modulator selbst vor. Ausserdem kann es sein, dass einem Paket eine sogenannte Präambel vorausgehen muss, bis die Elektronik des Modulators zur Übertragung bereit ist.
- **Fehlererkennung:** Spezielle Informationen innerhalb eines Paketes müssen sicherstellen, dass etwaige auftretende Fehler während der Übertragung als solche erkannt werden.
- **Effizienz:** Die Länge eines Paketes hat wesentlichen Einfluss auf die Effizienz eines Netzwerkes, da in der Zeit der Übertragung eines einzelnen Paketes der Kanal für alle anderen Teilnehmer gesperrt bleibt. In dieser Zeit warten immer mehr Pakete darauf, ebenfalls abgeschickt zu werden.

Die gewählte Paketstruktur (Abbildung 5.1) berücksichtigt alle diese Faktoren.

- **Präambel:** Gemäss Dokumentation der Radiomodems benötigt die Sendeeinheit 3ms, bis ein stabiles Trägersignal erreicht ist und eine zuverlässige Übermittlung gewährleistet werden kann. Start des Senders und Beginn der Paketübertragung beginnen gleichzeitig. Damit sichergestellt ist, dass das Trägersignal zum Zeitpunkt des eigentlichen Paketbeginns (beim Startzeichen) stabil ist, wird eine Präambel vorangestellt, die diese 3 ms mit dem Senden der Zahl $AA überbrückt. Wieviele solcher Füllbytes nötig sind, hängt von der gewählten Baudrate ab. Die Anzahl $n$ kann wie folgt berechnet werden:

$$n = \frac{Baudrate}{10000} \cdot 3$$

- **Initialisierungssequenz:** Nachdem ein stabiles Trägersignal erreicht ist, muss eine genau definierte Sequenz den Beginn eines Paketes anzeigen. Diese Sequenz darf nicht oder mit nur minimalster Wahrscheinlichkeit durch Rauschen erzeugt werden, damit mit grosser Sicherheit der Paketstart erkannt werden kann.

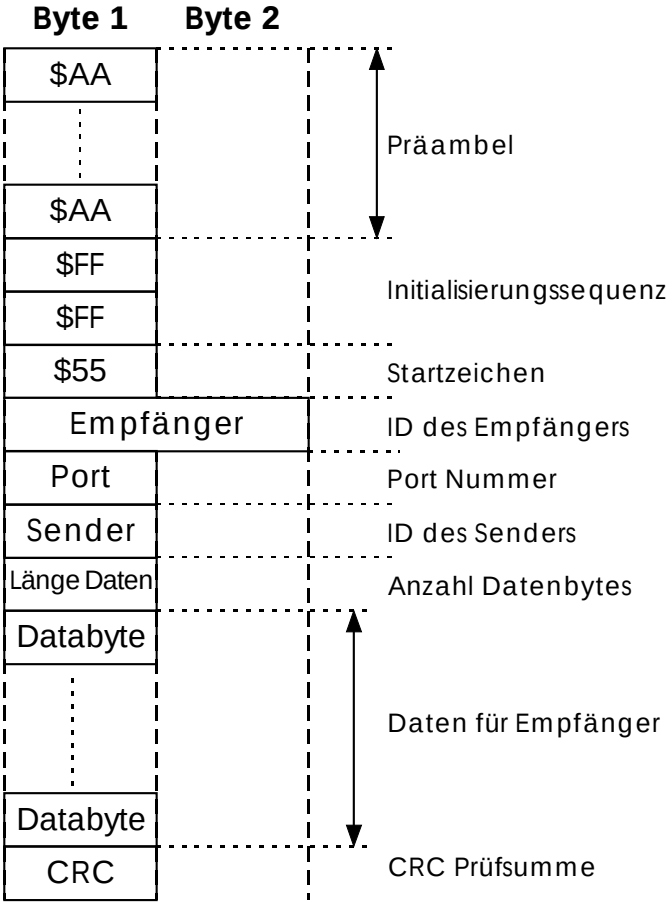


*Abbildung 5.1 - Die Paketstruktur*

- **Startzeichen:** Das Startzeichen markiert den Beginn der eigentlichen Informationen, die zu einem Paket gehören.
- **Empfänger:** Dieser 16 Bit lange Eintrag definiert, für wen das Paket bestimmt ist. Jedes Bit stellt dabei eine Station dar. Die Nummer des Bits entspricht der Stations-ID. Ist das Bit $x$ also gesetzt, so ist Station mit ID $x$ einer der Empfänger. Dadurch ist es möglich, Pakete an mehrere, beliebig kombinierbare Stationen zu senden.

- **Port Nummer:** Dieses Byte enthält unter anderem die Port Nummer. Die virtuellen Ports sind numeriert von 0 bis 15. Dadurch wird definiert, in welchen Port die Datenbytes geschrieben werden sollen. Jedes Paket besitzt zusätzlich eine Paketnummer im Bereich von 0 bis 3. Dadurch wird

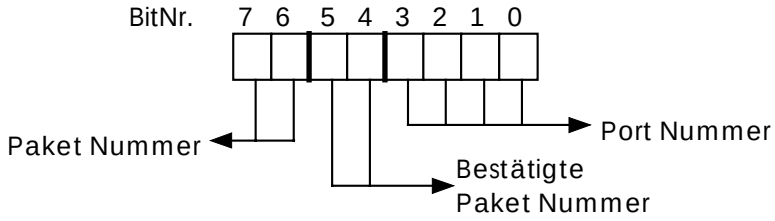


*Abbildung 5.2 - Aufbau Byte Port Nummer*

  sichergestellt, dass die Pakete chronologisch richtig und auch nicht doppelt in den Empfangsbuffer geschrieben werden. In 2 weiteren Bits dieses Bytes wird der Empfang von Paketen rückbestätigt (Abbildung 5.2). Diese Bits enthalten dann die entsprechende Paketnummer.

- **ID des Senders:** Enthält die Stations-ID des Absenders. Diese liegt im Bereich von 0 bis 15.
- **Anzahl Datenbytes:** Gibt an, wieviele Datenbytes folgen, bis die CRC Prüfsumme kommt.
- **CRC Prüfsumme:** Prüfsumme, mit der überprüft wird, ob die Daten fehlerfrei empfangen wurden. Details dazu in Kapitel 5.5.

Da das verwendete Radiomodem selbst keinerlei Bitsynchronisation vornimmt, ist es die Aufgabe der Sendestation, der seriellen Schnittstelle der Empfangsstation genügend Flankenwechsel anzubieten, damit eine Synchronisation auf dieser Stufe erreicht werden kann [RM95]. Wird also eine Null übermittelt, besteht keinerlei Möglichkeit für eine solche Synchronisation, da keine Flankenwechsel stattfinden. Es muss also ein Codierungsverfahren verwendet werden, das diese Aufgabe erfüllt. Aus der schier unendlichen Zahl von Verfahren wurde die Bi-Phase Codierung (Kapitel 4.1.3) ausgewählt. Dadurch wird zwar die Bandbreite halbiert, die Übertragung verläuft aber wesentlich zuverlässiger. Sämtliche Bytes nach dem Startzeichen bis inklusive der CRC Prüfsumme sind während der Übertragung Bi-Phase codiert. Diese Codierung muss softwaremässig vorgenommen werden.

## 5.4 DATENFLUSSKONTROLLE

Die Datenflusskontrolle übernimmt Funktionen, um die Kommunikation zwischen Sender und Empfänger sicherzustellen und findet in allen Schichten des Netzwerkes statt. Ein Prozess initiiert eine Übertragung durch Aufruf einer Funktion einer darunter liegenden Schicht. Die Datenflusskontrolle sorgt nun dafür, dass die Übertragung, wie immer sie auch technisch abläuft, dem sendenden sowie dem empfangenden Prozess folgende Garantien bietet:

- Die chronologische Reihenfolge bleibt gewährleistet. Dies bedeutet, dass die Daten in der Reihenfolge beim Empfänger ankommen, in der sie der Sender abgeschickt hat.
- Integrität der Daten ist sichergestellt. Die Datenflusskontrolle benutzt Verfahren, um fehlerhaft übertragene Daten zu erkennen und zu beheben.

Um obige Garantien geben zu können, müssen verschiedene Schichten des Netzwerkprotokolles spezielle Aufgaben erfüllen. Im folgenden wird das OSI-Referenzmodell [TAN92] verwendet, um die Protokollhierarchie darzustellen.

- **Bitübertragungsschicht:** Diese Funktionen werden gänzlich von der installierten Hardware, den Radiomodems [RM95], übernommen und können für dieses Projekt als gegeben betrachtet werden.
- **Sicherungsschicht:** Die Sicherungsschicht überprüft empfangene Daten auf deren Richtigkeit. Dies geschieht aufgrund der mit dem Paket erhaltenen Prüfsumme sowie der Paketnummer. Ist das Paket korrekt empfangen worden, muss der Sicherungsschicht des Senders dies mitgeteilt werden, da diese sonst nicht mit Sicherheit wissen kann, ob die gesendeten Daten auch wirklich angekommen sind. Daher wird ein positives Bestätigungssystem verwendet. Das bedeutet, dass richtig empfangene Pakete bestätigt werden, falsch empfangene jedoch nicht. Nach einer bestimmten Zeit wird der Sender dieses Pakets automatisch wiederholen, solange bis er eine positive Bestätigung über dessen Empfang erhält. Das Netzwerk verwendet also grundsätzlich zwei Arten von Paketen:
  - **Datenpakete:** Der Aufbau der Datenpakete wurde im vorherigen Kapitel beschrieben. Datenpakete enthalten immer eine Paketnummer im Bereich von 1 bis 3.
  - **Bestätigungspakete:** Diese entsprechen grundsätzlich der Struktur der Datenpakete nur mit dem einen Unterschied, dass die Länge der Daten Null beträgt. Bestätigungspakete enthalten immer die Paketnummer 0, was bedeutet, dass sie eigentlich keine Nummer haben und selbst vom Empfänger nicht bestätigt werden müssen.

  Funk ist ein stark störanfällliges Übertragungsmedium. Atmosphärisches Rauschen sowie andere Sendestationen auf der gleichen Frequenz können eine Übertragung stören oder sogar unterbinden. Somit muss sichergestellt

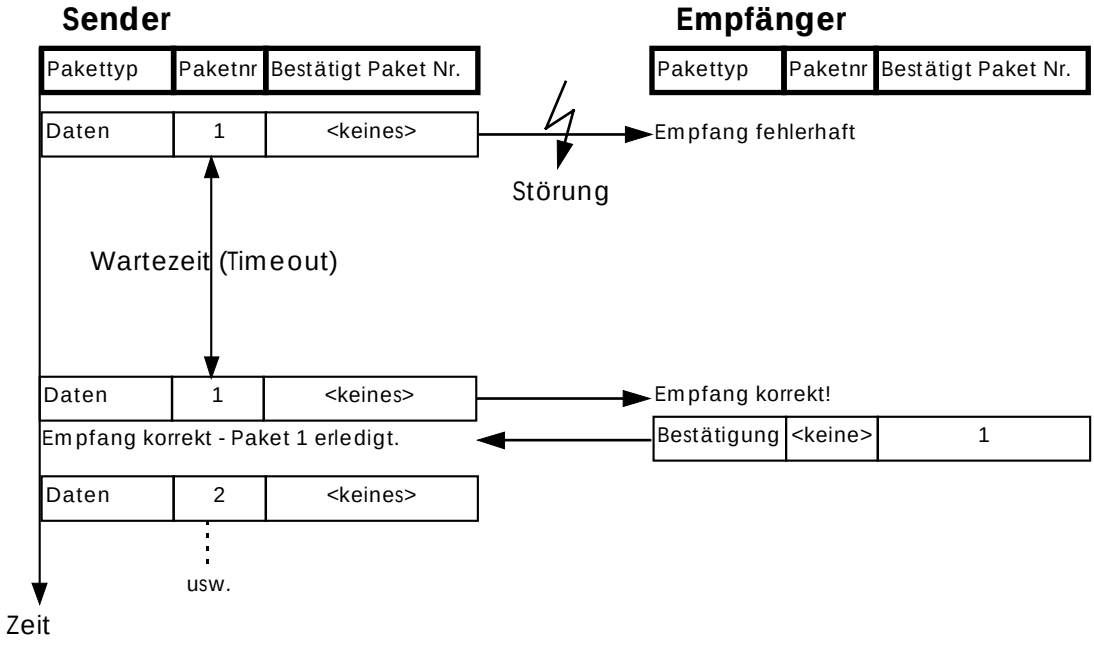


*Abbildung 5.3 - Fehlertyp 1 während einseitiger Übertragung*

werden, dass der Sender weiss, welche von ihm abgesandte Pakete auch wirklich korrekt empfangen wurden. Dies kann nur dadurch erreicht werden, dass der Empfänger den korrekten Empfang des Datenpaketes dem Sender zurückmeldet. Bei Störungen sind nun zwei Szenarien denkbar, die in der Praxis auftreten können:

Im ersten Fall ist die Übertragung eines Datenpaketes fehlerhaft (Abbildung 5.3). Der Sender wartet nun vergebens auf eine Bestätigung und zwar solange, bis die Warte- bzw. Timeoutzeit verstrichen ist. Danach wiederholt er das Paket mit der gleichen Paketnummer ein weiteres Mal. Dies wiederholt sich solange, bis er eine Bestätigung für dieses Paket erhält. Danach inkrementiert der Sender die Paketnummer und fährt mit der Übertragung fort. In einem zweiten Fall wäre denkbar, dass nicht das Datenpaket fehlerhaft übertragen wird, sondern die Bestätigung desselben (Abbildung 5.4). Dadurch wird der Empfänger das identische Paket zwei-

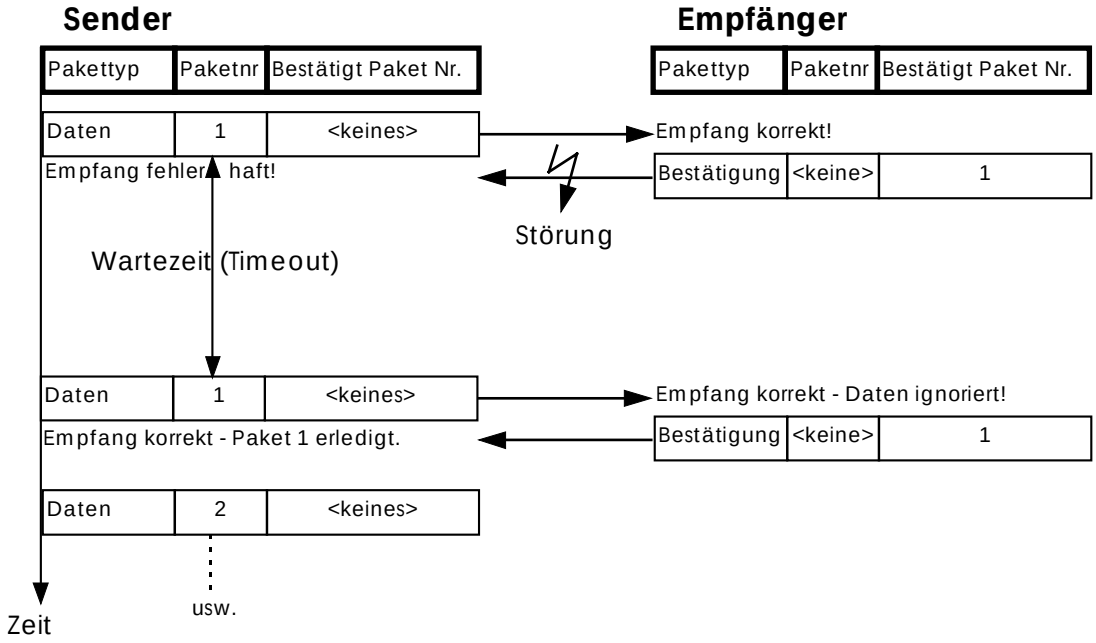


*Abbildung 5.4 - Fehlertyp 2 während einseitiger Übertragung*

oder mehrere Male empfangen. Da dieser aber die Paketnummern des Senders mitzählt, stellt er fest, dass das vorherige Paket die gleiche Nummer hatte und verwirft daher die Daten. Das Paket wird jedoch trotzdem bestätigt, da ja offensichtlich die Bestätigung beim letzten Mal nicht angekommen ist. Es wäre ja sonst nicht mehr wiederholt worden. Die Paketnummern sind unter anderem ein wesentlicher Bestandteil, damit mehrfach übertragene Pakete erkannt werden können. Dies bedingt jedoch auch immer, dass ein potentieller Empfänger die aktuelle Paketnummer des Senders im voraus kennt. Wurde aber noch kein Paket übertragen, kann der Empfänger diese Nummer noch gar nicht kennen. Daher wird die erste von einer Station nach dessen Initialisierung empfangene Paketnummer immer als die richtige Nummer akzeptiert.

Die Leistung dieses Übertragungsprotokolles hängt im wesentlichen von der Warte- oder Timeoutzeit ab. Sie ist nicht einfach ein konstanter Wert, sondern besteht aus zwei Komponenten.

- **Timeoutzeit:** Die Timeoutzeit gibt die Dauer an, während der keine Pakete verschickt werden und nur auf den Erhalt einer Bestätigung gewartet wird. Wird während dieser Zeit keine Bestätigung erhalten, so muss das Paket als fehlerhaft übertragen betrachtet werden und wird zu einem späteren Zeitpunkt wiederholt. Wird jedoch die Bestätigung vor Ablauf der Frist erhalten, so kann der Sender mit der Übertragung weiterer Pakete fortfahren.

Da Bestätigungspakete selbst nicht mehr bestätigt werden, stellen diese einen Spezialfall dar. Die empfangende (bestätigende) Station darf selbst keine Datenpakete zum Sender schicken, bevor nicht sichergestellt ist, dass seine Bestätigung angekommen ist. Dies weiss sie erst, wenn nach abgelaufener Timeoutzeit (inkl. maximaler Wiederholungszeit) das gleiche Paket nicht nochmal oder bereits ein neues Paket bei ihr ankommt.

- **Wiederholungszeit** ('rescheduling time'): Nach Ablauf der Timeoutzeit muss der Sender eine Wiederholung des betroffenen Paketes einleiten. Da aber unter Umständen eine Kollision mit einer anderen Station zum Verlust des Paketes geführt hat, darf nicht eine fixe Wartezeit der Wiederholung vorausgehen, da die andere betroffene Station das gleiche tun wird. Daher wird hier eine Zufallsvariable bestimmt, die sich innerhalb gewisser Grenzen bewegt und zufällig einer Station den Vorrang geben wird.

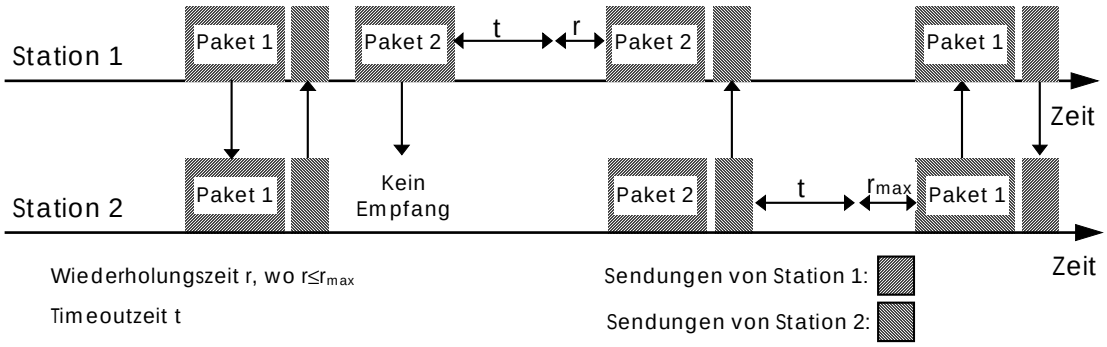


*Abbildung 5.5 - Beispiel für Datenflusskontrolle*

In Abbildung 5.5 wird die Datenflusskontrolle anhand eines Beispiels gezeigt. Station 1 führt als erstes eine Übertragung von Paket Nummer 1 durch, das auch prompt von Station 2 bestätigt wird. Aufgrund der Bestätigung ist die Verbindung wieder frei für neue Pakete, worauf Station 1 ein weiteres Paket, Nummer 2, übermittelt. Dieses kommt jedoch nur fehlerhaft bei Station 2 an, wodurch keine Bestätigung von Seiten der Station 2 erfolgt. Sobald die Timeoutzeit t abgelaufen ist, wird eine Wiederholung nach einer zufälligen Wiederholungszeit r eingeleitet. Station 2 bestätigt den erfolgreichen Empfang, wodurch die Verbindung für Station 1 wieder frei ist. Station 2 muss jedoch noch eine Timeoutperiode sowie die maximale Wiederholungszeit abwarten, bevor sie wieder senden darf und ihr Paket Nummer 1 übermitteln kann.

- **Transportschicht:** Der stetige Datenstrom eines Prozesses wird hier in kleinere Einheiten unterteilt und so für den paketorientierten Transport vorbereitet.
- **Sitzungsschicht:** Die Sitzungsschicht verwaltet die von den einzelnen Prozessen verlangten Verbindungen.

Eine Station besitzt 16 virtuelle Kanäle, über die sie Verbindungen zu anderen Stationen aufnehmen kann. Die Datenflusskontrolle behandelt jedoch jeweils eine ganze Station als Ganzes und macht keinen Unterschied, auf welchem Port eine Kommunikation stattfindet.

## 5.5 FEHLERERKENNUNGSVERFAHREN

Als Fehlererkennungsverfahren wird der 'Cyclic Redundancy Check' (CRC) [PFT86] verwendet. Da die Berechnung einer CRC sofwaremässig relativ aufwendig

ist, wird zu Beginn des Programmes eine Tabelle angelegt, die die Bestimmung der CRC enorm beschleunigt. Verwendet wird eine 16-Bit CRC in der Form, wie sie auch in anderen Anwendungen benutzt wird, z.B. im ZModem Protokoll.

## 5.6 DAS DIDABOT CONTROL INTERFACE - DCI

### 5.6.1 Grundsätzliche Überlegungen

Das Didabot Control Interface (DCI) stellt die notwendige Funktionalität zur Verfügung, um einen Didabot von einem PC aus fernzusteuern. Damit ein PC Programm einen Roboter steuern kann, bedarf es dessen Sensordaten sowie der Motorzustände. Man kann daher diese Problemstellung als ein einfaches Telemetriesystem betrachten. Die Funktionen zur Übermittlung der Daten werden jedoch von anderen Komponenten übernommen, die in vorherigen Kapiteln näher beschrieben wurden. DCI hat daher nur noch die Aufgabe, dem Anwender eine Schnittstelle anzubieten, über die er vollständige Kontrolle über den Didabot erhält. Damit auch Rückschlüsse über die Leistung eines Programmes gemacht werden können, müssen Möglichkeiten zur Veranschaulichung der internen Vorgänge angeboten werden. Dies kann darin bestehen, dass die Daten am Bildschirm visualisiert werden. Andererseits soll eine nachträgliche, eingehendere Analyse dadurch ermöglicht werden, dass die zur Laufzeit ermittelten Sensordaten sowie Motorzustände während der Ausführung in eine Datei protokolliert werden.

DCI ist im Grunde nichts anderes als eine Softwarebibliothek, die all diese Bedürfnisse befriedigt. Ein Anwender schreibt nur noch sein individuelles Steuerprogramm (UCP - 'User Control Program') für den Roboter und compiliert dieses zusammen mit DCI. Jedes ausführbare DCI-Programm besteht daher aus zwei Teilen. Erstens der DCI Softwarebibliothek, die sämtliche Einstellungen zur Kommunikation, Visualisierung und Protokollierung der Ein- und Ausgabeströme vornimmt und zweitens dem Anwenderprogramm (UCP), das die eigentliche Robotersteuerung übernimmt und von Fall zu Fall unterschiedlich sein kann. Ausser den von DCI angebotenen Funktionen sind natürlich auch alle Betriebssystemfunktionen verfügbar und geben dadurch dem Anwender maximale Flexibilität für seine Entwicklungen.

Damit DCI die vom Anwenderprogramm generierten Steuerbefehle auf dem Roboter ausführen lassen kann, wird ein Programm auf dem Didabot selbst benötigt.

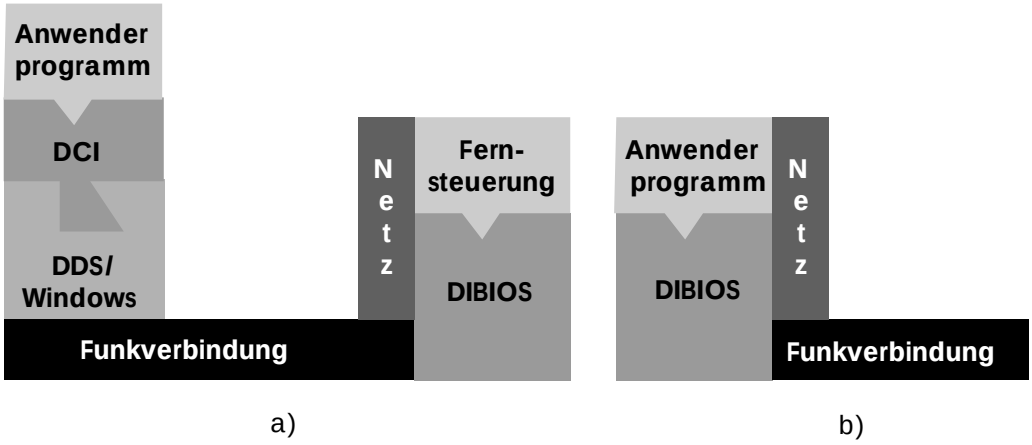


*Abbildung 5.6 - Schnittstellendesign mit a)hostgestützter oder b)direkter Steuerung*

Dieses empfängt die Steuerbefehle von DCI und setzt diese effektiv in Bewegungen des Roboters um. In manchen Fällen ist es aber nach wie vor wünschenswert, das Anwenderprogramm nicht auf dem PC, sondern direkt auf dem Didabot auszuführen. Dazu existierte bereits vor dieser Arbeit ein einfaches Betriebssystem (DIBIOS -

'Didabot BIOS') [MS95], das den Zugriff auf die Roboterhardware vereinfachte. Um nun maximale Portabilität von Steuerprogrammen zu erreichen besitzt DCI die identische Schnittstelle wie das DIBIOS. Dadurch kann ein Anwenderprogramm ohne Änderungen sowohl auf dem PC mittels DCI als auch auf dem Roboter selbst mittels DIBIOS ausgeführt werden (Abbildung 5.6).

### 5.6.2 Datenströme

In der Betrachtungsweise eines Telemetriesystems stellt DCI die Station dar, die die Datenauswertung und -verarbeitung vornimmt. In diesem Fall führt diese Datenauswertung zur Ausgabe von Steuersignalen, die wiederum zum Roboter übermittelt und dort ausgeführt werden. Es existieren also Datenströme in beide Richtungen. Beim Design von DCI werden daher Ein- und Ausgabeströme allgemein definiert. Als Eingabeströme sind denkbar:

- Sensordaten empfangen via Radiomodem. Dies dürfte der Normalfall sein.
- Sensordaten einer zuvor in einer Datei gespeicherten Situation.
- Sensordaten über Netzwerk empfangen von einem anderen Rechner, der evtl. Vorverarbeitungen erledigt hat.
- Virtuelle, berechnete Sensordaten für einen imaginären Raum.

Ähnliches gilt für Ausgabeströme:

- Steuerbefehle an Roboter übermitteln via Radiomodem.
- Steuerbefehle in eine Datei protokollieren.
- Steuerbefehle über Netzwerk an einen anderen Rechner zur Weiterverarbeitung senden.
- Simulation der Roboterbewegungen in einem virtuellen Raum.

Die Implementation wird vorerst nur jeweils die ersten beiden Varianten ermöglichen. Zur Abfrage der Sensordaten bzw. zur Steuerung des Roboters bedarf es vordefinierter Befehle. Die Struktur dieser Befehle ist nachfolgend aufgelistet. Sie besteht immer aus einem Steuerbyte, das den Befehl identifiziert, gefolgt von den notwendigen zusätzlichen Daten. Falls darauf eine Antwort erwartet wird, wird diese immer mit dem gleichen Steuerbyte eingeleitet.

- **Lese Sensordaten:**

| PC sendet | Didabot antwortet mit |
|---|---|
| **'A'** | 'A' |
| | Distanzmessung Sensor 0 - 5 |
| | Helligkeitsmessung Sensor 0 - 5 |
| | Aktuelle Geschwindigkeit Motor 0 - 1 |

- **De-/Aktiviere Licht:**

| PC sendet | Didabot antwortet mit |
|---|---|

| **'B'** | <keine Antwort> |
|---|---|
| 0=Licht aus, <>0 Licht an | |

- **Setze Geschwindigkeit:**

| PC sendet | Didabot antwortet mit |
|---|---|
| **'D'** | <keine Antwort> |
| Neue Geschwindig-keit Motor 0 | |
| Neue Geschwindig-keit Motor 1 | |

- **Setze Lautsprecherfrequenz:**

| PC sendet | Didabot antwortet mit |
|---|---|
| **'S'** | <keine Antwort> |
| Frequenz High-Byte | |
| Frequenz Low-Byte | |

- **Shell Modus ein/aus:**

| PC sendet | Didabot antwortet mit |
|---|---|
| **'T'** | <keine Antwort> |
| 0=Shell aus, <>0 Shell ein | |

Da in den meisten Anwendungen die Distanz- und Helligkeitsmessungen aller Sensoren benötigt werden, wurde nur eine einzige Funktion implementiert, mit der diese abgefragt werden können. Dies hat den grossen Vorteil, dass die Daten innerhalb dieses Datenpaketes immer zeitgleich ermittelt werden. Müsste jeder Sensor einzeln abgefragt werden, würde dies erstens zu einer ineffizienten Auslastung des Netzwerkes führen und zweitens für jeden Sensorwert einen anderen Messzeitpunkt ergeben. Dies wäre jedoch keine gute Basis für Entscheidungen des weiteren Verhalten des Roboters.

# 6 IMPLEMENTATION

Die Implementationsarbeiten umfassten Programmteile sowohl auf dem Didabot als auch auf dem PC. Durch die Veränderung der Funktionalität des DDS Servers wurden etliche weitere Komponenten auf dem PC indirekt betroffen (Abbildung 6.1). Die Standardbibliothek für den Zugriff auf den DDS Server musste um die neuen Kommunikationsfähigkeiten erweitert werden. Desweiteren musste das

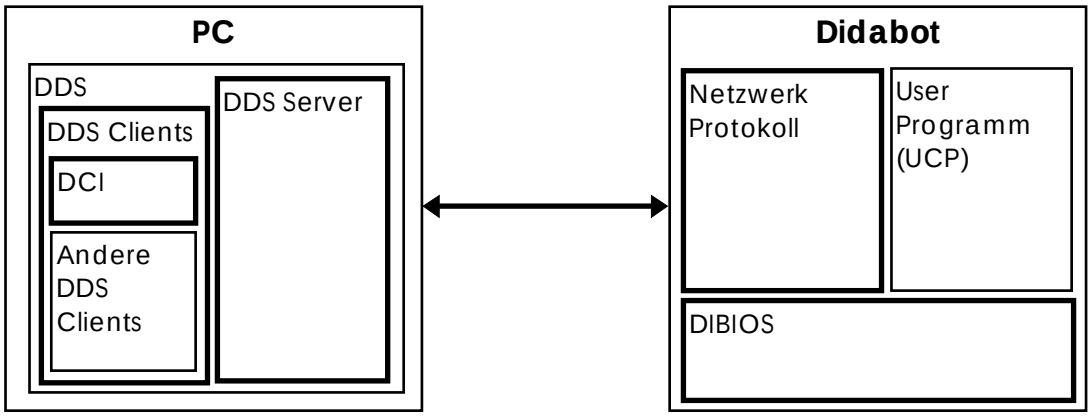


*Abbildung 6.1 - Neue/Umgeschriebene DDS Komponenten (fett dargestellt)*

DIBIOS mit dem Netzwerkprotokoll vereint werden. Gänzlich neu wurde das DCI entwickelt, das neben der Funkübermittlung die zweite wesentliche Erweiterung darstellt. In den anschliessenden Unterkapiteln werden alle diese Änderungen bzw. Neuentwicklungen näher beschrieben.

## 6.1 DIDABOT PROGRAMMTEILE

Die Hauptarbeit auf dem Didabot bestand darin, das gewählte Netzwerkprotokoll zu implementieren und mit dem bestehenden DIBIOS des Roboters zu vereinigen. Das DIBIOS wurde mit dem HITECH C-Compiler entwickelt, wodurch der Entscheid vorweggenommen ist, dass auch das Netzwerkprotokoll in C programmiert wird. Dadurch ist es auch möglich, dieses auf andere Systeme zu portieren, wodurch sich die Verwendbarkeit nicht nur auf die Didabots beschränkt.

Grundsätzlich kann das Netzwerkprotokoll in zwei grössere Subkomponenten unterteilt werden, die lose miteinander gekoppelt sind:

- **Empfangen**: Der Empfang von Paketen geschieht über eine Interruptroutine. Diese wird aufgerufen, sobald ein Zeichen an der seriellen Schnittstelle verfügbar ist. Der Vorgang des Empfangens kann als eine Finite State Maschine betrachtet werden. Durch den Empfang eines Zeichens geht diese von einem Zustand in einen anderen über, abhängig vom empfangenen Zeichen. Ist ein komplettes Paket empfangen worden, wird die Prüfsumme kontrolliert und die Daten in den entsprechenden Empfangsbuffer geschrieben. Gleich anschliessend wird das Senden eines Bestätigungspaketes ausgelöst.

- **Senden**: Das Senden geschieht über den Aufruf einer angebotenen Systemfunktion, das die zu sendenden Daten in einen temporären Buffer schreibt. Dort werden sie zu einem späteren Zeitpunkt von einer Interruptroutine gelesen. Diese Interruptroutine wird in bestimmten Zeitperioden aufgerufen und überprüft, ob neue Daten im Sendebuffer sind. Ist dies der Fall, so wird ein neues Datenpaket generiert und anschliessend gesendet, sofern der Kanal frei ist. Dies wird festgestellt, indem der Status der

Empfangsroutine, der Finite State Maschine, abgefragt wird. Befindet sich diese im Initialzustand, so steht der Kanal zur Verfügung und die Übermittlung wird eingeleitet. Die Interruptroutine wird nun erst dann wieder aufgerufen, wenn das Paket von der Gegenstation bestätigt wurde oder wenn die Timeoutzeit abgelaufen ist.

Der 80C196KD [INT92] Prozessor, mit dem der Didabot bestückt ist, bietet zwei Timer an. Das Problem ist nun, dass beide Timer bereits verwendet werden. Timer 1 wird zur Ermittlung der aktuellen Geschwindigkeit verwendet und tritt alle 52.4ms auf. Timer 2 dient der Tongenerierung und hat unterschiedliche Laufzeiten, je nach gewählter Frequenz. Als Lösung teilen sich Motorsteuerung sowie Netzwerkprotokoll Timer 1. Da jedoch die Frequenz von 19.1Hz zu gering ist, um ein effizientes Handling von Datenpaketen zu gewährleisten, wurde die Frequenz um das achtfache erhöht. Die Routinen zur Ermittlung der Geschwindigkeit werden nun nur noch jedes achte Mal aufgerufen, um die alte Frequenz von 19.1 Hz für diese Funktion zu gewährleisten. Das Netzwerkprotokoll hingegen nutzt jeden Interruptaufruf zur Übermittlung von neuen Paketen, wodurch eine akzeptable Leistung erreicht wird.

Um Daten asynchron versenden und empfangen zu können, müssen entsprechende temporäre Zwischenspeicher vorhanden sein. Da der Speicherplatz auf dem Didabot mit 32 KB ziemlich beschränkt ist, darf die Grösse dieser Buffer nicht zu gross gewählt werden, v.a. da diese für jeden einzelnen der 16 virtuellen Ports getrennt existieren müssen. Eine Grösse von je 256 Bytes scheint für die meisten Anwendungen absolut ausreichend.

## 6.2 PC PROGRAMMTEILE

Sämtliche PC Programmteile wurden unter Windows 3.1 implementiert, funktionieren jedoch genauso unter Windows 95. Der Speicherbedarf von DDS ist relativ bescheiden, jedoch sollte man mindestens über 8 MB RAM verfügen. Es sollte mindestens ein Intel 486 Prozessor zur Verfügung stehen, da der DDS Server zur Steuerung des Netzwerkprotokolles schon einige Rechenzeit in Anspruch nimmt. Werden dann noch weitere Programme ausgeführt, kann dies unter Umständen bei schwächeren Rechnern zu Problemen führen.

### 6.2.1 Der DDS Server

Der DDS Server ist das Bindeglied zwischen den DDS Client Applikationen und den Didabots. Die Inter-Applikationskommunikation wurde mit DDE, einem Systemdienst von Windows realisiert. Durch die Einführung der Kommunikation über Funk änderte sich an diesem Teil kaum etwas und entspricht bis auf eine kleine

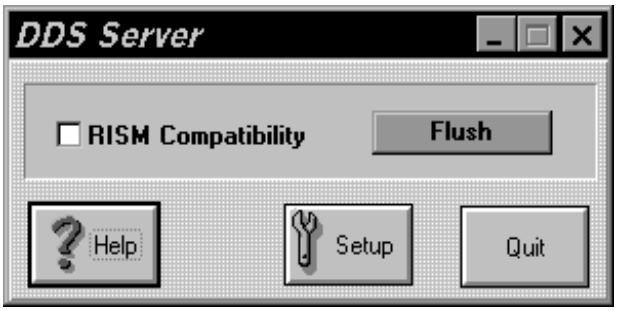


*Abbildung 6.2 - Der neue DDS Server*

Ausnahme der alten Version [RE95]. Die Ausnahme besteht in der Reservierung des gewünschten Ports, der von nun an allein nicht mehr ausreicht, um das Ziel der Kommunikation zu bestimmen. Neu ist nun auch die Angabe des Zielroboters mittels der sogenannten Stations-ID notwendig. Diese wird als zweiter Parameter dem DDE Befehl *GETPORT(2, 1)* [RE95] (reserviert Port 2 auf Roboter mit ID 1) mitübergeben.

Komplett neu geschrieben wurde das Netzwerkprotokoll zur Kommunikation mit dem Didabot. Zwar ist der DDS Server in Borland Pascal 7.0 geschrieben, doch entspricht das Protokoll dem C Programm, das auf dem Didabot läuft. Daher erübrigt sich eine nähere Erläuterung.

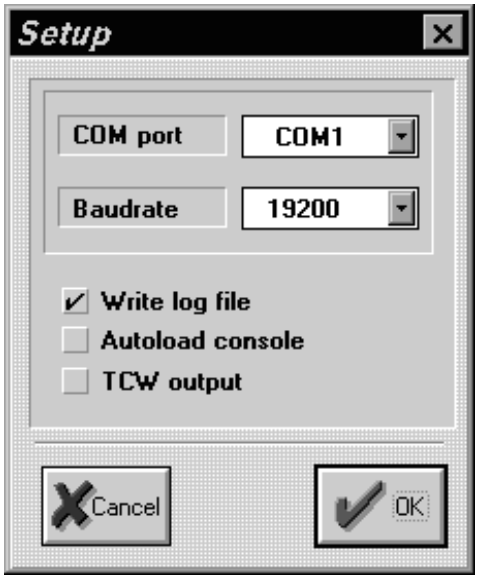


*Abbildung 6.3 - Setup Menu des DDS Servers*

Neu ist es möglich, diverse Einstellungen des DDS Servers vorzunehmen (Abbildung 6.3). Ausser des seriellen Ports lassen sich auch Baudrate und diverse andere Optionen verändern:

- **Write log file**: Ermöglicht eine Protokollierung aller gesendeten und empfangenen Pakete. Dadurch ist ein Einblick in die Funktionsweise des Netzwerkes möglich. Die Informationen werden standardmässig in die Datei 'LOGFILE.DAT' im Verzeichnis des DDS Servers gespeichert und können mit einem einfachen Texteditor betrachtet werden.
- **Autoload console**: Da die DDS Console [RE95] ein häufig verwendetes Werkzeug ist, kann sie durch Setzen dieser Option gleichzeitig mit dem Start des DDS Servers geladen werden.
- **TCW output**: Diese Option öffnet neben dem Hauptfenster des DDS Servers ein zweites Fenster, worin die aktuellen Vorgänge auf dem Netzwerk dargestellt werden (TCW - 'Traffic Control Window').

Alle diese Einstellungen werden beim Beenden des Programmes gespeichert.

### 6.2.2 Das Didabot Control Interface

Das Didabot Control Interface (kurz DCI) ist eine komplette Neuentwicklung. Da bisher alle DDS Komponenten auf dem PC mit Borland Pascal 7.0 entwickelt wurden, lag es nahe, dies auch für DCI zu verwenden. Doch da für die Implementierung der Steuerprogramme auf dem Didabot bisher C verwendet wurde und Portabilität des Quellcodes erreicht werden sollte, war es notwendig, auch für DCI C zu verwenden. Damit auch wie bei Borland Pascal die Objektbibliothek OWL ('Object Window Library') verwendet und damit ein minimaler Grad an Konsistenz erreicht werden konnte, wurde Borland C++ 4.0 als Programmiersprache gewählt.

Das Programm DCI besteht aus mehreren Komponenten, die nach dem Programmstart den erscheinenden Fenstern entsprechend zugewiesen werden können. Als erste und wichtigste Komponente soll nun das DCI Kontrollfenster

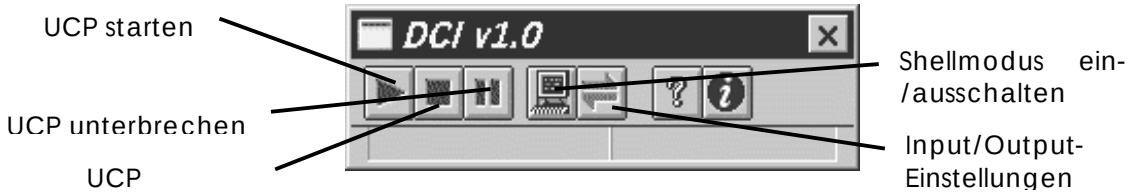


*Abbildung 6.4 - Das DCI Kontrollfenster*

betrachtet werden (Abbildung 6.4). Es gibt dem Anwender die Möglichkeit, den Ausführungsstatus seines eigenen Kontrollprogrammes (UCP), das zusammen mit DCI compiliert wurde, zu verändern. Ein UCP kann sich in drei möglichen Zuständen befinden - entweder wird es gerade ausgeführt, ist gestoppt oder kurzfristig unterbrochen. Diese drei Zustände lassen sich über die ersten drei Schalter links einstellen. Nach dem Programmstart befindet sich das UCP automatisch im Haltezustand.

Mit der mittleren Schaltergruppe können verschiedene Einstellungen vorgenommen werden. Wichtig ist dabei vor allem die Wahl der Eingabe- und Ausgabekanäle. Wird dieser Schalter angewählt, so erscheint eine weitere Dialogbox, die die verschiedenen Varianten anbietet. Unabhängig voneinander lassen sich

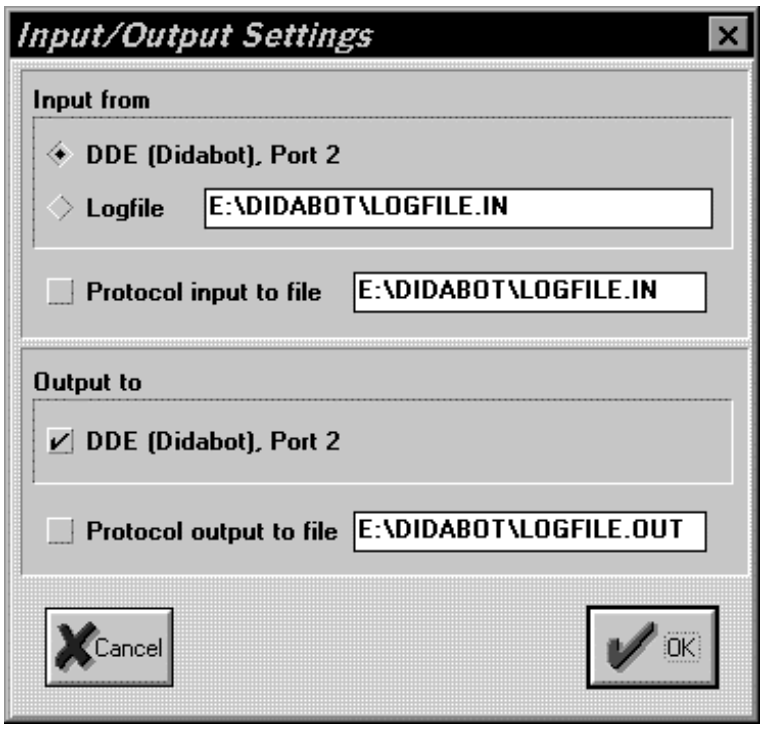


*Abbildung 6.5 - Die Ein-/Ausgabemöglichkeiten von DCI*

während der Ausführung des UCP dessen Ein- wie auch Ausgabestrom in eine Datei protokollieren und stehen für spätere Analysen dem Anwender zur Verfügung. Dies geschieht durch Setzen der entsprechenden Marke mit Angabe der Datei, in die diese Informationen geschrieben werden sollen. Ebenfalls ist möglich, ein gespeichertes Protokoll als Eingabestrom für das UCP wiederzuverwenden. Dadurch kann ein Programm immer wieder einer einmal protokollierten Situation ausgesetzt und dessen Reaktion aufs Neue beurteilt werden. Es muss jedoch berücksichtigt werden, dass in diesem Fall die Steuersignale keinen Einfluss auf die empfangenen bzw. aus einer Datei gelesenen Sensordaten haben. Generiert das UCP nämlich

andere Steuersignale, so müssten sich dadurch ja auch die weiteren Sensorwerte entsprechend ändern.

Das Format der gespeicherten Daten entspricht dem in Kapitel 5.6.2 gezeigten. Zusätzlich werden dazu noch die zeitlichen Abstände der einzelnen Datenpakete mitgespeichert. Dadurch kann eine Datei in der gleichen Geschwindigkeit abgespielt werden, in der sie ursprünglich gespeichert wurde.

Zur Visualisierung der Sensordaten besitzt DCI zusätzliche Fenster. In diesen werden jeweils die aktuellen Werte der Sensoren angezeigt. Dabei wird der Didabot als abstraktes, sechseckiges Symbol dargestellt, an dessen Kanten jeweils ein Sensor angebracht ist. In einem Fenster wird zum Beispiel die gemessene Helligkeit dargestellt

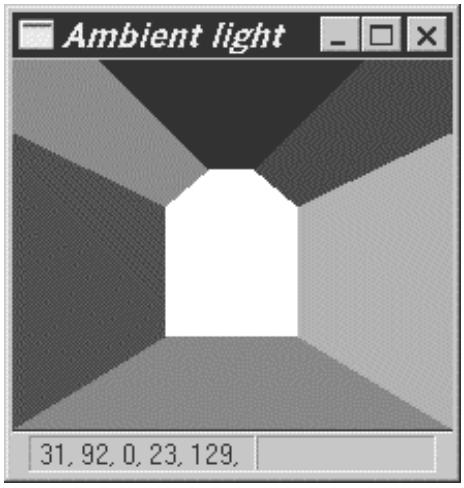


*Abbildung 6.6 - Visualisierung der Helligkeit*

(Abbildung 6.6). Die Helligkeit der zu einer Kante des Didabots hin verlaufende Fläche entspricht dem gemessen Wert in diese Richtung. Dadurch entstehen sechs verschiedene Sektoren, dessen Grauwerte äquivalent zu den Helligkeiten sind. Die exakten Messwerte werden zusätzlich in der Statuszeile am unteren Rand des Fensters eingeblendet. In ähnlicher Weise werden die Distanzmesswerte dargestellt

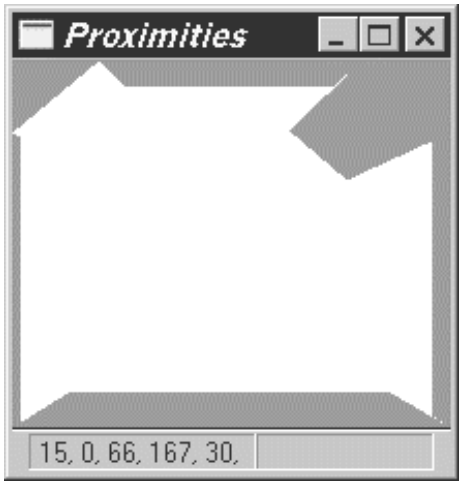


*Abbildung 6.7 - Visualisierung der Distanzen*

(Abbildung 6.7). Je näher sich gemäss den Messwerten Objekte in den sechs Richtungen befinden, desto näher bewegen sich die Kanten der sechs Sektoren zur Mitte des Fensters bzw. desto näher zum abstrahierten Didabot. Dadurch kann sich

ein Benutzer leicht über die momentane Situation des Roboters informieren. Ein weiteres Fenster stellt die aktuelle Geschwindigkeit des Roboters, bzw. seiner beiden Motoren dar (Abbildung 6.8). Da der Roboter sich vorwärts (positive Messwerte) wie auch rückwärts (negative Messwerte) fortbewegen kann, befindet sich der

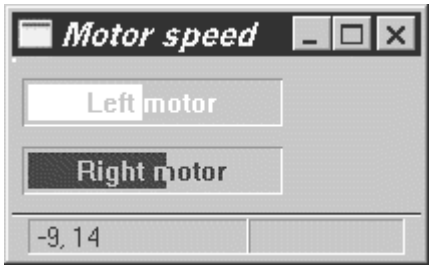


*Abbildung 6.8 - Visualisierung der Motorgeschwindigkeiten*

Nullzustand in der Mitte der Balkenanzeige. Bei Vorwärtsbewegungen wird der Balken blau, andernfalls rot eingefärbt, um die Fahrtrichtung besonders deutlich zu visualisieren.

#### 6.2.2.1 Das Modulkonzept

DCI soll grundsätzlich auch in Verbindung mit anderen Robotern verwendet werden können. Da das Netzwerkprotokoll in C programmiert wurde, ist es leicht auf andere Prozessorplattformen portierbar. Daher sollte nun auch DCI flexibel genug sein, um die Sensordaten anderer Roboter zu protokollieren bzw. zu visualisieren. Die Methode der Visualisierung hängt jedoch von der Art des Sensors ab. Um den Spielraum in der Wahl der Sensoren nicht unnötig einzuschränken, wurde DCI so flexibel wie möglich gehalten. DCI bietet für einen beliebigen Sensor eines Roboters drei Funktionen an (Abbildung 6.9).

- **Visualisierung**: Die erhaltenen Messdaten werden am Bildschirm in einem Fenster dargestellt. Alle diese Fenster beruhen auf einem abstrakten Basisobjekt, von dem alle effektiven Visualisierungsfenster abgeleitet werden. Innerhalb dieses Fensters kann nun die Art und Weise der Darstellung frei gewählt werden. Vorgegebene virtuelle Methoden übernehmen dabei bestimmte, spezifische Funktionen wie zum Beispiel die Aktualisierung der Darstellung etc.

- **Protokollierung**: DCI kann auf Wunsch sämtliche Messwerte in eine Datei protokollieren. Da die Messwerte jeweils in einem Datenblock übermittelt werden, werden diese dann auch in einem Block gespeichert.

- **Verarbeitung**: DCI bietet die Messwerte einem individuellen Anwenderprogramm, dem UCP, zur Weiterverarbeitung und Ermittlung von Steuerdaten an. Diese Schnittstelle besorgt sich die notwendigen Informationen über das Objekt des Visualisierungsfensters.

Soll nun ein neuer, weiterer Sensor einem Roboter hinzugefügt werden, so müssen nur diese drei Funktionen neu geschrieben werden. Die Protokollierung der Daten geschieht ohne Berücksichtigung auf deren Eigenschaften und bedarf daher keiner Änderung. Da die UCP Schnittstelle (Abbildung 6.9) ihre Daten von den Sensorobjekten holt, muss ein neues Objekt für diesen Sensor vom Basisobjekt abgeleitet werden und die spezifischen Methoden überschrieben werden. Damit auch das UCP auf diese Messdaten zugreifen kann, muss die UCP Schnittstelle um einen oder mehrere Funktionsaufrufe erweitert werden.

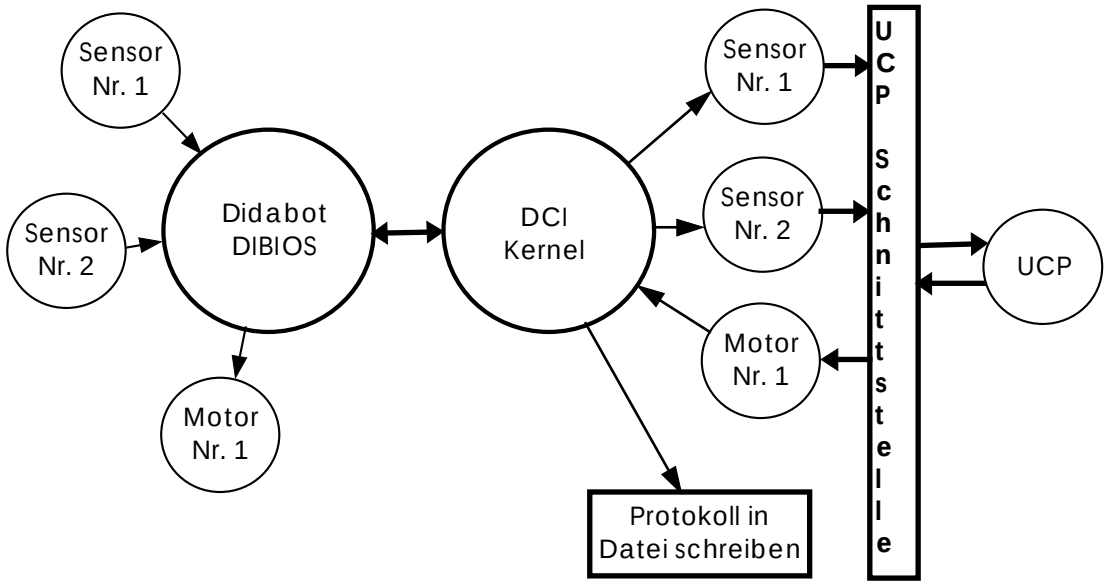


*Abbildung 6.9 - Das DCI Modulkonzept*

Damit kann DCI mit minimalsten Änderungen bzw. Erweiterungen flexibel an Roboter mit unterschiedlichsten technischen Eigenschaften und Fähigkeiten angepasst werden.

### 6.2.2.2 Programmfluss

Ein Anwenderprogramm (UCP) muss gewisse Bedingungen erfüllen, um innerhalb von DCI fehlerfrei funktionieren zu können. Grob besteht jedes UCP im wesentlichen aus zwei Hauptbestandteilen:

- **Initialisierungsfunktion**: Diese Funktion wird vor dem eigentlichen Start des UCP aufgerufen. Sie wird verwendet, um entsprechende Initialisierungen für das UCP vorzunehmen. Jedesmal also, wenn DCI das Anwenderprogramm startet, ruft es zuvor diese Funktion auf. Standardmässig muss diese

  *void init_control(void)*

  heissen. Auch wenn keine Initialisierungen vorgenommen werden, muss diese Funktion definiert sein.

- **Hauptschleife**: Die effektive Verarbeitung der Sensordaten und die daraus resultierenden Steuersignale werden innerhalb einer Hauptschleife berechnet. Dabei werden vor jedem Schleifendurchlauf die Sensordaten neu vom Roboter gelesen und stehen der Bearbeitung durch die Hauptroutine zur Verfügung (Abbildung 6.10). Diese Hauptschleife muss den Namen

  *void control(void)*

  tragen. Intern können selbstverständlich auch weitere eigene Funktionen eingeführt werden, doch ist diese der Einsprungpunkt von DCI.

Ausser den Funktionen von DCI stehen dem Anwenderprogramm sämtliche Windowsfunktionen zur Verfügung. Dadurch ist es zum Beispiel möglich, gewisse Parameter der Steuerung während der Ausführung anzupassen. Denkbar wäre auch eine Verteilung der Arbeit auf mehrere Rechner, die über Netzwerk ihre Ergebnisse an das UCP übermitteln.

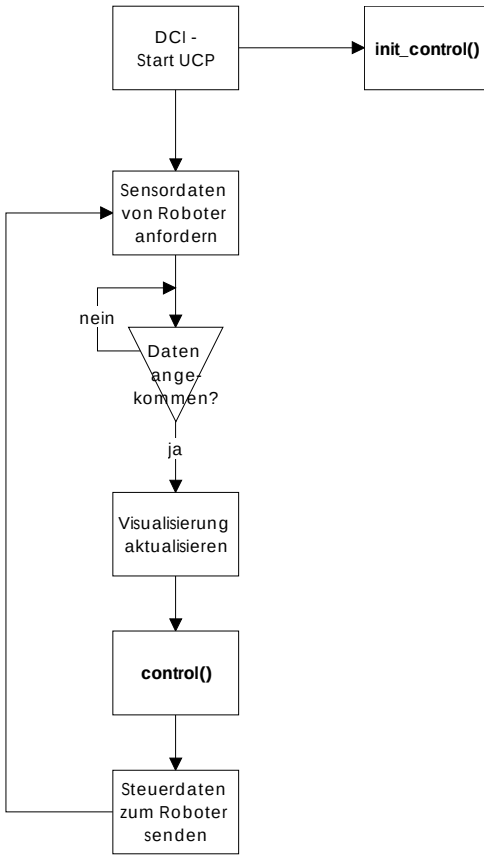


*Abbildung 6.10 - DCI Programmfluss*

# 7 SCHWACHSTELLENANALYSE

Da dieses Projekt eine zeitlich begrenzte Dauer hat, sollte nach abgelaufener Projektperiode eine Beurteilung vorgenommen werden. Diese Beurteilung beruht auf der Basis eines Vergleichs der gesetzten Ziele mit dem effektiv erreichten Resultat. Stellen, wo der Zielerreichungsgrad eher gering ausfällt, können als Schwachstellen angesehen werden. Dieses Kapitel soll solche Stellen aufzeigen, um die Möglichkeit für zukünftige Verbesserungen zu geben. Zu diesem Zweck wird nun die Aufgabenstellung dem erreichten Resultat gegenübergestellt.

- Als Kernaufgabe sollte eine Kommunikation über Funk zwischen einem oder mehreren Robotern und Computern implementiert werden. Dazu sollten die zur Verfügung gestellten Didabots mit dazugehörigen Funkmodems verwendet werden. Folgende Tabelle gibt einen Überblick über die Leistungsmerkmale des implementierten Protokolles (Tab. 7.1).

| | |
|---|---|
| Timeoutzeit | **60 ms** |
| Durchschnittl. Dauer bis Datenpaket bestätigt | **40 ms** |
| Durchschnittl. Dauer bis Paketwiederholung | **100 ms** |
| Maximale Paketgrösse | **16 Datenbytes** |
| Übertragungsrate einseitige Übermittlung | **2500-3500 Bits/Sek** |
| Übertragungsrate zweiseitige Übermittlung | **2000-3000 Bits/Sek** |
| Reichweite Funkmodem im Freien | **ca. 100 m** |
| Reichweite Funkmodem in Gebäuden | **ca. 25 m** |
| Anteil fehlerhafter Pakete (exkl. Kollisionen) | **1 - 10 %** |
| Durchschnittl. Anzahl Sensorpakete für DCI | **3 - 5 Refreshes/Sek** |
| Maximal nutzbare Geschwindigkeit für Didabot | **5 - 15 cm/Sek** |

*Tab. 7.1 - Leistungsmerkmale DDS-Netzwerkprotokoll*

Die Kommunikation wurde in verschiedenen Versuchen ausgetestet. Nach anfänglichen Schwierigkeiten konnte eine stabile, zuverlässige Kommunikation erreicht werden. Es erwies sich aber als ein wesentlichen Nachteil, dass die Modems (so wie sie eingebaut wurden) keine Möglichkeit boten, ein allfälliges, vorhandenes Trägersignal zu erkennen. Dadurch konnte erst bei Beginn eines effektiven Datenpaketes erkannt werden, dass eine andere Station sendet. In der Zeitspanne vom Starten des Senders bis zum Beginn des Datenpaketes ist es für eine Station unmöglich zu erkennen, dass schon eine andere Station den Kanal benutzt. Dies machte das CSMA Protokoll deutlich ineffizienter als ursprünglich angenommen, da Kollisionen schon bei nur zwei Stationen häufiger auftraten.

Die vom Didabot maximal unterstützte Übertragungsgeschwindigkeit betrug 19200 Bits/Sekunde. Dadurch musste man bereits beim Systemdesign davon ausgehen, dass wohl kaum Reserven bezüglich der Übertragungskapazität vorhanden sein würden. Da ein Codierungsverfahren wie die verwendete Manchestercodierung diese bereits halbierte, blieben nur noch 9600 Bits/Sekunde für Nutz- und Steuerdaten übrig. Durch die Paketierung musste der effektive Durchsatz zusätzlich noch deutlich unter diesem Wert zu liegen kommen. Bei einseitiger Datenübertragung erreicht man daher in der Praxis nur noch Werte von 2000-3000 Bits/Sekunde. Bei gleichzeitiger Übertragung in beide Richtungen sinkt der Wert zusätzlich, da vermehrt Kollisionen auftreten. Im Beispiel von DCI, das zur Steuerung eines Didabot verwendet wird, sind im besten Fall 5 Aktualisierungen der Sensordaten bzw. 5 Pakete mit Steuersignalen pro Sekunde möglich. Für eine Echtzeitdatenverarbeitung liegt dies natürlich an der untersten Grenze.

Jedoch ist ein Arbeiten ohne Probleme möglich, wenn die Geschwindigkeit des Didabots relativ gering gehalten wird.

Mit geringer Geschwindigkeit, aber mit einer Reichweite der Funkmodems im Freien Gelände von bis zu 100 Meter, wird dem Didabot eine beachtliche Autonomie ermöglicht, die auch umfangreichere Experimente zulassen.

Ein PC mit Windows 3.1 oder Windows 95 stellten sich nicht gerade als ideale Basis für die Entwicklung des Netzprotokolles heraus. Multitasking sowie die bei älteren PC Modellen verwendete serielle Schnittstelle erschwerten ein optimales Timing. Da sich dieses meist im Bereiche von 20 bis 60 Millisekunden abspielt, führten grössere Abweichungen von den optimalen Zeiten schnell zu Effizienzverlusten.

- Als weitere Kernaufgabe sollten die sensomotorischen Daten des Roboters überwacht und verarbeitet werden können. DCI bietet hierzu die notwendigen Werkzeuge an. Grösster Vorteil von DCI besteht in der Portabilität des Anwenderprogrammes, das ohne Änderungen auch direkt auf dem Didabot selbst ausgeführt werden kann.

  DCI bietet die notwendige Funktionalität sowie Flexibilität, um einem Benutzer auch aussergewöhnliche Anwendungen zu ermöglichen. Das Problem ist jedoch, dass das Protokoll zum Lesen der sensomotorischen Daten sowie der Robotersteuerung noch nicht einem übergreifenden Konzept unterworfen wurde, das Protokoll, Daten und Steuerung umfasst. Es ist sozusagen erst der Ansatz für eine allgemeine, umfassendere Lösung, die auch zusätzliche Funktionalität ermöglicht. Für die jedoch in der Aufgabenstellung geforderten Funktionen ist DCI ausreichend instrumentiert.

# 8 BEURTEILUNG

Obwohl DDS in seiner neuen Form immer noch einige Schwachstellen besitzt, kann es als einsatzfähig betrachtet werden. Die notwendigen Funktionen gemäss der Aufgabenstellung stehen zur Verfügung und bieten ausreichende Zuverlässigkeit. Da der zeitliche Rahmen für dieses Projekt jedoch sehr begrenzt war, sollte es trotzdem nur als ein Zwischenstadium betrachtet werden, das die notwendigen Grundlagen für ein umfangreicheres, professionelles System liefert.

Ursprünglich war noch geplant, die Verarbeitung der Daten einer Videokamera auf dem Hostrechner vorzunehmen. In Anbetracht, dass das Protokoll relativ rechenintensiv ist und zusätzlich ja auch noch die Steuerung des Roboters auf dem PC durch DCI verarbeitet wird, wäre es wenig sinnvoll, diesem auch noch die Bildverarbeitung zu übertragen. Diese Entscheidung wurde ausserdem dadurch erleichtert, dass die Übertragung des Videosignals unabhängig vom Rest von einer drahtlosen Kamera zu einem speziellen Empfänger erfolgt, der das Signal dann weiter an eine Videosteckkarte des PC's gibt. Diese Komponenten lassen sich sehr gut auch in einem anderen Rechner installieren, der damit die volle Rechenleistung zur Bildverarbeitung nutzen kann. Die Resultate dieser Verarbeitung können ja über Netzwerk an den DDS Hostrechner weitervermittelt werden. Dies würde auch der Idee der dezentralen Verarbeitung der sensomotorischen Signale unterstützen, die im nächsten Kapitel noch genauer erläutert wird. Besonders zu berücksichtigen wäre dabei die Synchronisation von Bild und Sensorsignalen. Da die Ergebnisse der Bildverarbeitung bei der Generierung der neuen Steuersignale berücksichtigt werden, müsste das Bild in die Verarbeitung der zum gleichen Zeitpunkt registrierten Sensordaten eingezogen werden.

# 9 ERWEITERUNGSMÖGLICHKEITEN

DDS stellt ein umfangreiches Arbeitswerkzeug dar. Da sich je nach Einsatzgebiet unterschiedlichste Anforderungen stellen, kann ein Hilfsmittel wie DDS beliebig erweitert werden. Die Frage stellt sich nun, in welche Richtung sich eine zukünftige Entwicklung bewegen könnte, um möglichst vielen Anwendern weitere Funktionalität zur Verfügung zu stellen. Als Ausgangspunkt für diese Überlegungen können die Schwachstellen von DDS beigezogen werden, die im Kapitel 7 diskutiert wurden.

Die Netzwerkfunktionen von DDS betreffen im Grunde nur die untersten OSI-Schichten [TAN92], von der Bitübertragungsschicht bis zur Sitzungsschicht. Diese bieten dem Anwender elementarste Kommunikationsfunktionen an. DCI implementiert auf dieser Basis ein eigenes, neues Protokoll mit proprietärem Charakter. Es bietet noch zu wenig Flexibilität in der Kommunikation mit einem Roboter. Es ist zwar möglich, ohne grossen Aufwand neue Sensoren anzusteuern, doch sind diese keinem vordefiniertem Protokoll unterworfen. Sie werden individuell

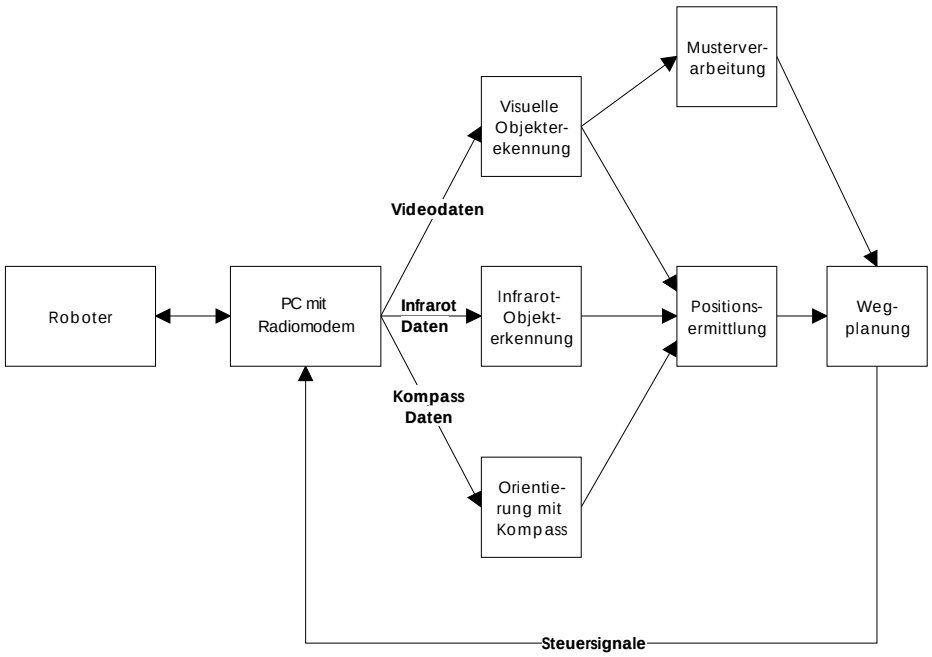


*Abbildung 9.1 - Verteilte Sensordatenverarbeitung*

vorgenommen und sind spezifisch auf eine Situation ausgelegt. Ein neues Konzept wäre nun notwendig, das die Kommunikation für alle denkbaren Sensoren exakt, aber mit genügend Spielraum, definiert. Ohne jeweils besondere Einstellungen vornehmen zu müssen, könnte über eine solche Schnittstelle jeder beliebige Roboter kontaktiert und gesteuert werden. Jeder Roboter besitzt ein individuelles Sortiment an Sensoren und Steuerungen. Diese Untereinheiten eines Roboters müssten in einzelne Klassen gruppiert werden, die vordefinierte Schnittstellen zu deren Steuerung anbieten. Zu Beginn einer Kommunikation müssten DCI dann nur Typ und Anzahl der einzelnen Sensorklassen mitgeteilt werden. Durch die definierte Schnittstelle würden diese dann automatisch unterstützt. Damit wäre es auch möglich, anderen Rechnern diese standardisierten Daten zur Verarbeitung weiterzuvermitteln. Um noch mehr

Rechenzeit zur Verfügung zu haben, könnten einzelne Untergruppen der Sensoren parallel von verschiedenen Rechnern verarbeitet werden, dessen Resultate dann schliesslich zu einem einzigen Paket von neuen Steuersignalen für den Roboter führen (Abbildung 9.1). Es existieren also in einer solchen Umgebung parallele sowie serielle Prozesse, die insgesamt einen kompletten Netzplan ergeben. Ein individuelles Programmieren der einzelnen Knoten des Netzplans wäre enorm aufwendig. Diese Knoten müssten daher aus einem immer gleichen Programmkörper bestehen, vergleichbar mit DCI, das für die gesamte Kontrolle des Datenflusses verantwortlich ist. Ein individueller Teil käme hinzu für die eigentliche Verarbeitung der Daten. Vor Gebrauch müssten den Rechnern die Knotenadressen der Eingabe- sowie der Ausgabekanal von zentraler Stelle mitgeteilt werden.

Es sind jedoch auch Erweiterungen auf dem Didabot selbst denkbar. Vor diesem Projekt existierte auf dem Roboter ein kleineres Betriebsystem namens DIBIOS, das für den Anwender eine Schnittstelle zur Steuerung der Hardware und damit auch der Sensoren zur Verfügung stellte. Zur Kommunikation wurde ein Assemblerkernel namens ERISM verwendet, der die Daten der virtuellen Ports für die Übertragung über das serielle Kabel de-/multiplexte. Diese Aufgabe wurde nun durch das Funknetz übernommen, das ebenfalls nun zum DIBIOS gehört. Dadurch ist nun das Betriebssystem des Didabots in einen provisorischen Zustand geraten, da diverse alte Funktionen, die mittels ERISM arbeiteten, zwar noch da sind, aber nun nicht mehr funktionieren und teilweise durch andere ersetzt wurden. Es wäre nun also auch sinnvoll, ein Anforderungsprofil für ein definitives Betriebssystem zu erstellen. Um nicht bei jedem neuentwickelten Roboter wieder ein komplett neues System schreiben zu müssen, sollte ein künftiges Betriebssystem einen kleinen, hardwareabhängigen Kernel besitzen (falls überhaupt nötig) und der grosse Rest in C gehalten werden, so dass jederzeit eine Portierung auf andere Prozessoren möglich ist.

# 10 LITERATURVERZEICHNIS

[ABR93]
N. Abramson, Multiple Access Communications - Foundations of emerging Technologies, IEEE Press, 1993.

[ASI92]
*Definition of the Advanced Unmanned Search System (AUSS) Sonar Characteristics,* Acoustic Systems Inc., RDT&E Division, 1992.

[CAR95]
F. Carden, *Telemetry systems design*, Artech House, 1995.

[EVR95]
H. R. Everett, *Sensors for Mobile Robots - Theory and Applications*, H. R. Everett, Peters, 1995.

[INT92]
*8XC196KC/8XC196KD User's Manual*, Order Number 272238-001, Intel Corporation, 1992.

[MCN88]
J. E. McNamara, *Technical Aspects of Data Communication*, Third Edition, Digital Press, 1988.

[ML85]
A. M. Michelson und Allen H. Leveseque, *Error-Control Techniques for Digital Communication*, Wiley-Interscience Publication, 1985.

[MS95]
M. Maris, R. Schaad, *Didabot Technical Report*, Artificial Intelligence Laboratory, University of Zurich, 1995.

[LB87]
C. A. Lynch und E. B. Brownrigg, *Packet Radio Networks - Architectures, Protocols, Technologies and Applications*, Pergamon Press, 1987.

[PFT86]
W. H. Press, B. P. Flannery, S. A. Teukolsky und W. T. Vetterling, *Numerical Recipes*, Cambridge University Press, 1986.

[RE95]
D. Regenass, *The Didabot Development System*, Artifical Intelligence Laboratory, University of Zurich, 1995.

[RM95]
*Low Power UHF Data Transceiver Module*, Radiometrix Ltd., 1995.

[RS90]
R. Rom, M. Sidi, *Multiple Access Protocols - Performance and Analysis*, Springer-Verlag, 1990.

[STR87]
O. J. Strock, *Introduction to Telemetry*, Instrument Society of America, 1987.

[TAN92]
A. S. Tanenbaum, *Computer-Netzwerke*, Wolfram's Verlag, 2. Auflage, 1992.

[VO89]
S. A. Vanstone und P. C. van Oorschot, *An Introduction to Error Correcting Codes with Applications*, Kluwer Academic Publishers, 1989.